eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA INFORMÁTICA DE GESTIÓN Y SISTEMAS DE INFORMACIÓN

TRABAJO FIN DE GRADO

CONTRIBUCIONES A UN PROYECTO OPEN SOURCE DE ÁMBITO INTERNACIONAL: GANTTPROJECT

Alumno/Alumna: Albizuri, Silguero, Oihane

Director/Directora: Pereira, Varela, Juanan

Curso: 2019-2020

Fecha: Bilbao, 18, Junio, 2020

Resumen

Castellano

Durante este proyecto se ha estudiado cómo contribuir a una aplicación de *software* libre examinando su código, arquitectura y posibles *issues*, así como el arreglo de *bugs* e implementación de nuevas funcionalidades. Para ello, se ha estudiado cada caso, diseñado alternativas válidas y funcionales, implementado, testeado y documentado, para finalmente ser defendido como proyecto en el trabajo de fin de grado.

Euskara

Proiektu honen garapenan zehar *software* libreko aplikazio batean ekarpenak nola egiten diren ikasi da. Bereziki, kodea, arkitektura eta ager daitezken *issue*-ak ikertu, *bug*-ak konpondu eta funtzionalitate berrien inplementazioa jorratu dira. Horretarako, kasuak banaka-banaka aztertu, hautabide egoki eta baliagarriak diseinatu dira, inplementatu, testeatu eta dokumentatu dira, azkenik gradu amaierako lan gisa aldeztu ahal izateko.

English

In the course of this project we will look at how to contribute to an open source software application; analyzing its code, architecture and the possible issues the project has, bug fixing and implementing new functionalities. Each issue was evaluated individually and offered a valid and functional alternative or fix. Each fix was also implemented, tested and documented in the memory of this degree's thesis.

Prefacio

El proyecto para contribuir en aplicaciones de *software* libre surge con el objetivo de introducir estudiantes de ingeniería informática en un proyecto «real» fuera del ámbito académico, de tal manera que dichos estudiantes puedan compartir su conocimientos con el resto de alumnos y aprender a defender sus acciones frente a programadores experimentados. Otro de los objetivos principales de este proyecto trata en implementar nuevas funcionalidades, así como solucionar *issues* existentes en la aplicación y solicitar su aprobación en el repositorio oficial.

El proyecto elegido como base para este TFG es *GanttProject*, una aplicación de software libre conocida por ser una de las referentes en la administración de proyectos usando el diagrama de *Gantt* y desarrollada en el lenguaje de programación *Java* aunque gran parte del código está siendo migrado a Kotlin.

Índice general

Índice de figuras

Índice de tablas

1. Introducción

Desde los años 50 hasta principios de los 70, era común que las personas que hacían uso de la informática en ámbitos universitarios y empresariales tuvieran acceso al código fuente de las aplicaciones que los programadores y desarrolladores *software* distribuían libremente.

Cuando la informática todavía no disfrutaba de un gran auge y el *software* no se consideraba un producto sino un añadido, se creaba y compartía sin ningún tipo de restricción.

Todo cambió con la llegada de los años 80, en los que las computadoras modernas comenzaron a utilizar *sistemas operativos* privativos, forzando a los usuarios a aceptar condiciones restrictivas que impedían compartir o modificar dicho *software*.

En 1983, Richard Stallman, uno de los programadores originales de *Emacs* y antiguo miembro de la comunidad *hacker* de *MIT Artificial Intelligence Laboratory*[1], anunció *GNU Project* cuyo propósito era producir un *sistema operativo* no propietario y compatible con Unix, alegando que el mundo informático se había vuelto frustrante.

Junto con la fundación del Proyecto GNU, se anunció la licencia *General Public License*[2] que se conoce extensamente hoy en día en el mundo del software libre y código abierto, y garantiza a los usuarios finales la libertad de usar, estudiar, compartir y modificar el software (Carver [2005]). A este tipo de software se le llama *Open Source Software*, tambien conocido como *OSS* (Laurent [2004]).

[1] Massachusetts Institute of Technology.
[2] Más información en: https://www.gnu.org/licenses/gpl-3.0.html.

1.1. Origen del proyecto

El proyecto fue idea de Juanan Pereira, docente del departamento de Lenguajes y Sistemas Informáticos en la Escuela de Ingeniería de Bilbao, centro dependiente de la Universidad del País Vasco (UPV/EHU).

Teniendo como objetivo que los alumnos salgan preparados al mundo laboral, propone un nuevo tipo de trabajo de fin de grado en el que los estudiantes se formen y aprendan conceptos de ingeniería software basándose en contribuciones a proyectos *open source*. Esto conlleva estudiar la arquitectura de una aplicación «real», fuera del ámbito universitario, y dar el primer paso como desarrollador y colaborador junto a otros programadores experimentados al proponer correcciones de código y nuevas implementaciones en base a los requerimientos de usuarios reales, distribuidos a lo largo del planeta —con lo que ello conlleva ya sólo en términos de localización e internacionalización del software—.

Gracias a la filosofía *open source*, es posible aprender y estudiar cómo funciona una aplicación de estas características; un producto final que bien podría ser distribuido en calidad de software privativo, pero que es constantemente mejorado por individuos de todo el mundo de manera altruista y desinteresada.

1.2. Descripción y situación del proyecto

El proyecto escogido se basa en el estudio y contribución de soluciones para la herramienta *GanttProject*, una aplicación que facilita la administración de proyectos usando diagramas *Gantt*.

Esta herramienta fue creada en 2003, y desde entonces ha estado en constante desarrollo[3] gracias a las contribuciones desinteresadas de colaboradores de *Open Source Software*.

GanttProject permite gestionar los proyectos en tareas y recursos humanos. A cada tarea se le añade un fecha de inicio y fin, así como antecesores y los recursos que se usarán para la misma. Por otro lado, la gestión de recursos se ve facilitada ya que proporciona otro gráfico con la cantidad de trabajo asignado a cada uno.

En la actualidad, GanttProject ha publicado 32 grandes versiones de la aplicación, y tiene activas 393 *issues* con diferentes *bugs* y mejoras propuestas por los usuarios, de las que se escogerán las más atractivas con el fin de implementar una solución viable.

[3] Más información sobre las versiones en: https://www.ganttproject.biz/about.

1.3. Motivaciones para la elección del proyecto

Algunas de las motivaciones principales que han llevado a la elección de este proyecto son las siguientes:

Situación actual en proyectos *open source*

Educadores del ámbito de *ingeniería del software* a menudo recalcan que sus cursos deberían ser complementados con experiencias del «mundo real» necesarias para permitir un aprendizaje efectivo de habilidades y conceptos. La realidad es que la mayoría de estudiantes jamás ha colaborado en proyectos *open source* y la idea de hacerlo les resulta complicada. Una de las razones puede ser que, en proyectos de ingeniería del software, el mundo real involucra participantes con diferentes habilidades y experiencias, y esta lleno de inconsistencias, resulta complejo y se encuentra en constante desarrollo (Sowe and Stamelos [2007]).

La mujer y su papel en el Open Source Software

Según un estudio realizado por la Comisión Europea [2019], se revela que únicamente el 2,4 % de los graduados en Tecnologías de la Información y la Comunicación (TIC) son mujeres, porcentaje que disminuye respecto al del 2011. Es aún más desolador saber que el número de mujeres que trabajan en el sector se reduce hasta el 0,6 %.

Otro estudio centrado en diversidad de género en las comunidades OSS reveló que alrededor del 11 % de los desarrolladores son mujeres: una cifra bastante más prometedora que la anterior. Sin embargo, la publicación también muestra que ese porcentaje es menor si se acota el perímetro a los 500 mejores desarrolladores de *GitHub*, siendo tan solo el 3 % representado por el género femenino. Otro dato curioso es que la tasa de aceptación de parches a código, cuando éstos reciben revisión por pares, es el 12 % superior en los casos en los que no es posible identificar el género de la desarrolladora (Balali et al. [2018]).

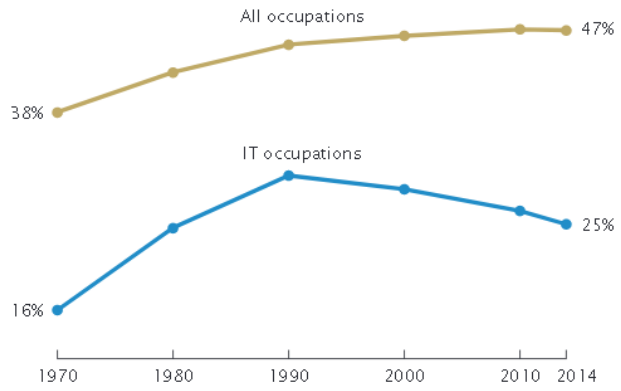


Figura 1.1: Porcentaje de mujeres en todas las ocupaciones y en el entorno TIC

La Figura 1.1 muestra la evolución de la ocupación de puestos de trabajo por mujeres (Beckhusen [2016]). La línea de color marrón muestra el total de las ocupaciones laborales, mientras que la azul muestra las ocupaciones IT (Information Technology) —tecnologias de la información en castellano—. La aparición del *Open Source Software* fue en 1983, y el pico de trabajadoras fue en 1990, por lo que no se puede descartar que tuviera un impacto positivo durante los primeros años, y luego fuera decayendo hasta la actualidad.

Aprendizaje continuo

La oportunidad de participar y contribuir a un proyecto antes de finalizar la formación bajo el paraguas de la universidad es algo realmente valorado en lo personal. Gracias a este trabajo de fin de grado, la brecha existente entre el mundo académico y el laboral se ve reducida en gran medida, y se proporcionan conocimientos adicionales muy bien valorados en un desarrollador.

La adquisición de conocimientos no termina ahí. Este tipo de proyectos obligan en cierta manera a mantenerse al día con las tecnologías, herramientas y metodologías vigentes en la actualidad.

Proyecto «real» fuera del ámbito académico y confianza en el docente

En lo que respecta a lo personal, cuando el docente Juanan me propuso personalmente este proyecto, lo veía fuera de mis posibilidades. La idea de embarcarme en un cometido de este calibre me asustaba. Fue en ese momento cuando recibí un correo electrónico suyo con la siguiente cita:

> «They always can, but they feel like they can't»

Adjunto al correo, se encontraba información sobre los proyectos OSS, estudios y la implicación femenina en los mismos. También un pequeño mensaje en el que manifestaba el apoyo y ganas de trabajar conmigo. Lo que puede parecer un pequeño gesto, fue el desencadenante que me empujó a aceptar su propuesta.

2. Planteamiento inicial

En esta sección se presentan los objetivos y las herramientas utilizadas, así como el alcance, la planificación temporal y la evaluación económica y de riesgos que se han realizado para este proyecto. También se explica, brevemente, la arquitectura que se utiliza para una mejor comprensión del funcionamiento de *GanttProject*.

2.1. Objetivos

Este proyecto consta de tres objetivos principales que se explicarán a continuación.

Estudiar como se hacen contribuciones en un proyecto OSS
Para comprender y analizar correctamente este apartado, se ha dividido en varios puntos generales:

- **Familiarizarse con las herramientas utilizadas**: Como en el siguiente punto se verá, se han utilizado una amplia variedad de herramientas como *Git* y su servicio de *hosting GitHub*, el lenguaje de programación *Java* y la aplicación de GanttProject donde se realizarán las contribuciones.
- **Estudiar el código**: Para poder solucionar o desarrollar cualquier implementación, se ha dedicado una parte importante del tiempo al estudio del código fuente de GanttProject y el sistema de construcción del mismo: IntelliJ, *Gradle*, módulos *Git*, dependencias, etc. Antes del iniciar el proyecto, se ha analizado la aplicación, y se han realizado varias contribuciones de *issues* más simples de solucionar para entender la dinámica de las contribuciones en proyectos «reales».
- **Estudiar y aplicar estilo aceptado por el desarrollador principal**: Cada proyecto consta de uno o varios desarrolladores que evalúan la calidad de las contribuciones y se cercioran de que la implementación es correcta antes de realizar un *merge* a la rama principal.

En este primer punto se resumen sólo los pasos a seguir antes de realizar ningún desarrollo. A continuación se detallará como ejecutar estas mejoras para la aplicación.

Aplicar lo estudiado y realizar las contribuciones

Una vez analizado y evaluado el modelo a seguir para realizar las contribuciones, los siguientes puntos describen cómo ejecutarlo.

- **Diseñar solución adecuada**: Como previamente se ha detallado, tras estudiar la situación actual del problema, se procede a diseñar una solución e implementación adecuada, siempre siguiendo las directrices pautadas por el desarrollador principal y tomando como ejemplo contribuciones pasadas de otros desarrolladores.
- **Testear**: Antes de publicar las contribuciones para ser valoradas por los desarrolladores oficiales del *software*, se ha testeado que las soluciones son óptimas y realizan la mejora o solución del problema que se planteaba en un inicio.
- **Defender decisiones tomadas**: Tras corroborar que es apta para ser aceptada, se ha preparado una descripción detallada con los cambios realizados y el porqué de los mismos, recopilando así los posibles diseños alternativos y testeos.
- **Documentar**: Este apartado es tan importante como el del desarrollo. Durante todo el proyecto de fin de grado se ha ido recopilando información sobre el desarrollo, problemas encontrados, diseños, soluciones, etc. con el fin de esclarecer y mostrar en lo que ha consistido.

Tras finalizar este apartado, el último de los objetivos consiste en traspasar los conocimientos adquiridos a otros alumnos o desarrolladores que deseen dar sus primeros pasos en las contribuciones de proyectos OSS.

Traspasar conocimientos a otros alumnos

Otro objetivo principal e igualmente importante que los otros dos mencionados consiste en trasladar los conocimientos adquiridos a otras personas.

La filosofía principal de los proyectos *open source* consiste en desarrollar y compartir herramientas que son valiosas para la comunidad, y de igual manera, el conocimiento ha de ser transmitido para seguir avanzando hacia una sociedad mejor informada y educada.

2.2. Herramientas utilizadas

En esta sección se detallarán todas las herramientas utilizadas en este proyecto de fin de grado. Algunas de las herramientas listadas se han utilizado previamente en otros proyectos, mientras que para poder usar otras se ha tenido que invertir tiempo extra con el fin de conseguir la formación necesaria.

- **Cacoo**: Se trata de una herramienta de dibujo que permite crear una variedad de diagramas que han sido utilizados para realizar el Diagrama EDT o Estructura de Descomposición de trabajo que posteriormente ha servido de guía para la gestión por fases del proyecto.
- **Draw.io**: Esta herramienta ha sido utilizada para la creación de diagramas UML —*Unified Modeling Language*—. Entre otros, ha resultado muy útil para la creación de diagramas arquitecturales, diagramas de clase y un nuevo tipo de diagramas en el que se han unido los componentes del interfaz gráfico con el nombre de las clases Java que los implementan —algo muy útil para los siguientes alumnos o desarrolladores que quieran empezar a desarrollar funciones o corregir *issues* en GanttProject—.
- **GanttProject**: Aplicación principal escogida en este proyecto para la administración de proyectos usando *diagramas de Gantt*.
- **GanttProject Cloud**: Versión cloud de la aplicación GanttProject que permite colaborar de manera simultánea y almacenar los proyectos en *Google Cloud Storage*.
- **Git**: Desarrollado originalmente por Linus Torvalds, creador del kernel Linux, es el sistema de control de versiones más popular hoy en día, distribuido bajo la licencias free software GPLv2[1] y LGPLv2.1[2]. Git sigue una arquitectura distribuida, lo que significa que existe una copia del repositorio y su historial completo de cambios en el sistema de cada desarrollador. También es descentralizado, lo que posibilita que los cambios puedan realizarse de manera local para posteriormente ser sincronizados con el resto de copias, sin requerir tener acceso a un servidor central Git también destaca por su buen rendimiento.
- **GitHub**: Se trata del servicio de hosting empleado para Git y usado en este proyecto. De hecho, la aplicación original GanttProject está publicada en el mismo.

[1] Más información sobre la licencia se puede encontrar en el siguiente enlace: https://www.gnu.org/licenses/old-licenses/gpl-2.0.html.

[2] También disponible información en: https://www.gnu.org/licenses/old-licenses/lgpl-2.1.html.

- **IntelliJ**: Esta herramienta es un entorno de desarrollo integrado para el desarrollo de programas informáticos. Soporta varios lenguajes de programación, entre ellos Java, y es por ello que esta herramienta será una de las principales para el proyecto.

- **Java**: Es un *lenguaje de programación interpretado* y orientado a objetos. También es uno de los lenguajes principales en el que está desarrollado GanttProject.

- **LaTeX**: Es un procesador de texto de lenguaje de marcado orientado a la creación de documentos escritos formales. Se basa en la utilización de *macros* de TeX para facilitar su uso, y es muy utilizado para la composición de artículos académicos, tesis y libros técnicos.

- **LibreOffice Calc**: Componente de LibreOffice (software libre) usado para la creación de hojas de cálculo.

- **Overleaf**: Editor de texto online basado en LaTeX. Se ha usado para la redacción de este documento.

- **SDKMAN**: *Software Development Kit* —kit de desarrollo de software en castellano— se trata de un conjunto de herramientas de desarrollo de software que permiten a un desarrollador crear una aplicación para un sistema concreto, por ejemplo ciertos paquetes de software, plataformas de hardware, entornos de trabajo, sistemas operativos —también conocidos en el mundo de la informática como Operating System (OS)—, etc.

  En esta ocasión, se ha utilizado SDKMAN! (SDK Manager). Esta herramienta está más enfocada en sistemas Unix, ofrece un *Command Line Interface (CLI)* y una API muy intuitiva con la que gestionar, por ejemplo, las versiones de *Java*, *Gradle*, etc.

- **Toggl**: Se trata de una aplicación de seguimiento de tiempo que ofrece servicios de seguimiento e informes a través de su sitio web.

2.3. Arquitectura

En este apartado se explica de forma general como es la arquitectura de la aplicación GanttProject. Más adelante (ver Sección 2.4.4) se evaluará cada *issue* de forma individual y se tratarán como proyectos independientes, pero como introducción a la gran herramienta que es GanttProject, se mostrará de forma general su funcionamiento.

GanttProject consta de varios paquetes en los que esta distribuido el código, siendo los más relevantes «ganttproject», «ganttproject-builder» y «ganttproject-tester». Estos tres paquetes forman la estructura base en la que se trabajará posteriormente.

Cada uno de los paquetes tiene una finalidad distinta:

- **ganttproject**: Código principal de la aplicación.
- **ganttproject-builder**: Clases dedicadas a la «construcción» del proyecto.
- **ganttproject-tester**: Dedicadas al testeo de la aplicación.

Se estudiarán en más profundidad en el apartado de análisis de la refactorización (ver Sección 5.1.3).

En especial, el paquete «net.sourceforge.ganttproject» que se encuentra dentro de «ganttproject» es el de más importancia. En el se encuentran otros paquetes que gestionan diferentes funciones de GanttProject —así como los paquetes «export», «gui», «resource» y «task»—, y clases como «GanttTreeTable», «GanttTask» y «GanttGraphicArea» que se estudiarán en el análisis de cada *issue* correspondiente.

A continuación se añade un diagrama de clases del paquete «ganttproject» con las clases más relevantes con el fin de conseguir una imagen general de GanttProject y su estructura.

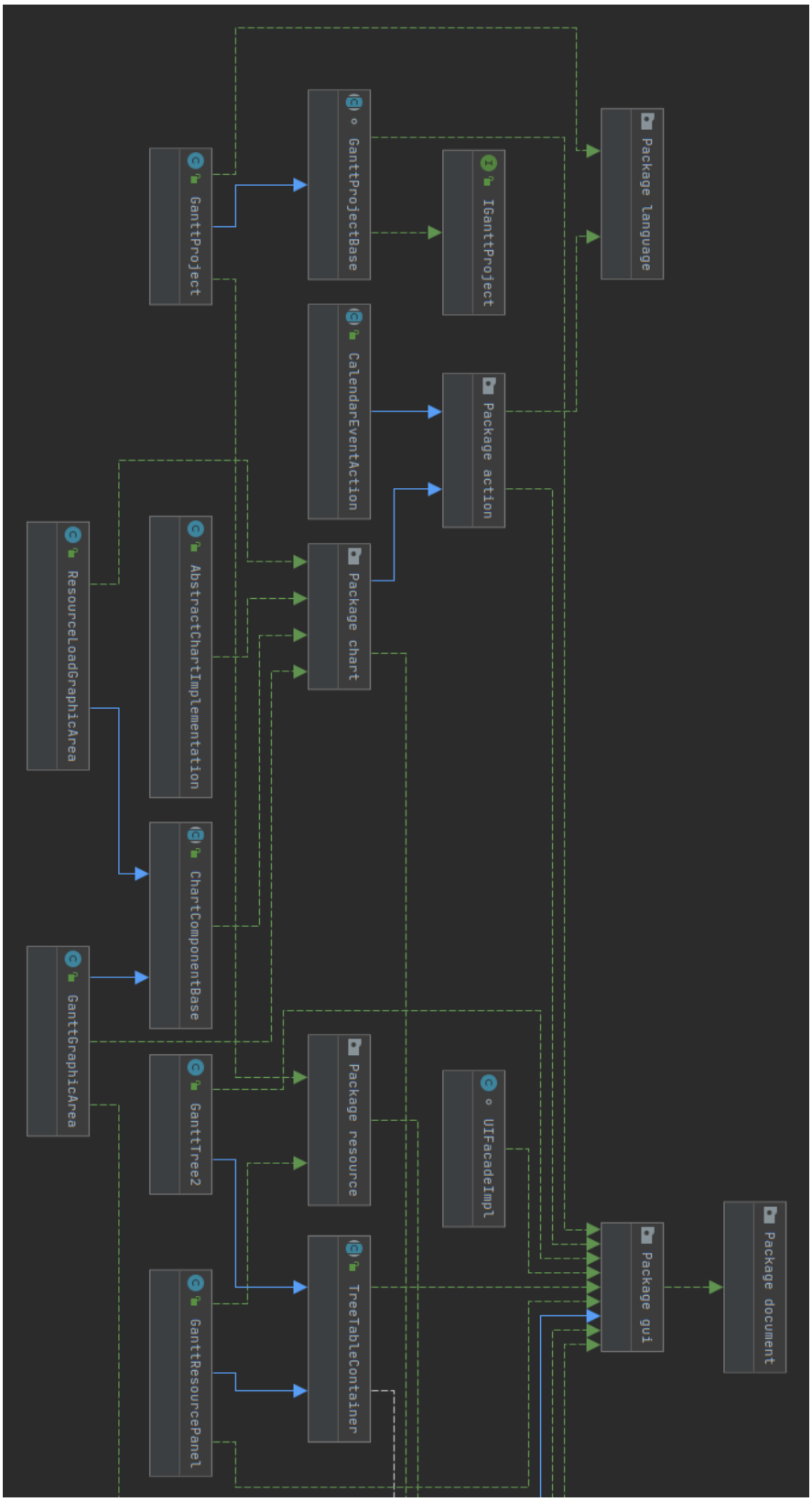


Figura 2.1: Arquitectura general de GanttProject (parte I)

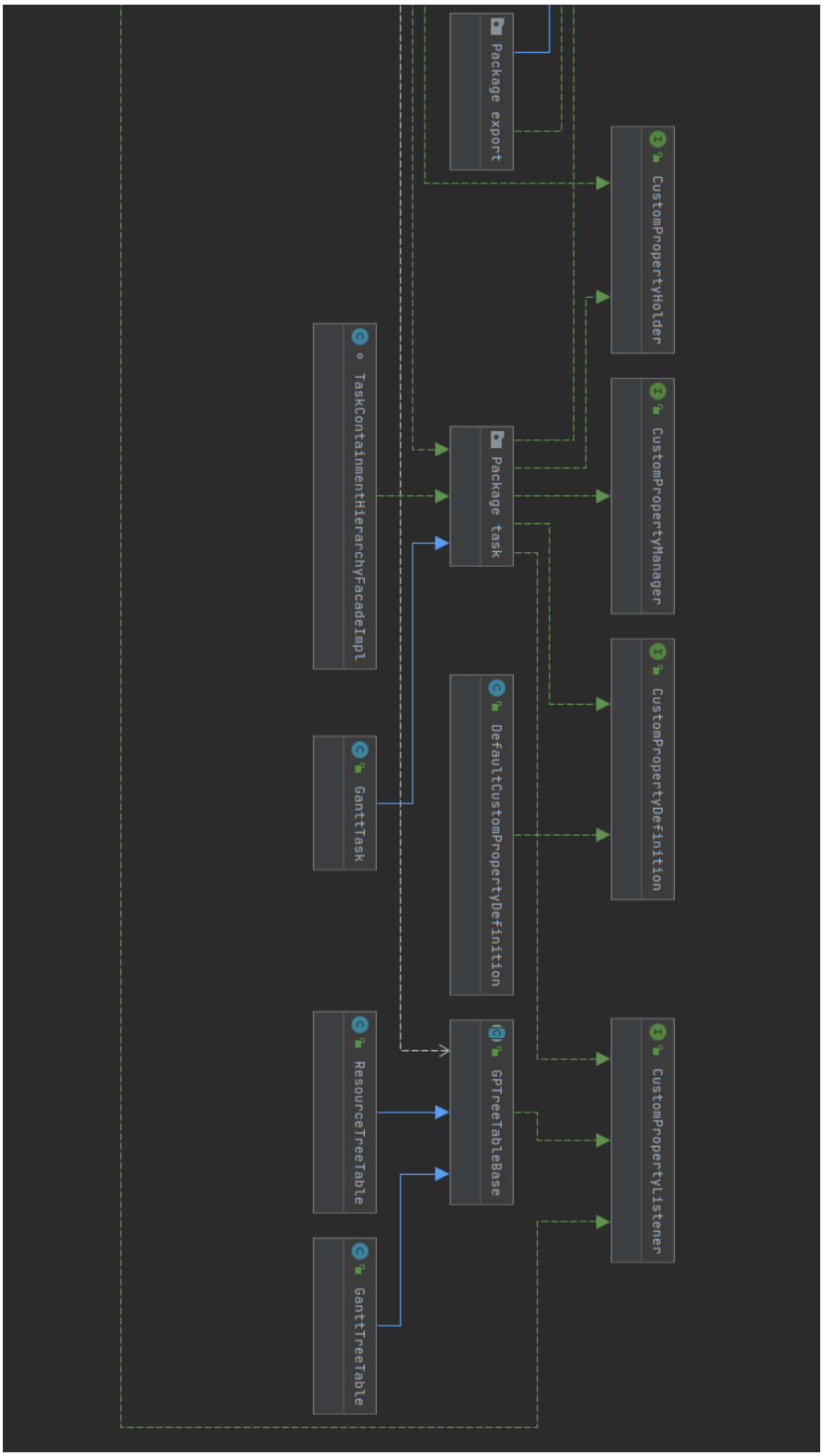


Figura 2.2: Arquitectura general de GanttProject (parte II)

2.4. Alcance

Para definir el alcance de este proyecto se ha realizado un diagrama EDT. Se han identificado ocho bloques principales: planificación y gestión, aprendizaje, captura de requisitos, análisis, diseño, implementación y desarrollo, pruebas y documentación.

1. **Planificación y gestión**: Este bloque incluye todas las tareas necesarias para desarrollar correctamente el proyecto.

2. **Aprendizaje**: Bloque al que pertenecen las tareas relacionadas con el estudio de las diferentes herramientas usadas durante el proyecto.

3. **Captura de requisitos**: Bloque que agrupa las tareas que recogen la información necesaria para reconocer las funcionalidades a realizar durante el desarrollo del proyecto.

4. **Análisis**: En este bloque se encuentran las tareas para analizar los requisitos necesarios de cada *issue* o *bug*.

5. **Diseño**: En este bloque se encuentran las tareas que se encargan de describir la posible solución al *issue* o *bug*.

6. **Implementación y desarrollo**: Este bloque alberga todas las tareas que impliquen la escritura de código en cualquier tipo de lenguaje de programación.

7. **Pruebas**: Bloque que recoge las tareas de comprobación de la correcta mecánica de las funcionalidades desarrolladas para el proyecto.

8. **Documentación**: A este último bloque pertenecen las tareas que requieran la redacción de información sobre el proyecto y se realizará a lo largo de todo el mismo.

Cada bloque del EDT es precursor al siguiente descrito en la lista, exceptuando el aprendizaje y la documentación. Las tareas de estos dos bloques se desarrollan en paralelo a las demás durante todo el proyecto.

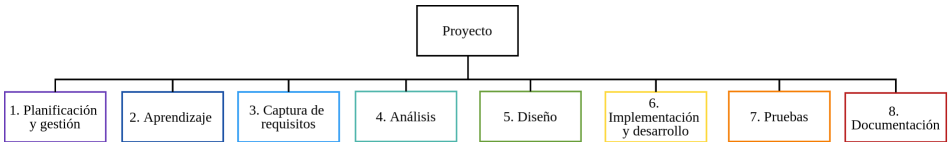


Figura 2.3: Diagrama EDT por bloques de nivel 1.

La Figura 2.3 muestra la división del proyecto en los bloques mencionados previamente.

A continuación se mostrarán detalladamente los datos sobre la planificación de cada uno de los bloques junto con las tablas y figuras correspondientes.

2.4.1. Planificación y gestión

En este apartado se muestran las tareas relacionadas con la planificación y gestión que han servido para llevar a cabo este proyecto.

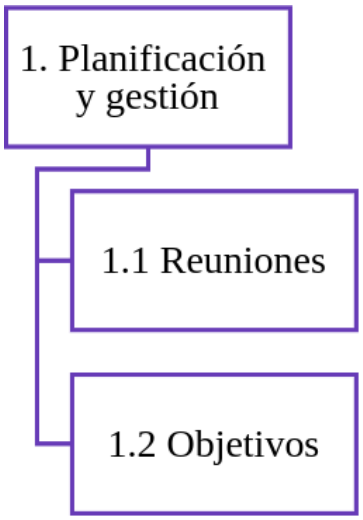


Figura 2.4: Diagrama EDT del bloque de planificación y gestión

La Figura 2.4 muestra el apartado del EDT relacionado con la planificación y gestión del proyecto. En las siguientes tablas se muestran los detalles de cada tarea.

Reuniones
Paquete de trabajo: Planificación y gestión. **Duración estimada**: 20 horas.
Descripción: Reuniones con el director del proyecto para guiar el desarrollo durante el transcurso del mismo. En las reuniones de tratan temas como problemas, modificaciones en la planificación o desarrollos de las etapas del proyecto, con el fin de tener un control periódico.
Salidas/Entregables: Anotaciones con lo acordado durante la reunión o de futuras tareas a desarrollar.
Recursos necesarios: Ordenador o bloc de notas.

Tabla 2.1: Reuniones

Objetivos
Paquete de trabajo: Planificación y gestión. **Duración estimada**: 10 horas.
Descripción: Identificar, definir y estimar la duración de las tareas a realizar durante el proyecto. Del mismo modo, fijar objetivos y metas a superar.
Salidas/Entregables: Borrador con toda la información recopilada.
Recursos necesarios: Elemento para apuntar notas.

Tabla 2.2: Objetivos

2.4.2. Aprendizaje

En el apartado de aprendizaje se ha trabajado con el fin de adquirir los conocimientos suficientes para poder realizar el trabajo de fin de grado. Para ello se han estudiado varios proyectos *Open Source Software (OSS)* similares al escogido, algunas de las herramientas con más peso y menos conocidas, así como el proyecto original GanttProject.

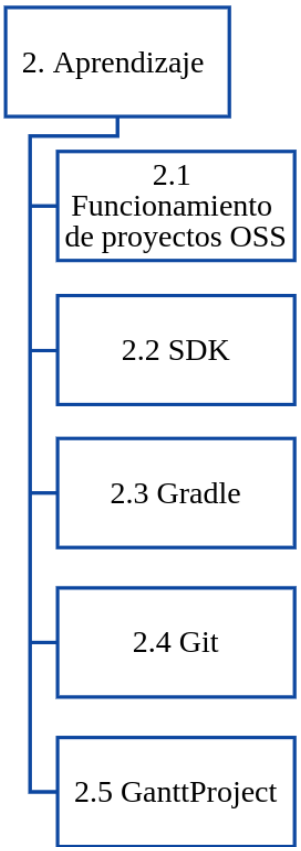


Figura 2.5: Diagrama EDT del bloque de aprendizaje

La Figura 2.5 muestra el apartado del EDT relacionado con el aprendizaje. En las siguientes tablas se muestran los detalles de cada tarea.

Funcionamiento de proyectos OSS
Paquete de trabajo: Aprendizaje. **Duración estimada**: 25 horas.
Descripción: Para poder trabajar correctamente en un proyecto open source, estudio de la mecánica de proyectos similares.
Salidas/Entregables: N/A.
Recursos necesarios: Ordenador y navegador.

Tabla 2.3: Funcionamiento de proyectos OSS

SDK
Paquete de trabajo: Aprendizaje. **Duración estimada**: 10 horas.
Descripción: Estudio y aprendizaje de como utilizar la herramienta escogida para el proyecto llamada SDKMAN!. Del mismo modo, configuración del mismo —Java, gradle, Kotlin, etc— para su utilización en el proyecto.
Salidas/Entregables: N/A.
Recursos necesarios: Ordenador y navegador.

Tabla 2.4: SDK

Gradle
Paquete de trabajo: Aprendizaje. **Duración estimada**: 10 horas.
Descripción: Estudio y aprendizaje de la herramienta de Gradle y su implicación en el proyecto.
Salidas/Entregables: N/A.
Recursos necesarios: Ordenador, navegador, código fuente de la aplicación de GanttProject y entorno de desarrollo.

Tabla 2.5: Gradle

Git
Paquete de trabajo: Aprendizaje. **Duración estimada**: 20 horas.
Descripción: Aprendizaje de *Git Branching Model* (Driessen [2010]) para la gestión del proyecto en el servidor de hosting en el que está alojado.
Salidas/Entregables: N/A.
Recursos necesarios: Ordenador, navegador y extensión Git.

Tabla 2.6: Git

GanttProject
Paquete de trabajo: Aprendizaje. **Duración estimada**: 35 horas.
Descripción: Aprendizaje de la herramienta principal a desarrollar en este projecto.
Salidas/Entregables: N/A.
Recursos necesarios: Ordenador y GanttProject.

Tabla 2.7: GanttProject

2.4.3. Captura de requisitos

Uno de los puntos importantes en cualquier proyecto es la captura de requisitos. Para ello, en las tareas a continuación se recogen los requisitos a cumplir por la aplicación de una forma general. Más adelante, y en el apartado de análisis (ver Sección 2.4.4), se recogen los requisitos necesarios enfocados en cada uno de los *issues*.

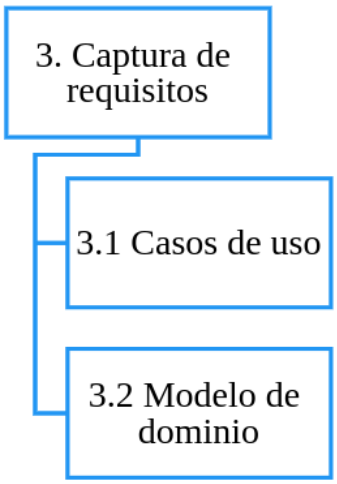


Figura 2.6: Diagrama EDT del bloque de captura de requisitos

La Figura 2.6 muestra el apartado del EDT relacionado con la captura de requisitos. En las siguientes tablas se muestran los detalles de cada tarea.

Casos de uso
Paquete de trabajo: Captura de requisitos. **Duración estimada**: 8 horas.
Descripción: Captura de requisitos de la aplicación GanttProject de una forma general.
Salidas/Entregables: Lista de requisitos.
Recursos necesarios: Ordenador o bloc de notas.

Tabla 2.8: Casos de uso

Modelo de dominio
Paquete de trabajo: Captura de requisitos. **Duración estimada**: 12 horas.
Descripción: Generar estructura general del proyecto de GanttProject.
Salidas/Entregables: Modelo de dominio.
Recursos necesarios: Ordenador o bloc de notas.

Tabla 2.9: Modelo de dominio

2.4.4. Análisis

Tal y como se ha mencionado en la Sección 2.4.3, en este bloque se profundizará en el análisis de cada *issue*. Se estudiarán más detalladamente en la Sección 4 —referente a la captura de requisitos— los motivos de la elección de los mismos.

En *Git*, un *issue* se trata de una característica de *GitHub* para el seguimiento de tareas, mejoras y *bugs* para los proyectos.

Los *issues* que a continuación se exponen han sido escogidos porque se considera que pueden ser una mejora muy valiosa que carece la aplicación.

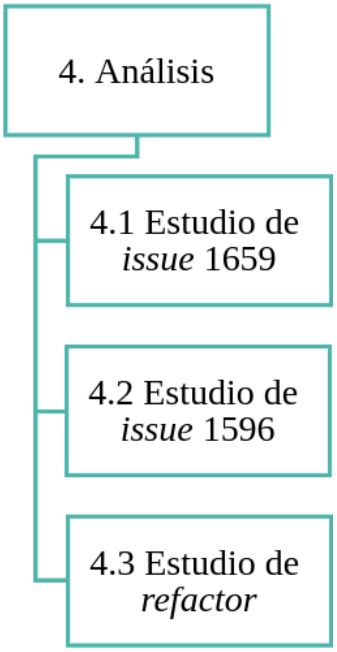


Figura 2.7: Diagrama EDT del bloque de análisis

La Figura 2.7 muestra el apartado del EDT relacionado con el análisis de este proyecto. En las siguientes tablas se muestran los detalles de cada tarea.

Estudio de issue 1659
Paquete de trabajo: Análisis. **Duración estimada**: 10 horas.
Descripción: Analizar el issue relacionado con los decimales en la columna de costes.
Salidas/Entregables: Anotaciones.
Recursos necesarios: Ordenador, navegador y GanttProject.

Tabla 2.10: Estudio de issue 1659

Estudio de issue 1596
Paquete de trabajo: Análisis. **Duración estimada**: 15 horas.
Descripción: Analizar el issue relacionado con la posibilidad de ordenar la tabla de recusos de un proyecto junto a sus respectivos gráficos.
Salidas/Entregables: Anotaciones.
Recursos necesarios: Ordenador, navegador y GanttProject.

Tabla 2.11: Estudio de issue 1596

Estudio de refactor
Paquete de trabajo: Análisis. **Duración estimada**: 7 horas.
Descripción: Analizar el código fuente por paquetes con el fin de refactorizarlo.
Salidas/Entregables: Anotaciones.
Recursos necesarios: Ordenador, navegador y GanttProject.

Tabla 2.12: Estudio de refactor

2.4.5. Diseño

En esta sección se han diseñado los casos previamente analizados (Sección 2.4.4). Para cada uno de ellos, se ha diseñado una alternativa válida y eficaz, siguiendo siempre las directrices del desarrollador principal.

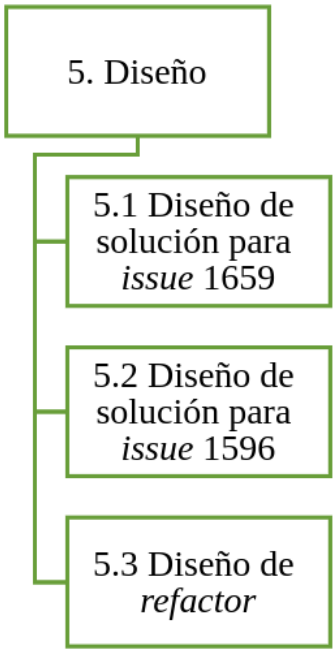


Figura 2.8: Diagrama EDT del bloque de diseño

La Figura 2.8 muestra el apartado del EDT relacionado con el diseño de este proyecto. En las siguientes tablas se muestran los detalles de cada tarea.

Diseño de solución para issue 1659
Paquete de trabajo: Diseño. **Duración estimada**: 14 horas.
Descripción: Diseño de solución para el issue relacionado con los decimales en la columna de costes.
Salidas/Entregables: Anotaciones.
Recursos necesarios: Ordenador o bloc de notas.

Tabla 2.13: Diseño de solución para issue 1659

Diseño de solución para issue 1596
Paquete de trabajo: Diseño. **Duración estimada**: 18 horas.
Descripción: Diseño de solución del issue relacionado con la posibilidad de ordenar la tabla de recursos de un proyecto junto a sus respectivos gráficos.
Salidas/Entregables: Anotaciones.
Recursos necesarios: Ordenador o bloc de notas.

Tabla 2.14: Diseño de solución para issue 1596

Diseño de refactor
Paquete de trabajo: Diseño. **Duración estimada**: 7 horas.
Descripción: Diseño de refactor y planteamiento de los procedimientos a seguir.
Salidas/Entregables: Anotaciones.
Recursos necesarios: Ordenador o bloc de notas.

Tabla 2.15: Diseño de refactor

2.4.6. Implementación y desarrollo

Uno de los puntos fuertes de este trabajo consiste en la implementación y desarrollo de los *issues* escogidos. Para ello, y teniendo en cuenta que son completamente independientes, se han desarrollado en serie, uno tras otro. En ocasiones también se han desarrollado en paralelo, ya que tras cada contribución, el desarrollador principal tiene que darle el visto bueno y eso genera cierto retraso que traspasa las limitaciones del proyecto —queda fuera del control de estas tareas—.

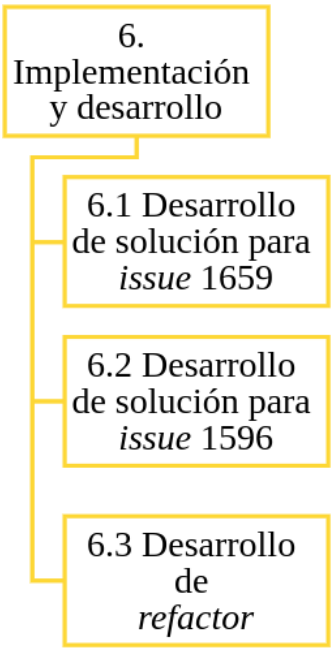


Figura 2.9: Diagrama EDT del bloque de implementación y diseño

La Figura 2.9 muestra el apartado del EDT relacionado con la implementación y desarrollo de este proyecto. En las siguientes tablas se muestran los detalles de cada tarea.

Desarrollo de solución para issue 1659
Paquete de trabajo: Implementación y desarrollo. **Duración estimada**: 40 horas.
Descripción: Desarrollo de la solución para el issue relacionado con los decimales en la columna de costes.
Salidas/Entregables: Código referente al issue 1659.
Recursos necesarios: Ordenador, entorno de desarrollo y código fuente de GanttProject.

Tabla 2.16: Desarrollo de solución para issue 1659

Desarrollo de solución para issue 1596
Paquete de trabajo: Implementación y desarrollo. **Duración estimada**: 60 horas.
Descripción: Desarrollo de la solución para el issue relacionado con la posibilidad de ordenar la tabla de recursos de un proyecto junto con sus respectivos gráficos.
Salidas/Entregables: Código referente al issue 1596.
Recursos necesarios: Ordenador, entorno de desarrollo y código fuente de GanttProject.

Tabla 2.17: Desarrollo de solución para issue 1596

Desarrollo de refactor
Paquete de trabajo: Implementación y desarrollo. **Duración estimada**: 22 horas.
Descripción: Desarrollo del refactor separado por paquetes.
Salidas/Entregables: Código referente a la refactorización.
Recursos necesarios: Ordenador, entorno de desarrollo y código fuente de GanttProject.

Tabla 2.18: Desarrollo de refactor

2.4.7. Pruebas

En esta sección se exponen las pruebas realizadas a cada *issue* con el fin de cerciorar el correcto funcionamiento antes de realizar cualquier *pull request*.

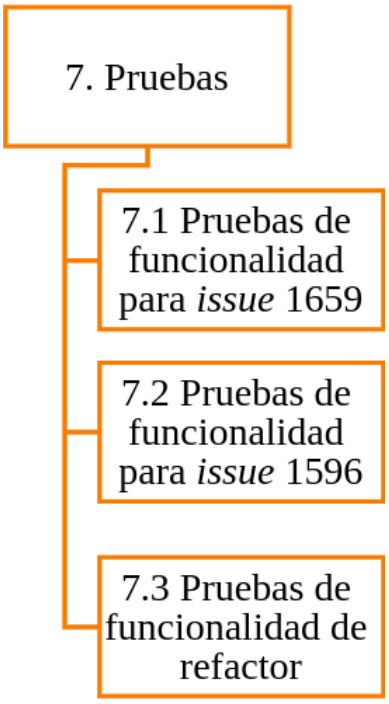


Figura 2.10: Diagrama EDT del bloque de pruebas

La Figura 2.10 muestra el apartado del EDT relacionado con las pruebas de este proyecto. En las siguientes tablas se muestran los detalles de cada tarea.

Pruebas de funcionalidad para issue 1659
Paquete de trabajo: Pruebas. **Duración estimada**: 7 horas.
Descripción: Pruebas para el correcto funcionamiento para el issue relacionado con los decimales en la columna de costes.
Salidas/Entregables: N/A.
Recursos necesarios: Ordenador, entorno de desarrollo y código fuente de GanttProject.

Tabla 2.19: Pruebas de funcionalidad para issue 1659

Pruebas de funcionalidad para issue 1596
Paquete de trabajo: Pruebas. **Duración estimada**: 10 horas.
Descripción: Pruebas para el correcto funcionamiento para el issue relacionado con la posibilidad de ordenar la tabla de recursos de un proyecto junto con sus respectivos gráficos.
Salidas/Entregables: N/A.
Recursos necesarios: Ordenador, entorno de desarrollo y código fuente de GanttProject.

Tabla 2.20: Pruebas de funcionalidad para issue 1596

Pruebas de funcionalidad de refactor
Paquete de trabajo: Pruebas. **Duración estimada**: 5 horas.
Descripción: Pruebas relacionadas con la refactorización por paquetes.
Salidas/Entregables: N/A.
Recursos necesarios: Ordenador, entorno de desarrollo y código fuente de GanttProject.

Tabla 2.21: Pruebas de funcionalidad de refactor

2.4.8. Documentación

En este último bloque se detallan las tareas relacionadas con la documentación del proyecto.

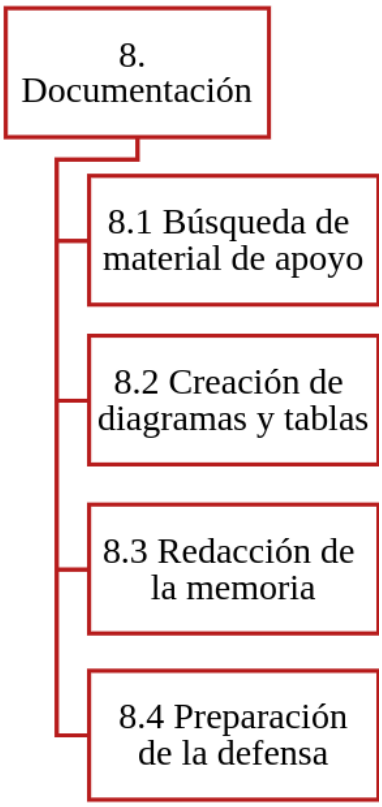


Figura 2.11: Diagrama EDT del bloque de documentación

La Figura 2.11 muestra el apartado del EDT relacionado con la documentación de este proyecto. En las siguientes tablas se muestran los detalles de cada tarea.

Búsqueda de material de apoyo
Paquete de trabajo: Documentación. **Duración estimada**: 10 horas.
Descripción: Búsqueda de artículos, libros o documentos que tengan relación con las herramientas usadas durante este proyecto, así como las relacionadas con el desarrollo del mismo.
Salidas/Entregables: Bibliografía.
Recursos necesarios: Ordenador con navegador o acceso a recursos físicos (archivo, librería, etc).

Tabla 2.22: Búsqueda de material de apoyo

Creación de diagramas y tablas
Paquete de trabajo: Documentación. **Duración estimada**: 10 horas.
Descripción: Creación de las tablas y diagramas independientes de los apartados de casos de uso y análisis y diseño que se usarán en este documento.
Salidas/Entregables: Tablas y diagramas.
Recursos necesarios: Overleaf, GanttProject y Cacoo.

Tabla 2.23: Creación de diagramas y tablas

Redacción de la memoria
Paquete de trabajo: Documentación. **Duración estimada**: 80 horas.
Descripción: Redacción de la memoria del trabajo de fin de grado.
Salidas/Entregables: Memoria del proyecto.
Recursos necesarios: Overleaf.

Tabla 2.24: Redacción de la memoria

Preparación de la defensa
Paquete de trabajo: Documentación. **Duración estimada**: 30 horas.
Descripción: Elaboración y preparación de la presentación de la defensa del proyecto.
Salidas/Entregables: Presentación en formato Power Point o PDF.
Recursos necesarios: Ordenador, proyecto y documentación.

Tabla 2.25: Preparación de la defensa

2.4.9. Resumen de la planificación realizada

En la siguiente tabla se muestra el nombre, duración y dependencias de cada una de las tareas identificadas. Como se puede observar en a figura 2.12, el número de horas totales que se estima para la duración del proyecto asciende a 495 horas.

Nombre de la tarea	Duración estimada
1. Planificación y gestión	**30:00:00**
1.1 Reuniones	20:00:00
1.2 Objetivos	10:00:00
2. Aprendizaje	**100:00:00**
2.1 Funcionamiento de proyectos OSS	25:00:00
2.2 SDK	10:00:00
2.3 Gradle	10:00:00
2.4 Git	20:00:00
2.5 GanttProject	35:00:00
3. Captura de requisitos	**20:00:00**
3.1 Casos de uso	08:00:00
3.2 Modelo de dominio	12:00:00
4. Análisis	**32:00:00**
4.1 Estudio de issue 1659	10:00:00
4.2 Estudio de issue 1596	15:00:00
4.3 Estudio de refactor	07:00:00
5. Diseño	**39:00:00**
5.1 Diseño de solución para issue 1659	14:00:00
5.2 Diseño de solución para issue 1596	18:00:00
5.3 Diseño de refactor	07:00:00
6. Implementación y desarrollo	**122:00:00**
6.1 Desarrollo de solución para issue 1659	40:00:00
6.2 Desarrollo de solución para issue 1596	60:00:00
6.3 Desarrollo de refactor	22:00:00
7. Pruebas	**22:00:00**
7.1 Pruebas de funcionalidad para issue 1659	07:00:00
7.2 Pruebas de funcionalidad para issue 1596	10:00:00
7.3 Pruebas de funcionalidad de refactor	05:00:00
8. Documentación	**130:00:00**
8.1 Búsqueda de material de apoyo	10:00:00
8.2 Creación de diagramas y tablas	10:00:00
8.3 Redacción de la memoria	80:00:00
8.4 Preparación de la defensa	30:00:00
TOTAL	**495:00:00**

Figura 2.12: Dependencias y duración de las tareas

2.5. Planificación temporal

Una vez especificadas las tareas a realizar, se ha de planear como se realizarán durante el periodo de desarrollo del proyecto. Para ello se ha realizado el diagrama de Gantt que se muestra en la Figura 2.13. En él se pueden observar los diferentes bloques mencionados en el alcance del proyecto (Sección 2.4).

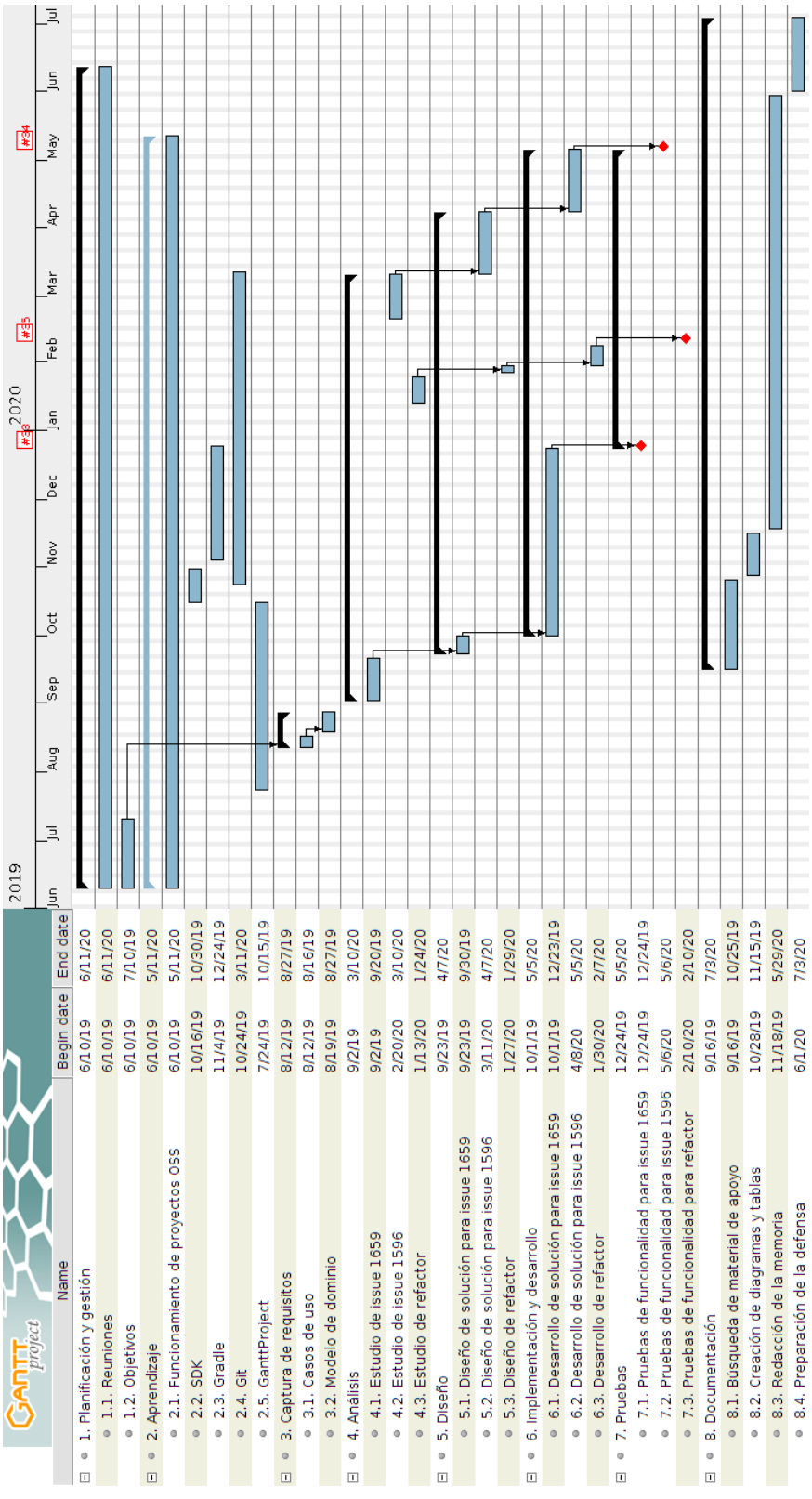


Figura 2.13: Planificación de las tareas

Durante este proyecto se quiere simular un horario de un trabajador a media jornada con un total de 28 horas semanales a partir del mes de Septiembre —hasta entonces se empleará una hora diaria (5h semanales)—. El proyecto comienza el 9 de Junio, y se ha planificado su finalización para el 5 de Junio, calculando así un total de nueve meses a media jornada para la realización del mismo. Suponiendo que un mes tiene cuatro semanas, se han realizado los siguientes cálculos:

$$\begin{aligned}\text{Horas totales} &= \text{Número de meses} \times \text{Número de semanas} \times \text{Horas de trabajo} \\ &\quad + \text{Número de meses} \times \text{Número de semanas} \times \text{Horas de trabajo} \\ &= (9 \times 4 \times 28) + (3 \times 4 \times 5) = 1068 \text{ horas}\end{aligned} \tag{2.1}$$

De las 1068 horas resultantes, se deben restar días festivos, eventos y fines de semana en los cuales no se trabajará a menos que exista un retraso con los plazos pautados. El total de días festivos en el territorio tanto en el año 2019 como en el 2020 son doce, a las que hay que sumar las vacaciones de verano.

$$\begin{aligned}\text{Horas festivas} &= (\text{Número de días festivos} \times \text{Número de horas diarias}) \\ &\quad + (\text{Número de días de vacaciones} \times \text{Número de horas diarias}) \\ &= (12 \times 4) + (30 \times 4) = 168 \text{ horas}\end{aligned} \tag{2.2}$$

Una vez calculadas las horas totales y las festivas, se procede a calcular el máximo tiempo empleable en el proyecto:

$$\begin{aligned}\text{Horas disponibles} &= \text{Horas totales} - \text{Horas festivas} \\ &= 1068 - 168 = 900 \text{ horas}\end{aligned} \tag{2.3}$$

Teniendo en cuenta que la estimación de horas totales que se pretenden invertir en el proyecto ascienden a 495 horas (Figura 2.12), por lo que se puede apreciar que no habrá dificultades para entregar el proyecto a tiempo. Además, y tal y como se ha mencionado en la implementación y desarrollo (Sección 2.4.6), algunas de las tareas se realizarán en paralelo ya que las dependencias no lo impiden.

2.6. Evaluación económica

A continuación se realiza un presupuesto con el que calcular los costes de desarrollo de un proyecto de estas magnitudes. Puesto a que se trata de un trabajo de fin de grado, no se espera obtener ningún beneficio económico, pero en caso de ser distribuido o comercializado en un futuro, se ha realizado la siguiente valoración.

2.6.1. Mano de obra

En la publicación de las tablas salariales del Boletín Oficial del Estado (BOE)[3] para el año 2020, se estipula que el salario mínimo de los titulados de 1er ciclo universitario es de 1.291,04€/mes distribuido en 14 pagas anuales. El anual neto se traduce a 18.074,56€, por lo que si se integran las pagas extra al sueldo mensual, se obtiene un sueldo neto de 1.506,21€. Una vez calculado el mensual, el salario por hora de trabajo se puede obtener con la siguiente ecuación:

$$\text{Sueldo/Hora} = \frac{\text{Sueldo mensual}}{\text{Semanas/Mes} \times \text{Días trabajo/Semana} \times \text{Horas trabajo/Día}} \tag{2.4}$$

Un mes consta por lo general de cuatro semanas, y la jornada laboral se extiende de lunes a viernes con una duración de 8 horas diarias. Una vez más, despejando la siguiente ecuación se obtendría el sueldo obtenido por hora:

$$\text{Sueldo/Hora} = \frac{1506,21}{4 \times 5 \times 8} = \frac{1506,21}{160} = 9,41 \tag{2.5}$$

Consiguiendo así un total de 9,41€/hora.

En la planificación del proyecto (Figura 2.12) se han estimado 495 horas necesarias para completar todas las tareas necesarias para este trabajo de fin de grado. De las horas mencionadas, 100 serán utilizadas para adquirir los conocimientos necesarios sobre las herramientas que se van a emplear, por lo que, y a la hora de calcular los costes por la mano de obra, estas quedarán excluidas.

$$\begin{aligned} \text{Horas de mano de obra} &= \text{Horas estimadas} - \text{Horas de aprendizaje} \\ &= 495 - 100 = 395 \text{ horas} \end{aligned} \tag{2.6}$$

[3]Documento oficial disponible en https://www.boe.es/diario_boe/txt.php?id=BOE-A-2019-14977.

Por último, el calculo del gasto económico en lo referente a la mano de obra se calcularía de la siguiente manera:

$$\begin{aligned} \text{Sueldo} &= \text{Horas trabajadas} \times \text{Sueldo/Hora} \\ &= 395 \times 9,41€ = 3716,95€ \end{aligned} \tag{2.7}$$

2.6.2. Gasto de *software*

Con el fin de seguir con la filosofía de *Open Source Software (OSS)*, se ha procedido a usar herramientas totalmente gratuitas y al alcance de cualquier desarrollador. Por lo tanto, el gasto de *software* es inexistente.

2.6.3. Gasto de *hardware*

En este apartado se procede a calcular los gastos realizados en cuanto a *hardware*. Para el desarrollo de este proyecto se ha utilizado un ordenador portátil valorado en 624€, con una vida útil de 3 a 5 años. Para realizar el cálculo de la amortización, se ha considerado que la vida útil ronda los 4 años, por lo que el cálculo se haría de la siguiente manera:

$$\text{Amortización mensual} = \frac{\text{Precio}}{\text{Vida útil}} = \frac{624}{4 \text{ años} \times 12 \text{ meses}} = \frac{624}{48} = 13€/\text{mes} \tag{2.8}$$

Una vez calculada la amortización mensual de *hardware*, y teniendo en cuenta la duración del proyecto, es posible calcular los gastos totales:

$$\begin{aligned} \text{Amortización total} &= \text{Amortización mensual} \times \text{Meses} \\ &= 13 \times 12 = 156€ \end{aligned} \tag{2.9}$$

Se puede concluir que el gasto de *software* en lo relativo a este proyecto asciende a 156€, tal y como se puede apreciar en la ecuación 2.9.

2.6.4. Gastos indirectos

En esta sección se calcularán los gastos que no tienen relación directa con el proyecto, como los gastos de luz e internet.

2.6.4.1. Gastos de luz

En la subsección 2.12 se pueden apreciar un total de 495 horas dedicadas a este proyecto, en las cuales se ha utilizado el ordenador en la totalidad de ellas. Otro factor a tener en cuenta para este cálculo es la tarifa de luz contratada en el lugar de trabajo, que en este caso el precio del kilovatio hora

(kWh) de la compañía contratada es de 0,179€/hora.

Por lo tanto, es posible calcular los gastos totales de luz invertidos en este proyecto de fin de grado:

$$\begin{aligned} \text{Gastos de luz} &= \text{Horas totales} \times \text{Tarifa/Hora} \\ &= 495 \times 0,179 = 88,605€ \end{aligned} \tag{2.10}$$

2.6.4.2. Gastos de internet

Para la elaboración de este proyecto se ha hecho un uso ininterrumpido de internet. La tarifa de internet en el lugar de trabajo asciende a 38€/mes, por lo que es posible calcular el coste total de internet de la siguiente manera:

$$\begin{aligned} \text{Gastos de internet} &= \text{Número de meses} \times \text{Tarifa/Mes} \\ &= 12 \times 38 = 456€ \end{aligned} \tag{2.11}$$

2.6.4.3. Suma de gastos indirectos

En resumen, para calcular la suma de los gastos indirectos debemos sumar los componentes de luz e internet previamente calculados:

$$\begin{aligned} \text{Gastos indirectos} &= \text{Gastos de luz} + \text{Gastos de internet} \\ &= 88,605 + 456 = 544,605€ \end{aligned} \tag{2.12}$$

2.6.5. Gastos totales

En este último apartado de la evaluación económica, se calcula el gasto total que origina la realización del proyecto.

Tipo de gasto	***Gasto ocasionado***
Mano de obra	3.716,95€
Gasto en *software*	0€
Gasto en *hardware*	156€
Gastos indirectos	544,605€
Total	**4.417,555€**

Tabla 2.26: Costes totales del proyecto

En la tabla 2.26 se muestra de una manera mas clara el total de cada apartado, y la suma del conjunto en lo que refiere al proyecto, que asciende a 4.417,55€.

2.6.6. Posibilidades de negocio

Los conocimientos que se adquieren con este trabajo de fin de grado no tienen un fin lucrativo, por lo que no se espera que pueda ser comercializado en un futuro.

En caso de enfocar los conocimientos hacia un ámbito mas educativo, se podrían impartir seminarios o publicar los conocimientos de manera esporádica con la que conseguir algún ingreso.

2.7. Gestión de riesgos

Durante la realización de un proyecto pueden surgir imprevistos que pongan en riesgo la planificación e inversión del mismo. Para ello, en este apartado se hablará de ellos, su plan de prevención y contingencia para disminuir las posibilidades de que suceda y reducir su impacto, así como la probabilidad con la que pueden darse estas situaciones.

Dentro de todos los posibles riesgos se ha hecho una división y se han separado en categorías generales que afectan en mayor o menor medida al proyecto. Del mismo modo, se ha establecido que la medida de impacto en el proyecto será regulada en base a la pérdida de horas semanales, tal que: se considerará de impacto *muy bajo* si esta por debajo de 1 hora, *bajo* si oscila entre la hora y las 2 horas, *medio* si supera las 2 horas pero no llega a 3, y *alto* si supera las 3 horas ya que equivale a un día completo para un horario de media jornada equivalente al establecido para este proyecto.

2.7.1. Enfermedad o lesión

Indisponibilidad pautada por un experto en salud, ya sea derivada por trabajo o por causas ajenas a este.

Prevención

- **Derivadas del trabajo**
  - Mantener una postura correcta a la hora de trabajar
  - Mantener la distancia correcta frente al monitor
  - Realizar descansos periódicamente
- **No derivadas del trabajo**
  - Descansar lo recomendado por los expertos en salud
  - Realizar actividad deportiva sin dañar la salud

Plan de contingencia

- Acudir al médico para diagnosticar el alcance de la lesión
- En caso de que el diagnostico sea favorable, continuar el trabajo con precaución
- En caso de no ser favorable, ceñirse al tiempo de recuperación pautado por el médico

Probabilidad

- Diagnostico favorable: Baja = 15 %
- Diagnostico desfavorable: Muy baja = 5 %

Impacto

- Diagnostico favorable: 0,15 x 2 días = 1,2 horas. Impacto bajo
- Diagnostico desfavorable: 0,05 x 7 días = 1,4 horas. Impacto bajo

2.7.2. Problemas con el equipo informático

En esta división se encuentran los problemas surgidos por acciones relacionadas con el trabajo, transporte de material o agentes externos que afectan directamente a software o hardware utilizados para el proyecto.

Prevención

- Mantener el equipo en condiciones ambientales adecuadas
- Realizar revisiones periódicamente
- Realizar copias de seguridad con bastante frecuencia y almacenarlas en diferentes localizaciones para evitar su extravío

Plan de contingencia

- **Software**: Se procederá a restaurar la última copia de seguridad guardada y se procederá a rehacer el trabajo perdido
- **Hardware**: Se procederá a remplazar el ordenador dañado por uno funcional, en el cual se instalarán todas las herramientas necesarias para reanudar el proyecto, así como a volcar la ultima copia de seguridad en el nuevo ordenador

Probabilidad

- Problema de software: Baja = 10 %
- Problema de hardware: Baja = 10 %

Impacto

- Fallo de software: 0,10 x 1 día = 0,4 horas. Impacto muy bajo
- Fallo de hardware: 0,10 x 1 día = 0,4 horas. Impacto muy bajo

2.7.3. Carencia de conocimientos tecnológicos

Resulta difícil determinar el grado de dificultad que puede surgir dado que la mitad de las herramientas que se van a utilizar para este proyecto son conocidas. Aún así, puede llegar a resultar un problema importante por lo que debe ser analizado.

Prevención

- Dedicar tiempo suficiente al aprendizaje de las nuevas herramientas
- Usar fuentes fiables en las que respaldarse para ejecutar un desarrollo limpio y sin fallos

Plan de contingencia

- Realizar búsquedas en internet sobre el problema
- Aplicar los conocimientos que se tienen sobre las herramientas para intentar solventarlo

Probabilidad

- Encontrar un problema con solución: Media/Alta = 60 %
- Encontrar un problema sin solución: Baja = 10 %

Impacto

- Problema con solución: 0,60 x 7 días = 16,8 horas. Impacto crítico
- Problema sin solución: 0,10 x 10 días = 4 horas. Impacto alto

2.7.4. Posibilidad de contratación

Pese a iniciar el proyecto con tiempo suficiente, se debe considerar que —partiendo desde una situación laboral de prácticas a media jornada—, cabe la posibilidad de incorporación a la plantilla a tiempo completo, lo que implicaría la perdida de horas disponibles para la realización del proyecto.

Prevención

- Ejecución de la planificación temporal
- Establecer un margen de al menos 2 semanas desde la finalización del proyecto hasta la fecha límite de entrega del mismo, a fin de considerar ajustes en la planificación

Plan de contingencia

- Modificación del horario establecido de media jornada diaria, disminuyendo o eliminando las horas disponibles entre semana y aumentando las horas dedicadas los fines de semana
- Modificación del cómputo de horas semanal dedicadas al desarrollo del proyecto

Probabilidad

- Contratación a jornada completa: Alta = 85 %

Impacto

- Contratación: 0,85 x 5 días = 17 horas. Impacto crítico

2.7.5. Parálisis por sobre-análisis

Relacionado con el perfeccionismo, es el término por el que se conocen aquellas situaciones excesivamente reflexivas que dificultan la toma de decisiones. Estas acciones se encuentran muy ligadas a los proyectos de desarrollo de software —considerados en algunos casos un *antipatrón de diseño* (Brown et al. [1998])— y detienen el avance del desarrollo por culpa de un análisis excesivo con el fin de evitar escoger la opción incorrecta.

Prevención

- Definir con precisión la extensión de las tareas en el alcance
- Planificación temporal correcta al inicio del proyecto

Plan de contingencia

- Dividir y trocear las tareas con el fin de simplificarlas
- Búsqueda en foros relacionados con el tema bloqueante
- Consultar al director del proyecto en busca de consejo
- Cambiar el foco a otra tarea, para después volver con otra mentalidad

Probabilidad

- Sobre-análisis: Alta = 75 %

Impacto

- Sobre-análisis: 0,75 x 1 días = 3 horas. Impacto medio

2.7.6. Pérdida de conexión a Internet

Como su propio nombre indica, son problemas causados por la pérdida de conexión a Internet que limitan la búsqueda de información e impiden la gestión de código con los servidores en los que se aloja el código fuente.

Prevención

- Contratación de una línea de Internet con suficiente velocidad
- Preferentemente usar una conexión vía Ethernet en vez de tecnología Wi-Fi

Plan de contingencia

- Cambiar de localización con el fin de disponer de una red con conexión activa a Internet, o utilizar el dispositivo móvil personal como punto de acceso —también conocido como *tethering*—

Probabilidad

- Perdida de conexión: Muy baja = 5 %

Impacto

- Perdida de conexión: 0,05 x 1 días = 0,2 horas. Impacto muy bajo

2.7.7. Estimación de tiempo

Sobreestimación, o en su defecto, desestimación del tiempo necesario para el desarrollo del proyecto.

Prevención

- Crear una buena y precisa planificación, dando tiempo suficiente al análisis del mismo

Plan de contingencia

- **Valorar más tiempo del debido**: Añadir mejoras que no estaban planteadas en un inicio
- **Valorar menos tiempo del debido**: Dar prioridades a las tareas pendientes y realizarlas de mayor a menor impacto. Ampliación del tiempo dedicado al proyecto reestructurando la planificación

Probabilidad

- Valorar más tiempo del debido: Muy baja = 5 %
- Valorar menos tiempo del debido: Baja = 15 %

Impacto

- Valorar más tiempo: 0,05 x 1 días = 0,2 horas. Impacto muy bajo
- Valorar menos tiempo: 0,15 x 5 días = 3 horas. Impacto medio

3. Antecedentes

El concepto de software de código abierto, u *Open Source Software*, surge en 1983 gracias a un movimiento social impulsado con el objetivo de obtener y garantizar las libertades que ofrecen este tipo de *software* (ver Sección 1). Desde entonces, han sido numerosas las aportaciones en proyectos de este tipo con el fin de mantener herramientas al alcance de cualquier individuo.

En concreto, y centrando el foco en GanttProject, las contribuciones han sido continuas desde sus inicios en Agosto del 2011[1] (ver Figura 3.1), pero se han visto incrementadas exponencialmente en los últimos años. Una de las razones principales se debe al impulso que ha recibido la comunidad de software libre por el interés que ha adquirido en el ámbito educativo. Las contribuciones en este tipo de *software* provienen de una amplia variedad de disciplinas incluyendo el área de ingeniería de software (Linåker et al. [2018]).

Figura 3.1: Número de contribuciones de código a GanttProject, ordenadas por año

Desde una perspectiva financiera y empresarial, el mismo estudio previamente mencionado, realizó un exhaustivo análisis que involucraba 489 proyectos de la Unión Europea. Este estudio reveló que las empresas que invertían en *Open Innovation (OI)*, es decir, innovación abierta, recibían una retribución mayor a las que no lo hacían.

[1] Histograma de contribuciones: https://github.com/bardsoftware/ganttproject/graphs/contributors

3.1. Objetivos

Colaborar en proyectos *open source* suele ser un proceso complicado, especialmente las primeras contribuciones. La literatura recomienda llevar a cabo, como primera tarea, la "comprensión de proyecto" —o *project comprehension*— que usualmente consta de varios meses de estudio del mismo para la mayoría de los desarrolladores. Pese a invertir tiempo y esfuerzo durante meses, la comprensión total del proyecto no resulta completa (Pergamenshik [2017]). La mencionada "comprensión de proyecto" suele ser una tareas que comúnmente suele ser pasada por alto en el ámbito del desarrollo de *software*.

Uno de los objetivos de este trabajo de fin de grado será el adquirir conocimientos y formación adecuada en lo referente a *Open Source Software*. Se considera fundamental para la carrera de «Grado en Ingeniería Informática de Gestión y Sistemas de Información» tener una base y experiencia contribuyendo en un proyecto real, fuera del ámbito académico.

Se estima que tener experiencia previa y guiada antes de embarcarse en un proyecto real esta muy valorada y presenta una gran oportunidad para alumnos recién matriculados sin experiencia laboral.

Dado que las contribuciones en este tipo de proyectos todavía esta en fases tempranas en el ámbito educacional, la introducción a comunidades de desarrolladores puede resultar en un aspecto significativo que haga resaltar al alumno del resto de estudiantes.

3.2. Situación actual de las contribuciones OSS

La situación actual en cuanto a las contribuciones se puede considerar mejorada en lo referente a años pasados. Como ejemplo, se ha referenciado el número de contribuciones realizadas en la plataforma *GitHub*.

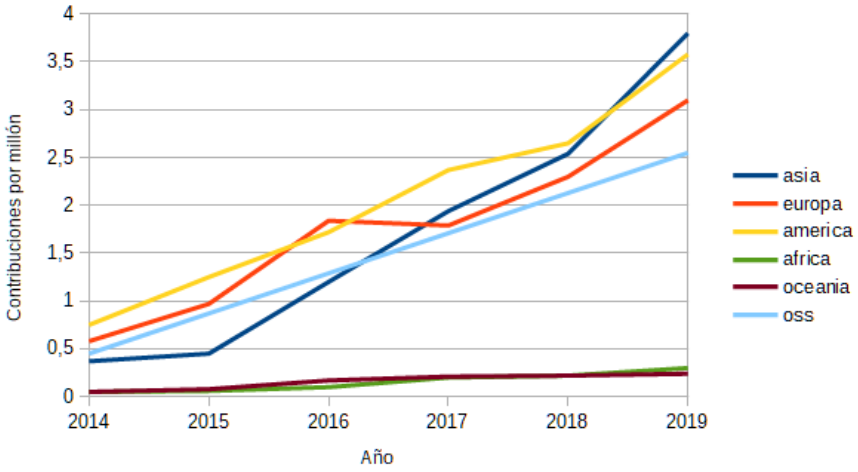


Figura 3.2: Evolución de las contribuciones en GitHub por continente. Fuente: https://octoverse.github.com/

La Figura 3.2 muestra el número de contribuciones realizadas en GitHub por continentes. Se puede apreciar una evolución favorable en todos los casos, lo que indica un aumento también en contribuciones en *repositorios* públicos con licencia *open source* (ver etiqueta "oss").

En el caso de la educación, una de las grandes ventajas que supone trabajar con *Open Source Software* es su faceta pedagógica (Kamthan [2007]) que puede ser desplegada en las aulas —con el fin de sustituir los ejemplos teóricos de libros con ejemplos «reales»— y en formación independiente —en entornos personales—.

La fundación *Linux* (Peters and Ruff [2020]) por ejemplo, impulsa este tipo de contribuciones *open source* y hace públicos los datos relacionados con su *kernel* (ver Figura 3.3)

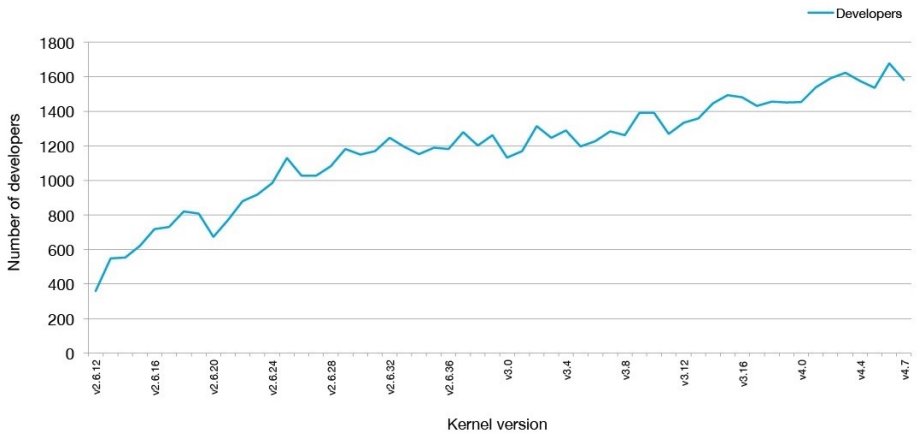


Figura 3.3: Versiones del kernel de Linux

Se puede apreciar que la evolución desde la versión del kernel 2.6.12 llamada «Woozy Numbat» y publicada el 18 de Junio de 2005, se ha mantenido un constante incremento en el numero de contribuyentes.

No solo los proyectos de gran calibre se han visto beneficiados; la diversidad de los contribuyentes también. El papel de la mujer en este tipo de proyectos (Sección 1.3) se ha visto muy influenciado e impulsado, cosa muy positiva ya que eso muestra que la barrera invisible y el miedo a las contribuciones está decayendo.

4. Captura de requisitos

En este capítulo se plasman todos los requisitos necesarios que ha de cumplir la aplicación de GanttProject. Como criterios iniciales de selección de los 3 *issues* a desarrollar, se han considerado los siguientes:

- Tareas factibles en 3 meses de trabajo.
- Viables teniendo en cuenta la inexperiencia en la base del código de GanttProject.
- Bien documentadas en la lista de *issues*.
- Asociado un test unitario o sea factible construir uno basándose en otros ejemplos.

El punto de partida será la página web de *GitHub* de la propia aplicación, en la que se muestra la lista[1] de *issues* disponibles.

Entre los preseleccionados —mostrar decimales en la tabla de tareas, ordenar recursos por nombre, detectar actualización, descargar actualización y calcular coste del recurso por tarea—, se reduce la lista teniendo en cuenta su posible utilidad y viabilidad, y se procede a capturar los requisitos y casos de uso.

Para ello, se han dividido los *issues* en diferentes apartados que se comentarán a continuación junto con los casos de uso de cada uno de ellos.

4.1. Vista general de los casos de uso

Este TFG consta de 3 contribuciones independientes al proyecto GanttProject que se detallarán a lo largo de este documento. Estos *issues* están disponibles en la página de *GitHub* de GanttProject y se caracterizan por mantener una estructura concreta. Cada *issue* tiene asociado un identificador numérico y una descripción. Para el desarrollo e implementación de una alternativa válida al "error" presente se debe crear una nueva *branch* que siga el patrón *tkt_ID_keywords*, siendo *tkt* una alusión a ticket, el *ID* igual al

[1] Lista completa de issues de GanttProject: https://github.com/bardsoftware/ganttproject/issues

identificador numérico del *issue* y *keywords* —o palabra clave— una manera rápida de identificar de que trata.

Cabe destacar que existen diferentes *issues* y no todos tratan de resolver un conflicto o error en el código:

- **Bug**: Arreglo de un error presente en la aplicación.
- **Enhancement**: Mejora de funcionalidad.
- **New functionality**: Implementación de una nueva funcionalidad.

4.1.1. Issue: tkt_1659_decimales

El primer *issue*[2] trata de una mejora para la aplicación. La pantalla principal de GanttProject consta de una tabla de tareas que se utilizará para la gestión de los proyectos. Cada tarea esta compuesta por varios atributos —nombre, fecha de inicio, fecha de finalización, costo, etc— y tiene asociado un gráfico que facilita la visualización de los datos.

En concreto, en este *issue* abierto el 8 de Abril del 2019, se pide que la columna de coste que inicialmente no muestra decimales y consta de números reales, muestre un total de dos (ver Figura 4.1).

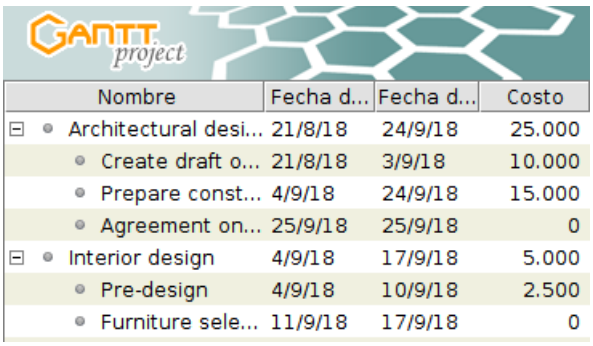


Figura 4.1: Columna de costo a mejorar

Para más información, acudir a la página web donde se detalla el *issue*.

4.1.2. Issue: tkt_1596_recursos

En esta ocasión el *issue*[3] será una mejora. Esta vez se realizará en la otra "cara" de GanttProject —la tabla de recursos— y será la opuesta a la tabla

[2] tkt_1659_decimales: https://github.com/bardsoftware/ganttproject/issues/1659

[3] tkt_1596_recursos: https://github.com/bardsoftware/ganttproject/issues/1596

de tareas mencionada previamente. Visualmente, la construcción de ambas interfaces es similar: las dos constan de una tabla y gráficos asociados a la misma. En cambio, en la tabla de recursos se trabaja con recursos humanos en vez de tareas.

Es posible ordenar dichos recursos de manera manual (arrastrándolos individualmente hasta su deseada posición), pero no es posible hacerlo de manera sencilla pulsando en la cabecera de la columna "Nombre".

El *issue* abierto el 9 de Septiembre del 2018 trata de ordenar los recursos por nombre y en orden alfabético al pulsar la cabecera "Nombre". Estos recursos se usarán para el desarrollo de las tareas de los proyectos (ver Figura 4.2).

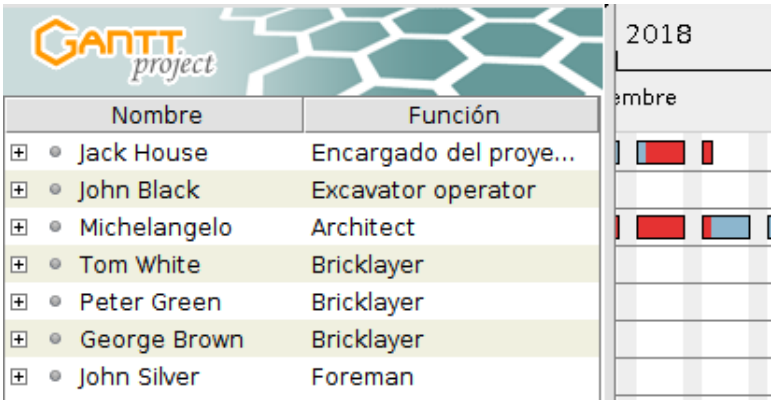


Figura 4.2: Columna de nombre a mejorar

Para más información, acudir a la página web donde se detalla el *issue*.

4.1.3. Refactorización

Uno de los puntos más valorados en los proyectos de *software* es la limpieza y claridad con la que está estructurado y desarrollado el proyecto. Estudios concluyen que los esfuerzos de refactorización reducen la complejidad del código fuente de manera notable y, por tanto, afectan directamente a la comprensión y mantenimiento del mismo positivamente (Capiluppi et al. [2004]).

'Strive to add function by deleting code.' — Jon Bentley[4]

[4]Más información sobre Jon Bentley: https://en.wikipedia.org/wiki/Jon_Bentley_(computer_scientist)

Para ello, este apartado consistirá en analizar el código y definir alternativas más sencillas, precisas, mantenibles y escalables que realicen la misma tarea que hacían previamente.

4.2. Casos de uso

En este apartado se contemplan los diferentes casos de uso para cada una de las contribuciones que se han realizado.

4.2.1. Issue: tkt_1659_decimales

La precisión en costes puede resultar un aspecto muy importante para el cálculo de los costes totales de un proyecto y para la gestión de los tiempos que pueden ser empleados para cada tarea.

4.2.2. Issue: tkt_1596_recursos

Todos los proyectos utilizan una gran variedad de recursos para su desarrollo. GanttProject gestiona estos recursos de manera que cada tarea presente en la tabla de tareas es asignada a un recurso humano. Estos recursos pueden variar desde una única persona, hasta una cantidad alta.

En los casos en que la cantidad es alta, puede resultar complicado buscar un recurso en concreto, por lo que el usuario lo encontraría con mayor facilidad si la tabla tendría opción de ser ordenada alfabéticamente en orden ascendente o descendente.

4.2.3. Refactor

Esta ultima sección no tiene impacto directo sobre el usuario, pero si sobre el resto de desarrolladores. Un código limpio y claro facilita mucho las posibles modificaciones o mejoras que se le quieran aplicar.

5. Análisis y diseño

En este apartado se analizará cada *issue* de manera independiente, realizando además los diagramas de clases y de secuencia correspondientes para su mejor comprensión.

5.1. Análisis de *issues*

En esta sección se analizará cada *issue* de manera independiente, con el fin de realizar un diseño adecuado y de calidad para cada apartado. Del mismo modo, se representará de modo visual el diseño de las clases principales que intervienen en cada apartado en los casos que se considere conveniente.

5.1.1. Issue: tkt_1659_decimales

Al inicio de esta tarea se ha realizado un boceto de la pantalla principal. Para ello, se han analizado cuidadosamente las clases de GanttProject, uniendo así los componentes que intervienen en ella y sus respectivas clases.

Esta tarea a facilitado mucho la relación entre clases a posterior, sobre todo a la hora de conseguir una idea de la estructura general del proyecto y de los elementos que lo constituyen.

En este caso, se prestará especial interés en las clases relacionadas con «GPTreeTableBase» (ver Figura 5.3) porque se considera que son las más cercanas y relacionadas con la columna de costes que se quiere mejorar, tal y como se puede apreciar en la Figura 5.1.

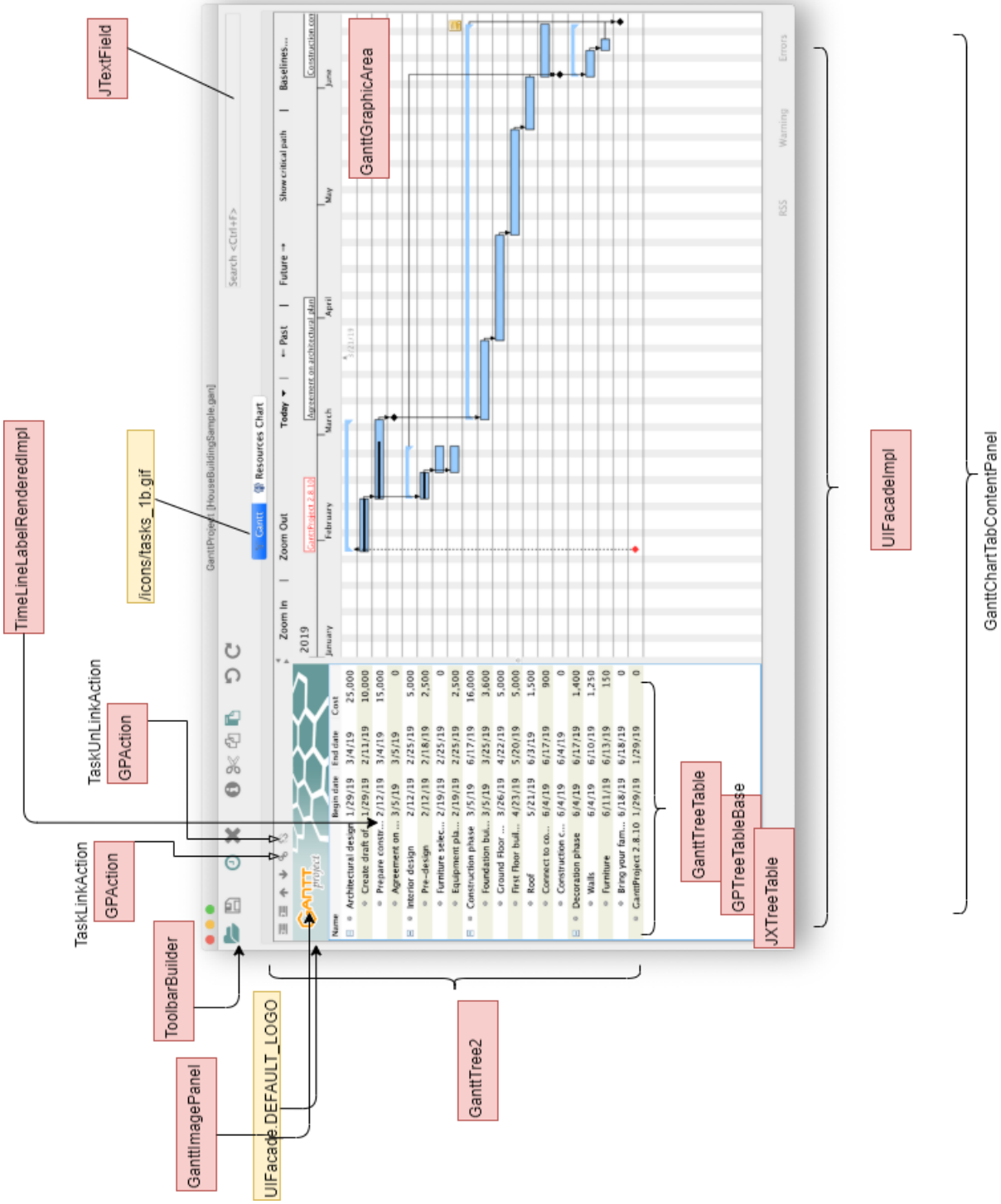


Figura 5.1: Main frame de GanttProject

5.1.2. Issue: tkt_1596_recursos

En esta segunda tarea se presta especial interés a la otra "cara" de GanttProject: la tabla de recursos. Esta tabla tiene cierta similitud visual con la tabla de tareas, pero se gestiona de manera diferente.

Siguiendo el patrón de análisis que se ha seguido para el anterior *issue*, en esta ocasión también se estudia la interfaz y se relacionan los componentes de la misma y las clases que intervienen en ella (ver Figura 5.4). De esta manera, se considera que la clase más relevante y la primera que se estudiará será «ResourceTreeTable».

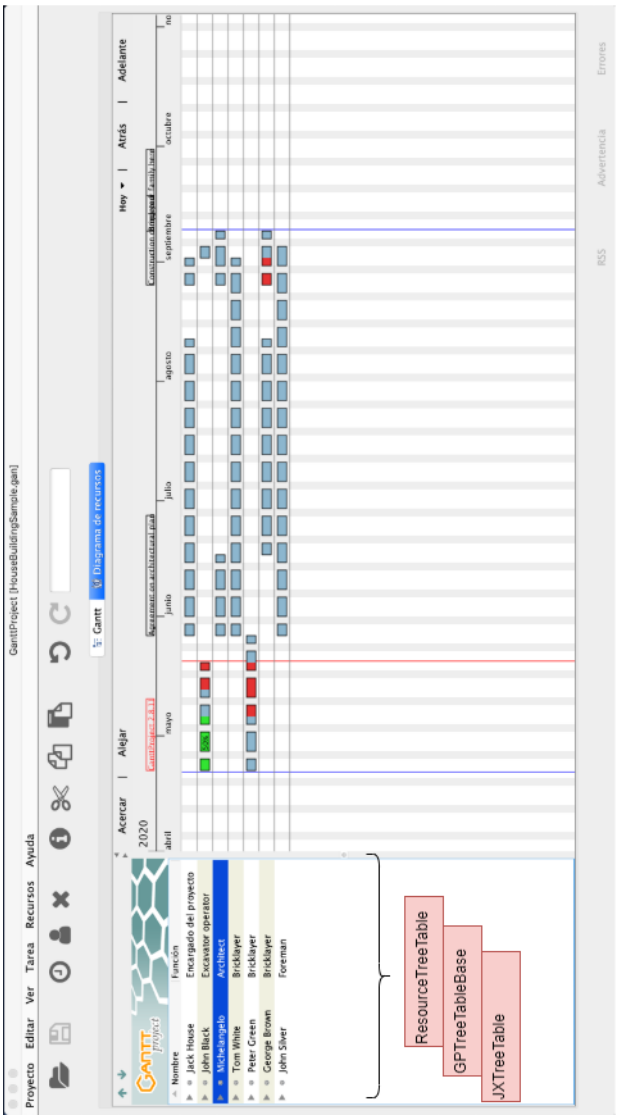


Figura 5.2: Resources frame de GanttProject

5.1.3. Refactorización

Para esta última sección —y considerando que el proyecto de GanttProject consta de una gran cantidad de clases— se ha considerado que la mejor manera de analizar el código con objetivo de refactorizarlo es hacerlo por paquetes.

Para ello, se clasifican los paquetes que se consideran de mayor importancia de la siguiente manera:

- **ganttproject**: Código principal que gestiona el núcleo de la aplicación GanttProject.
- **ganttproject-builder**: Clases dedicadas a la «construcción» del proyecto que resulta en un objeto observable y tangible. GanttProject esta formado por numerosas dependencias de *software* que se encargan de compilar y enlazar la dependencias en archivos `.class` y empaquetarlas en un archivo `.jar` ejecutable. Al ser *Java* un lenguaje compilado, se deben convertir los archivos de código fuente a *artefactos standalone* de software aislados para posteriormente ser ejecutados en un ordenador.
- **ganttproject-tester**: Clases dedicadas al testeo de las clases implementadas con el fin de detectar errores de una forma precoz y así poder solventarlas antes de publicar el código.

5.2. Diagrama de clases

Esta sección se ha dedicado al estudio de las clases presentes en GanttProject relacionadas con el *issue* estudiado en cada ocasión.

Las tareas de refactorización se van a centrar en la eliminación de código muerto —definido pero no usado— por tanto, no requieren de diagramas de clase ni secuencia.

5.2.1. Issue: tkt_1659_decimales

Siguiendo el análisis realizado previamente (Sección 5.1.1), se han estudiado las clases relacionadas con la clase «GPTreeTableBase».

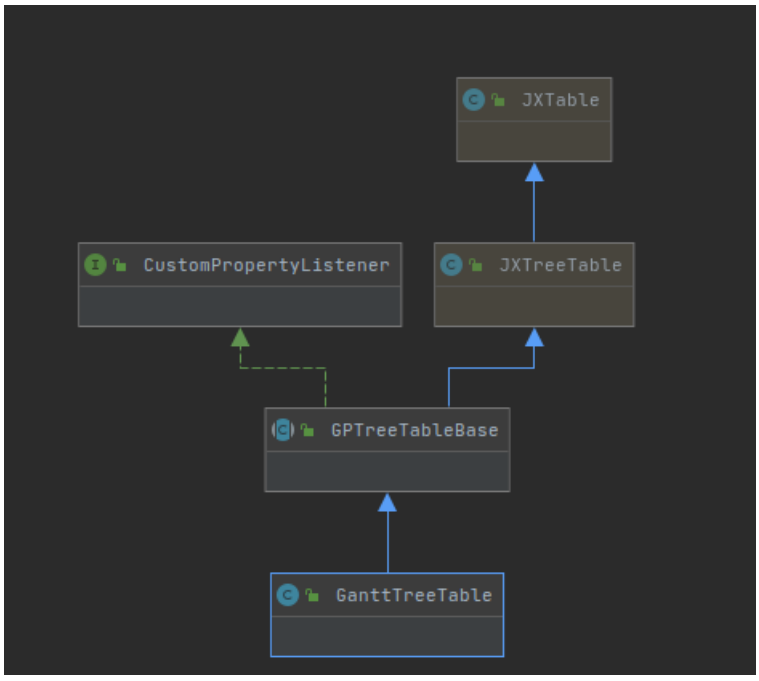


Figura 5.3: Diagrama de clases relacionado con GPTreeTableBase

La clase «GanttTreeTable» trata de una clase `.java` común, la clase en estudio «GPTreeTableBase» trata de una clase abstracta, «JXTreeTable» y «JXTable» son clases `.class` utilizadas para la creación de los ejecutables y, por último, «CustomPropertyListener» es una interfaz.

Se pueden observar cuales son las dependencias directas con la clase «GPTreeTableBase», por lo que se decide que serán tomadas en cuenta para analizar una solución a este *issue*.

5.2.2. Issue: tkt_1596_recursos

Tal y como se ha realizado en el anterior *issue*, en este también se seguirán los pasos previamente analizados. Para ello, se hará especial hincapié en la clase «ResourceTreeTable».

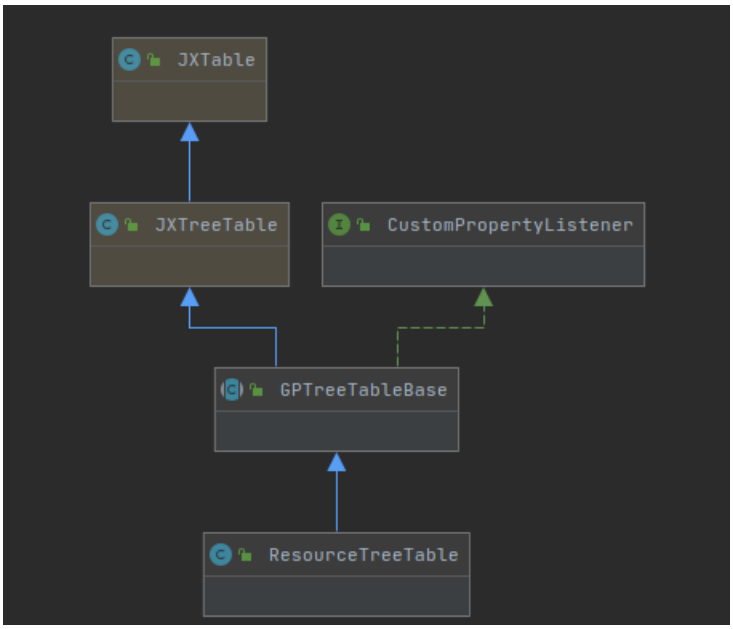


Figura 5.4: Diagrama de clases relacionado con ResourceTreeTable

El diagrama de clases para este *issue* es bastante similar al anterior. Eso se debe —tal y como se ha mencionado con anterioridad— a que se trata de la "cara" opuesta a tabla principal de tareas de GanttProject.

Cabe mencionar que la clase «ResourceTreeTable» trata de una clase común de *Java*, y el resto de componentes mantiene la misma estructura que el *issue* previamente estudiado.

5.3. Diagrama de secuencia

Los diagramas de secuencia tienen el objetivo de complementar los diagramas de clase con el fin de facilitar la lectura de las diferentes secuencias que se quieren analizar. A continuación, se estudia cada caso, aportando un diagrama de secuencia cuando se crea oportuno.

5.3.1. Issue: tkt_1659_decimales

Un examen más exhaustivo revela que el método `newTableColumnExt` puede ser el indicado para solucionar el *issue*. Para ello, se realiza el siguiente diagrama de secuencia entorno al método.

Cuando el usuario crea cualquier columna, GanttProject ejecuta el método `newTableColumnExt` perteneciente a la clase «GPTreeTableBase».

El primer paso trata de crear el *renderizador* para la tabla acorde con el tipo de celda que se esta creando. De esta función se encarga el método `createCellRenderer`.

A continuación, se crea el editor de celdas haciendo uso del método `createCellEditor`. Este comprueba si el tipo de creación para la columna se trata de una fecha u otro tipo de columna. En caso de cumplirse la primera condición, se hace uso de la clase «UiUtil» para conseguir el nuevo editor con el método `newDateCellEditor`. En caso contrario, se utiliza el editor por defecto.

Una vez especificada la configuración para la columna, se procede a crear el editor que se utilizará con el método `wrapEditor`.

El último paso trata de crear la celda. Para ello, se consigue el índice del modelo correspondiente a la columna, y si se trata de un rol o de un rol dependiente de una tarea, se procede a crear el componente de interfaz correspondiente —en este caso se trata de un `JComboBox` y sus configuraciones—. De todo ello se encarga el método `newTableColumnExt` de la clase «ResourceTreeTable».

Figura 5.5: Diagrama de secuencia relacionado con el método `newTableColumnExt` de «GPTreeTableBase»

5.3.2. Issue: tkt_1596_recursos

En esta subsección se analiza la clase «ResourceTreeTableModel», en especial el método λ asociado a una acción que será posteriormente implementado desde cero y permitirá ordenar la tabla de recursos en orden ascendente y descendente.

La implementación de este *issue* estará desarrollada dentro de la clase «ResourceTreeTable» en la que se creará una nueva interfaz `Runnable` necesaria para instanciar un hilo.

Se capturará la acción del ratón en la cabecera de la tabla de recursos y se filtrará en que estado se encuentra ordenada la misma. En caso de estar ordenada ascendentemente, se procede a cambiar el orden tal que se ordenaran los recursos de forma descendente. En caso de encontrarse desordenada o de forma descendente, se procederá a cambiarla a forma ascendente.

Tras especificar que tipo de orden será el usado, se ordenan los recursos siguiendo el orden elegido. Por cada recurso se compara la posición actual y la deseada —siguiendo el orden especificado— y se decide si el recurso tiene que ser subido —usando el método `moveUp`— o bajado —usando el método `moveDown`—.

A continuación, se mueve el recurso con la clase «HumanResourceManager» y se procede a notificar un cambio en la tabla usando `fireResourceChanged` con el fin de que sea actualizada. Se crea el evento de cambio y se procede a modificar no solo la tabla sino la interfaz también.

Se consigue el nodo del recurso junto a la información de las tareas asignadas y se aplican los cambios en la tabla de recursos en la que aparecerá en el orden definido.

Los pasos especificados previamente se aplican por cada recurso de la tabla. Una vez ordenados, se actualiza la interfaz en la que aparecerán los cambios.

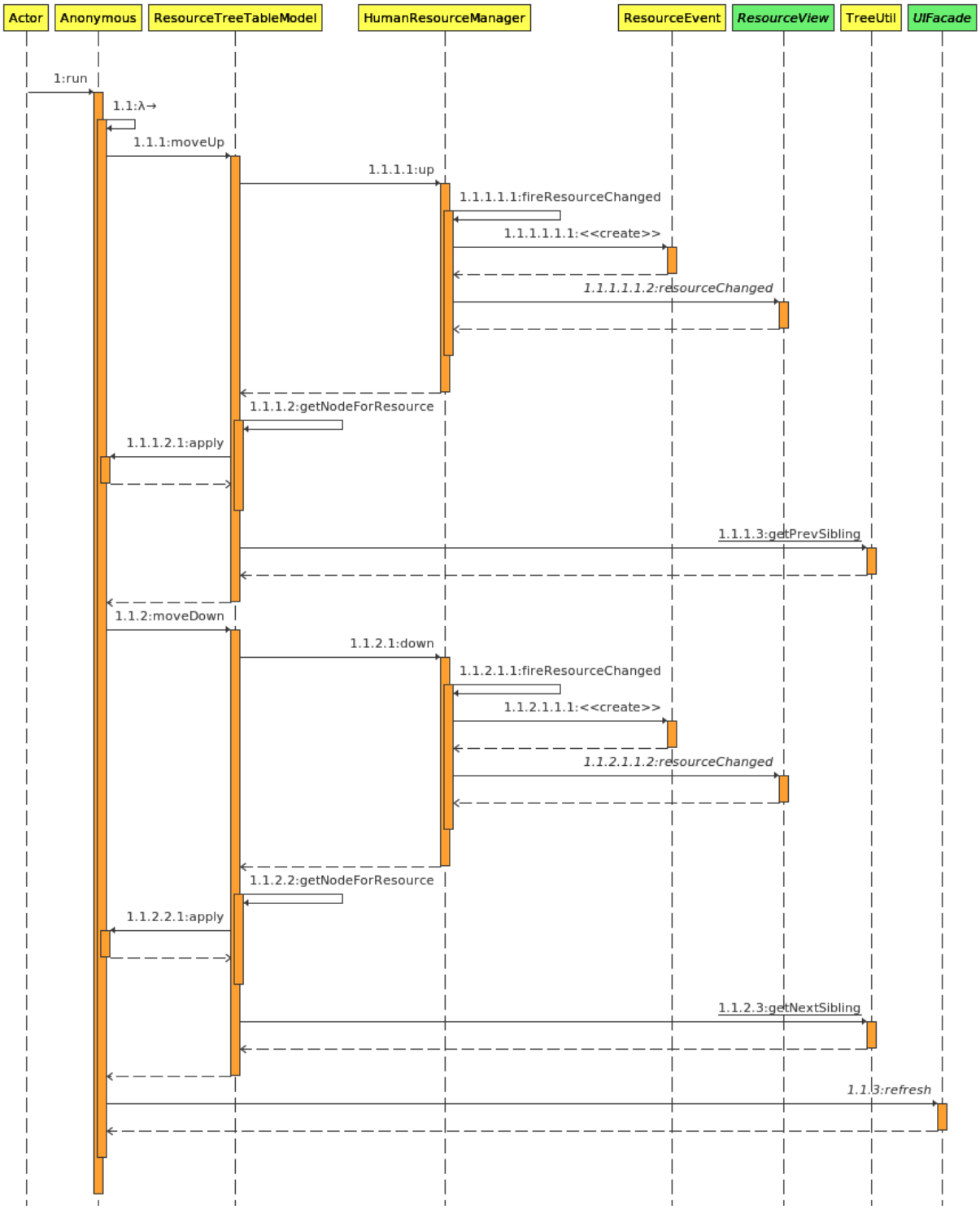


Figura 5.6: Diagrama de secuencia relacionado con el método `run` de «ResourceTreeTableModel»

6. Desarrollo

En este apartado se detallarán los pasos seguidos para el desarrollo de cada uno de los *issues* escogidos del *repositorio* oficial de GanttProject con el fin de ser mejorados e implementados.

Junto al desarrollo de cada *issue*, se detallarán los problemas encontrados y las soluciones que se han aplicado para la continuidad del mismo.

6.1. Desarrollo de Issue: tkt_1659_decimales

El objetivo de esta tarea consiste en añadir 2 decimales a la columna de costes de la tabla principal de tareas de GanttProject. Para ello, y teniendo en cuenta que se ha entablado una conversación directa[1] con el desarrollador principal, se han realizado varios *pull request* en base al *feedback* recibido.

Por cada *pull request*, el desarrollador principal realiza un *code review* y comunica al desarrollador del *issue* una serie de mejoras que implementar —también llamado *feedback*—.

6.1.1. Primer desarrollo

En un primer desarrollo, sin *feedback* del desarrollador principal Dmitry Barashev, se actúa en la clase «GPTreeTableBase». En ella, se crea una clase anidada —conocida en inglés por *nested class*— que consiste en crear una clase interna que será solo accesible desde la misma. La utilización de esta técnica de programación tiene varias ventajas[2], entre las cuales se encuentran:

- **Es una forma de agrupar lógicamente las clases que solo son usadas en un lugar**: Si una clase es solo útil para otra clase, entonces lo correcto es incorporarlo a la clase que la utilizará y mantenerlos unidos. Anidar este tipo de clases ayuda a mantener el paquete de código más simplificado.

[1] Enlace al pull request: https://github.com/bardsoftware/ganttproject/pull/1677

[2] Más información sobre las clases anidadas en: https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

- **Incrementa el encapsulamiento**: Se consideran dos clases, A y B, donde B necesita acceso a atributos de la clase A que de otra manera serían privados. Si se anida la clase B dentro de A, los atributos de A pueden mantenerse privados y la clase B tendrá acceso a ellos. Además, B puede ser ocultada del resto de clases.
- **Puede llevar a conseguir un código más fácil de leer y de mantener**: Anidar clases pequeñas dentro de clases de nivel superior coloca el código más cerca de donde es usado.

A continuación se muestra parte del código en lo referente a la clase anidada previamente mencionada:

```
1  public abstract class GPTreeTableBase extends JXTreeTable
       implements CustomPropertyListener {
2
3    // atributos privados de la clase
4
5    class DecimalRenderer extends DefaultTableCellRenderer{
6      private final DecimalFormat myFormatter = new
           DecimalFormat("#0.00");
7
8      public Component getTableCellRendererComponent (JTable
           table, Object value, boolean isSelected, boolean
           hasFocus, int row, int column) {
9
10       value = myFormatter.format((Number)value);
11       return super.getTableCellRendererComponent(table,
             value, isSelected, hasFocus, row, column);
12     }
13   }
14
15   // ...
16
17 }
```

En la línea 5 se puede apreciar que se crea una clase interna llamada «DecimalRenderer» que extiende a la clase «DefaultTableCellRenderer». El objetivo principal de «DecimalRenderer» es conseguir el valor original del campo numérico, formatearlo y enviar al renderer de la super clase (ver línea 10).

Cuando la aplicación se pone en marcha y se empiezan a generar las tablas con los valores, se comprueba si el tipo de valor recibido es «Double». Si es así, se utiliza la clase «DecimalRenderer», en caso contrario, se utilizará el *renderizador* por defecto para cada columna (ver línea 18).

```
18 if (Double.class.equals(columnClass) && columnName.equals("
       cost")) {
19   renderer = new DecimalRenderer();
20   costColumn = true;
21 } else {
22   renderer = createCellRenderer(columnClass);
23 }
```

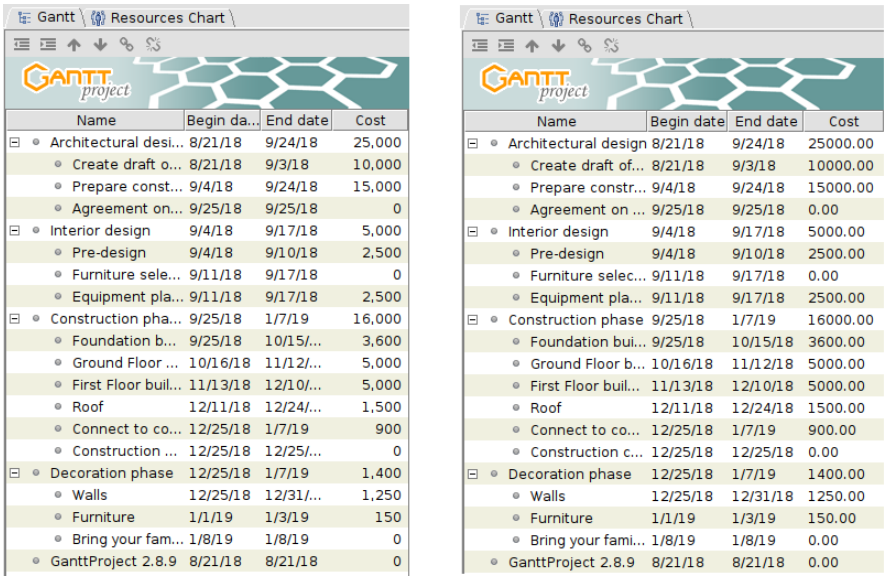


Figura 6.1: Comparación con la versión original y la implementación realizada

Una vez testeado que el código implementado funciona correctamente y no se solapa con el original, se procede a realizar un *pull request* con el fin de conseguir un *merge* del código.

6.1.2. Mejoras recibidas por parte del desarrollador principal

Una vez que el desarrollador principal realiza la evaluación del código implementado, propone varias mejoras.

Por un lado, resalta un inconveniente que no se había tenido en cuenta. Tal y como se ha planteado la filtración de los valores para usar un *renderizador* u otro, se filtran todos los valores que estén en formato «Double». En el caso del costo sería correcto ya que se trata de dinero, pero en el caso en el que se quiera usar otro tipo de medida, por ejemplo de peso, puede que no sea lo mas acertado.

Para ello, propone filtrar el nombre de la columna que será creada, y si coincide con "costo", que sea filtrada por el *renderizador* creado.

Por otro lado, otra detalle que menciona (ver línea 10) es que el formato es «locale-specific», es decir, específica del lugar. En otros países se utiliza un punto en vez de la coma para realizar la división entre los números enteros y los decimales, y en la primera implementación (ver Figura 6.1) no se ha tenido en cuenta.

6.1.3. Solución de problemas con la gestión de versiones

Durante el transcurso del desarrollo de una aplicación *open source*, se realizan una cantidad elevada de procesos *merge*, por lo que es muy recomendable mantener el código actualizado con la última versión de la *branch* master.

En el periodo de desarrollo de este *issue*, se realizaron varias integraciones en el código, por lo que al realizar la sincronización entre la *branch* master de *upstream* y la de *origin* se generaron conflictos difíciles de ser automáticamente unidos.

Figura 6.2: Conflicto de versiones

El conflicto surge con el submódulo del paquete de localización. Este paquete proviene de otro *repositorio* de *GitHub* y es completamente independiente del original.

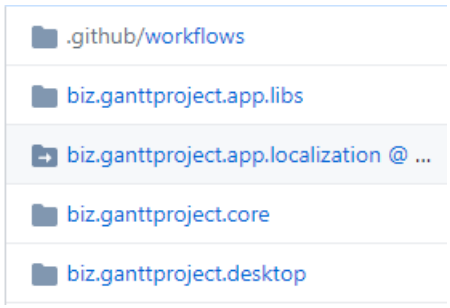


Figura 6.3: Primeros paquetes de GanttProject en GitHub

La primera acción que se toma es la de cambiar de directorio al de localización y realizar de nuevo un *pull* desde el mismo, de esa manera, se actualiza el repositorio independiente previamente mencionado.

Tras realizar dicha acción, se descargan modificaciones en 2 archivos, por lo que ahora sí, se procede a sincronizar *origin* con *upstream*.

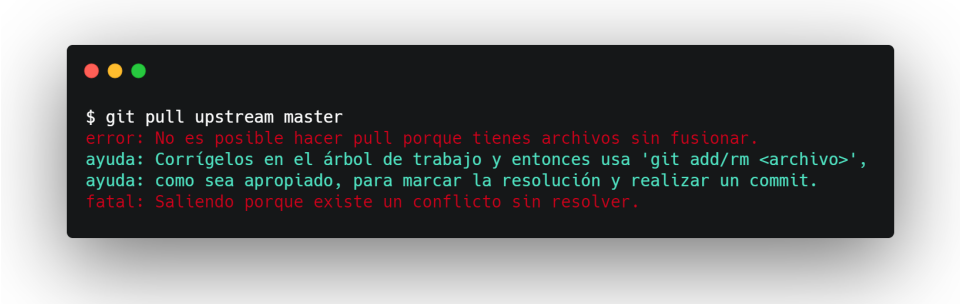


Figura 6.4: Conflicto sin resolver en curso

En esta ocasión, no es posible avanzar sin corregir los conflictos a mano, por lo que se decide utilizar la herramienta mergetool (Burns [2012]). Se realiza la configuración necesaria y preferida, y a posterior se ejecuta.

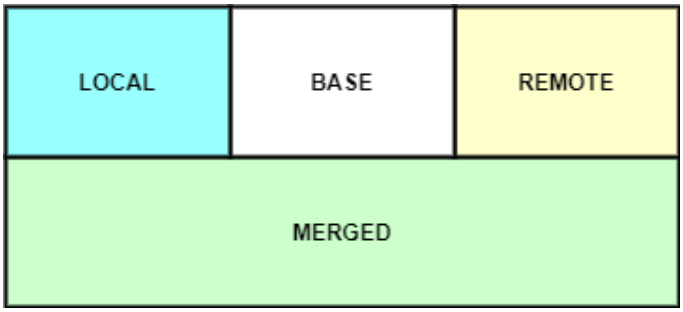


Figura 6.5: Vista simplificada tras ejecutar mergetool

La distribución en la terminal se realiza de la siguiente manera (ver Figura 6.5):

- **LOCAL**: Cambios realizados en local.
- **BASE**: Archivo en versiones posteriores, antes de ninguno de los cambios que afectan al conflicto.
- **REMOTE**: Cambios que se quieren integrar en local.
- **MERGED**: Resultado tras el *merge*.

Una vez arreglados los conflictos de manera manual, es posible proseguir con el desarrollo de los cambios propuestos por el desarrollador jefe.

6.1.4. Aplicación de cambios

A continuación, y tras resolver los conflictos de versiones, se procede a aplicar los cambios propuestos por el desarrollador jefe.

Por un lado, se modifica la clase anidada, cambiando el atributo de `myFormatter` (ver línea 6) de modo que este adquiere el formato de los decimales en base a la clase «GanttLanguage».

```
24 // ...
25 private final DecimalFormat myFormatter = new DecimalFormat
       (NumberFormat.getNumberInstance(GanttLanguage.
       getInstance().getLocale()).toString());
26 // ...
```

A continuación, se modifica el método protegido `newTableColumnExt` en el que se filtran las columnas de la tabla de tareas. Por cada columna, se comprueba si se trata de una «Double» y de si se crea con el nombre de "cost", y a continuación, se adjudica el *renderizador* correspondiente a cada columna.

6.1.5. Integración del módulo de localización

Un inconveniente de esta implementación es que la instancia del formato no será actualizada si el usuario decide cambiar el idioma desde la configuración. Para ello se deben escuchar los eventos de cambio de idioma.

Para ello, y para poder filtrar las columnas asignadas como "cost", se deben implementar cambios en la estructura planteada e incluir una integración con el modulo de localización perteneciente al submódulo de GanttProject.

Se decide avanzar hasta este punto con el *issue* ya que requiere más tiempo y recursos del esperado.

6.2. Desarrollo de Issue: tkt_1596_recursos

El objetivo de esta tarea consiste en dar la funcionalidad de ordenar los recursos de la tabla de recursos humanos en orden ascendente o descendente.

6.2.1. Primer desarrollo

En un primer desarrollo[3] y sin *feedback* del desarrollador principal, se procede a implementar mejoras a las clases «ResourceDefaultColumn» —relacionada con las columnas de la tabla— y «ResourceTreeTable» —con la tabla—.

En la clase «ResourceDefaultColumn» se crea un comparador privado y sus respectivos métodos `get` y `set` para su posterior uso e integración con el resto de código.

A diferencia de la clase «ResourceDefaultColumn», «ResourceTreeTable» contiene mucha mas implementación.

Se crea un método privado para la obtención de `UIFacade`. El objetivo principal de este, trata de gestionar la interfaz que más adelante se modificará.

A continuación se crean dos clases anidadas «AscendingNameComparator» y «DescendingNameComparator» que servirán para comparar cada uno de los nombres de la tabla de recursos y así poder ordenarla alfabéticamente.

Para poder gestionar cuándo ordenar los nombres, se añade un `Listener` a la tabla y otro al ratón. En este segundo, se filtra según la cabecera donde se haya echo click con el ratón (ver Figura 4.2), y si se ha utilizado alguna combinación de teclas, por ejemplo, pulsar «Alt», «Shift» o «Ctrl» mientras se pulsa la cabecera. De esta manera, se evitan acciones erróneas e indeseadas.

Por otro lado, se declara una nueva interfaz `Runnable` necesaria para instanciar un hilo. Estos hilos —*threads* en inglés— permiten mantener varios subprocesos ejecutándose en paralelo, algo necesario para este *issue*. En la nueva interfaz se implementará la función `run` que se ejecutará cuando el hilo se cree por primera vez.

[3]Enlace al pull request: https://github.com/bardsoftware/ganttproject/pull/1740

```
27 getUiFacade().getUndoManager().undoableEdit(GanttLanguage.
       getInstance().getText("task.sort"), new Runnable() {
28   @Override
29   public void run() {
30     // ...
31   }
32 }
```

Se procede a implementar el método `run`. Para ello, se filtra la columna en la que se actuará, que en este caso se trata de la columna por defecto de la tabla de recursos. Si la columna en la que se actúa ya está ordenada de modo ascendente, se le adjudica el comparador en modo descendente. En caso contrario, se le adjudicará el comparador en modo ascendente.

A continuación se crea una lista compuesta de `HumanResource`, y se recorrerán todos los recursos añadiéndolos de forma ordenada gracias al método `sort` de `ArrayList`.

Una vez se ha creado la lista ordenada, se compara con la lista original en la que se encuentran los recursos. Se comparan los índices en cada una de las tablas, y se calcula la cantidad de "pasos" que necesita ese recurso para colocarse en la posición correcta.

```
33 int delta = idxs - idxo;
34 for(int i = 0; i < Math.abs(delta); i++){
35   if (idxs < idxo) {
36     myResourceTreeModel.moveUp(h);
37   } else {
38     myResourceTreeModel.moveDown(h);
39   }
40 }
```

El fragmento de código aquí mostrado moverá el recurso humano `h`, `delta` pasos hasta colocarlo en la posición adecuada en la tabla de recursos humanos llamada `myResourceTreeModel`.

Por último, se añade una nueva clase anidada llamada «ModelListener» para la gestión de los eventos de cambio que se producen en la tabla.

6.2.2. Solución de problemas con Kotlin

Durante el proceso de cada desarrollo, se debe mantener el proyecto actualizado a la última versión. Ya que se trata de un proyecto en constante desarrollo, se ha encontrado de nuevo con otra actualización que ha causado problemas.

En este caso, se ha actualizado la versión de *Kotlin*. Para ello, y gracias a la herramienta de IntelliJ que notifica cuando se necesitan instalar nuevas versiones, se ha procedido a la actualización de la misma. A continuación, y sin modificar la configuración aplicada en un principio, se ha procedido a ejecutar el programa. La sorpresa surge cuando la aplicación no se ejecuta por un problema con *Kotlin*.

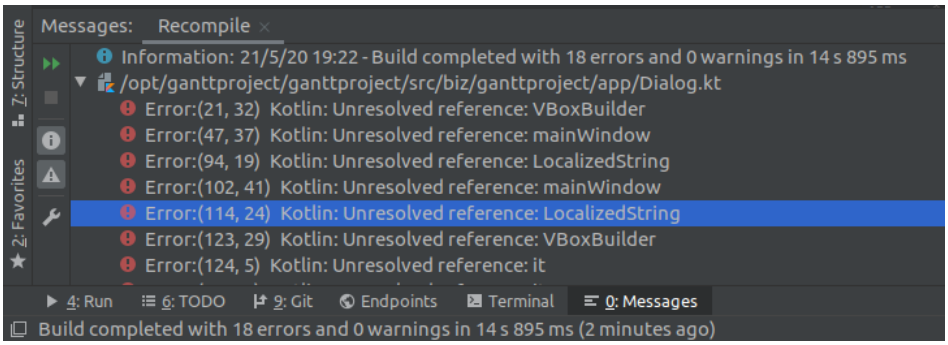


Figura 6.6: Error en Kotlin tras actualización

Precisamente, todos los errores indicados apuntan a un fallo desconocido en el archivo `Dialog.kt`. Este archivo tiene la extensión `.kt` conocida por ser la extensión de *Kotlin*.

Los errores que surgen coinciden todos en una misma línea:

```
import javafx.embed.swing.JFXPanel
```

Hasta el momento, la configuración adecuada de *Java* ha sido la versión 11.0.6 de Oracle, y *Kotlin* en su versión 1.3.11, tal y como se puede apreciar en la figura 6.7.

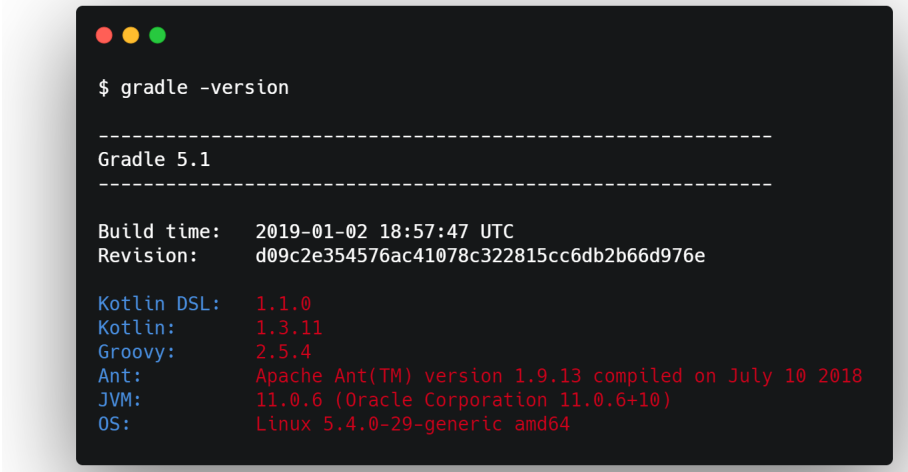


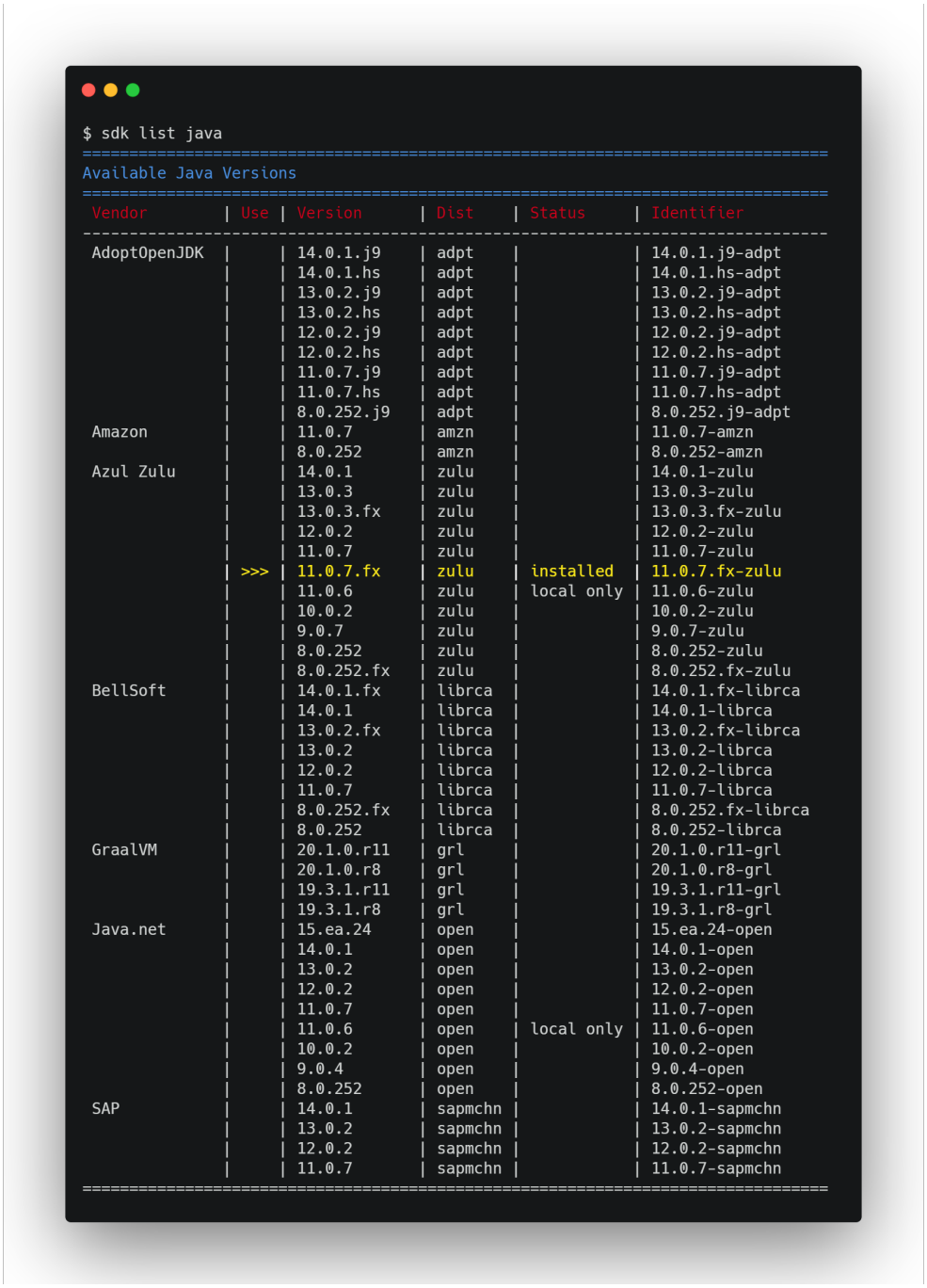
Figura 6.7: Configuración inicial

Tras varios cambios en la configuración e incontables intentos, un post[4] publicado en la página de ayuda de GanttProject revela un cambio importante: las dependencias de JavaFX han sido movidas de Gluon, por lo que es necesario utilizar una versión de *Java* que las integre.

Para ello, se cambia la versión de Java a una que integre estos módulos. Se escoge una versión de Zulu FX[5], ya que Oracle —el proveedor principal de Java— no las ofrece en su versión *Long Term Support (LTS)*.

[4]Post de ayuda: https://help.ganttproject.biz/t/unresolved-reference-javafx-embed-when-building-latest-version/1959

[5]Existen diferentes versiones de Java de distintos proveedores y cada uno de ellos integra diferentes módulos



```
$ sdk list java
================================================================================
Available Java Versions
================================================================================
 Vendor        | Use | Version      | Dist    | Status     | Identifier
--------------------------------------------------------------------------------
 AdoptOpenJDK  |     | 14.0.1.j9    | adpt    |            | 14.0.1.j9-adpt
               |     | 14.0.1.hs    | adpt    |            | 14.0.1.hs-adpt
               |     | 13.0.2.j9    | adpt    |            | 13.0.2.j9-adpt
               |     | 13.0.2.hs    | adpt    |            | 13.0.2.hs-adpt
               |     | 12.0.2.j9    | adpt    |            | 12.0.2.j9-adpt
               |     | 12.0.2.hs    | adpt    |            | 12.0.2.hs-adpt
               |     | 11.0.7.j9    | adpt    |            | 11.0.7.j9-adpt
               |     | 11.0.7.hs    | adpt    |            | 11.0.7.hs-adpt
               |     | 8.0.252.j9   | adpt    |            | 8.0.252.j9-adpt
 Amazon        |     | 11.0.7       | amzn    |            | 11.0.7-amzn
               |     | 8.0.252      | amzn    |            | 8.0.252-amzn
 Azul Zulu     |     | 14.0.1       | zulu    |            | 14.0.1-zulu
               |     | 13.0.3       | zulu    |            | 13.0.3-zulu
               |     | 13.0.3.fx    | zulu    |            | 13.0.3.fx-zulu
               |     | 12.0.2       | zulu    |            | 12.0.2-zulu
               |     | 11.0.7       | zulu    |            | 11.0.7-zulu
               | >>> | 11.0.7.fx    | zulu    | installed  | 11.0.7.fx-zulu
               |     | 11.0.6       | zulu    | local only | 11.0.6-zulu
               |     | 10.0.2       | zulu    |            | 10.0.2-zulu
               |     | 9.0.7        | zulu    |            | 9.0.7-zulu
               |     | 8.0.252      | zulu    |            | 8.0.252-zulu
               |     | 8.0.252.fx   | zulu    |            | 8.0.252.fx-zulu
 BellSoft      |     | 14.0.1.fx    | librca  |            | 14.0.1.fx-librca
               |     | 14.0.1       | librca  |            | 14.0.1-librca
               |     | 13.0.2.fx    | librca  |            | 13.0.2.fx-librca
               |     | 13.0.2       | librca  |            | 13.0.2-librca
               |     | 12.0.2       | librca  |            | 12.0.2-librca
               |     | 11.0.7       | librca  |            | 11.0.7-librca
               |     | 8.0.252.fx   | librca  |            | 8.0.252.fx-librca
               |     | 8.0.252      | librca  |            | 8.0.252-librca
 GraalVM       |     | 20.1.0.r11   | grl     |            | 20.1.0.r11-grl
               |     | 20.1.0.r8    | grl     |            | 20.1.0.r8-grl
               |     | 19.3.1.r11   | grl     |            | 19.3.1.r11-grl
               |     | 19.3.1.r8    | grl     |            | 19.3.1.r8-grl
 Java.net      |     | 15.ea.24     | open    |            | 15.ea.24-open
               |     | 14.0.1       | open    |            | 14.0.1-open
               |     | 13.0.2       | open    |            | 13.0.2-open
               |     | 12.0.2       | open    |            | 12.0.2-open
               |     | 11.0.7       | open    |            | 11.0.7-open
               |     | 11.0.6       | open    | local only | 11.0.6-open
               |     | 10.0.2       | open    |            | 10.0.2-open
               |     | 9.0.4        | open    |            | 9.0.4-open
               |     | 8.0.252      | open    |            | 8.0.252-open
 SAP           |     | 14.0.1       | sapmchn |            | 14.0.1-sapmchn
               |     | 13.0.2       | sapmchn |            | 13.0.2-sapmchn
               |     | 12.0.2       | sapmchn |            | 12.0.2-sapmchn
               |     | 11.0.7       | sapmchn |            | 11.0.7-sapmchn
================================================================================
```

Figura 6.8: Versiones disponibles de java

Gracias a la herramienta SDKMAN!, es posible ver de forma ordenada y rápida qué tipo de versiones Java están disponibles. Como se puede apreciar en la figura 6.8, hay una gran cantidad de opciones, pero se escoge la

versión 11.0.7.fx-zulu, recomendada por el desarrollador principal en el post de ayuda.

Pese al cambio de *Java* y la actualización de *Kotlin*, no es posible ejecutar de nuevo la aplicación.

Tras realizar indexaciones del proyecto, limpiar la caché de IntelliJ y realizar numerosas compilaciones, no se consigue ejecutar correctamente la aplicación, por lo que se procede a comentarlo con el director del proyecto Juanan Pereira, y se decide usar una de las aplicaciones de escritorio remoto llamadas «ngrok[6]» o «remote.it[7]».

Mientras se realiza la instalación de esta última herramienta, se actualiza también la versión de *Gradle* a una superior, de modo que la configuración final quedaría como se puede ver en la figura 6.9.

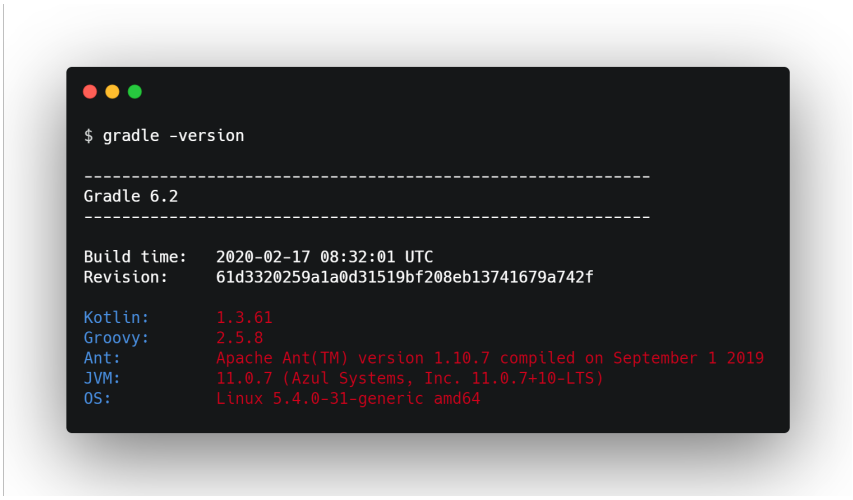


Figura 6.9: Configuración posterior

Antes de realizar la conexión con el director del proyecto, se procede a dejar abierto IntelliJ con el fin de facilitar el trabajo, y se ejecuta una última vez. En esta ocasión, el programa se ejecuta sin ninguna dificultad y es posible comprobar que el desarrollo del *issue* es correcto.

Para finalizar este desarrollo, se limpia el código, se adapta a las exigencias del desarrollador principal y se realiza el *pull request*.

» [6]ngrok: https://ngrok.com/
» [7]remote.it: https://remote.it/

6.2.3. Contestación del desarrollador principal

Cuando se recibe contestación del desarrollador principal, se proponen algunas mejoras con el fin de que la implementación sea más correcta.

Por un lado, sobre la clase «ResourceDefaultColumn» pregunta si el comparador implementado se utiliza en algún otro lugar fuera de la implementación actual. Si no es así, propone mantener un comparador local en vez de almacenar la referencia en la clase actual.

Por otro lado, en la clase «ResourceTreeTable» propone subir el método de obtención de `UIFacade` a la clase «GPTreeTableBase» y convertirlo a `protected`.

Siguiendo con los cambios propuestos para esta segunda clase, propone también fusionar las dos clases anidadas de comparación implementadas «DescendingNameComparator» y «AscendingNameComparator» a una única clase, para ello, usando el método `Comparator` para conseguir el comparador inverso.

Otra de las mejoras que propone es la de gestionar los `Listener` de ratón y tabla de una forma global —ya que la gestión de las tablas puede ser común—. Para ello, una opción podría ser la de implementar un método común en la clase «GPTreeTableBase» de modo que podrá ser llamado desde las clases «GanttTreeTable» y la actual «ResourceTreeTable». Otra opción sería la de mover el código mencionado a la clase «GPTreeTableBase» y llamar a un método abstracto una vez la ejecución pase todas las comprobaciones.

En el apartado de ordenación de recursos (ver línea 33), propone usar el método `updateResources` para que la propia tabla la gestione.

El último comentario menciona mover la clase anidada «ModelListener» previamente implementada a la clase base para poder gestionarla de forma común para la clase actual y la de «GanttTreeTable».

6.2.4. Implementación de cambios propuestos

Se estudian las propuestas por parte del desarrollador principal y se procede a aplicar los cambios.

El primer cambio que se realiza trata de eliminar el comparador creado en «ResourceDefaultColumn» y crear uno local en «ResourceTreeTable» ya que, en el momento en el que se está escribiendo este documento, el compa-

rador no tiene ningún otro uso a parte de en la implementación de este *issue*.

En cuanto al método `getUIFacade()`, se procede a su eliminación en la clase «GanttTreeTable» y «ResourceTreeTable», y posterior creación como método protegido en la clase abstracta «GPTreeTableBase». El motivo principal de este cambio trata en que ambas clases —«GanttTreeTable» y «ResourceTreeTable» respectivamente— extienden a la clase abstracta previamente mencionada, por lo que si la implementación se encuentra en ella, ambas clases podrán usarla.

A continuación, se elimina una de las clases anidadas de comparación llamada «DescendingNameComparator». A partir de esta implementación se usará únicamente «AscendingNameComparator» para crear un comparador ascendente, y para conseguir el descendente, se generará el inverso.

```
41 Comparator<?> comparator = new AscendingNameComparator().
       reversed();
```

En relación con la gestión de los `Listener` de ratón y tabla, se opta por la primera opción: mantener una única implementación en la clase «GPTreeTableBase». Se crea el método `configureMouseListener` que recibirá el evento del ratón y filtrará las acciones no deseadas, devolviendo un 0 si la acción ha sido deseada, y un -1 si no lo ha sido. Se opta por esta opción en vez de un `Boolean` con el fin de poder filtrar diferentes acciones en un futuro.

Otro de los cambios se realiza en el método de ordenación de recursos. Para ello, se recarga —también conocido por *method overriding/overloading* en inglés— el método existente `updateResources` para que acepte la lista de recursos previamente ordenada y se gestione en «ResourceTreeTableModel». Para ello —y en el método recargado—, se limpia la lista con los datos anteriores, y se añaden los nuevos componentes en el orden previamente definido. Por último, se llama al método original para crear la estructura de la interfaz correctamente, ya que de no ser llamada, no aparecerán los nodos de cada recurso en pantalla.

Por último, la clase anidada «ModelListener» es movida a la clase abstracta «GPTreeTableBase» con el mismo fin aplicado a la obtención de `UIFacade`.

6.2.5. Últimos detalles

Para finalizar este *issue*, el desarrollador principal detalla pequeños cambios para que la contribución quede adecuada a los estándares.

Propone renombrar el método `configureMouseListener` a uno más especifico ya que se trata de un manipulador de eventos. También añade que por ahora será suficiente con devolver un `Boolean`. Se opta por asignarle el nombre `mouseEventHandling` y cambiar el tipo a `Boolean`.

El comparador implementado (ver línea 41) acepta todo tipo de estructura de datos, por lo que más adelante se tiene que detallar que se usará una del tipo `HumanResource`. Propone detallarla en la creación para evitar el *casting*.

Una vez realizados los cambios, se realiza el *pull request* y queda aceptado y el *merge* elaborado[8].

[8]Enlace al commit aprobado: https://github.com/bardsoftware/ganttproject/pull/1740

6.3. Desarrollo de la refactorización

En este desarrollo[9], y siguiendo con el enfoque especificado en el análisis (ver Sección 5.1.3), se procede a analizar los paquetes «ganttproject», «ganttproject-builder» y «ganttproject-tester».

Durante el análisis y desarrollo de los *issues* detallados en las secciones anteriores, se detectaron numerosas redundancias de código, *import*s que en su momento fueron usados pero por cambio en el código han quedado en desuso, etc.

Para generar un barrido general por paquetes, se utiliza la herramienta IntelliJ. Este entorno de desarrollo contiene herramientas de inspección de código que permiten analizar las posibles mejoras que pueden existir en el proyecto (ver Anexo A).

6.3.1. Paquete «ganttproject»

Se comienza con el paquete de «ganttproject», para el que se encuentran diferentes mejoras y correcciones.

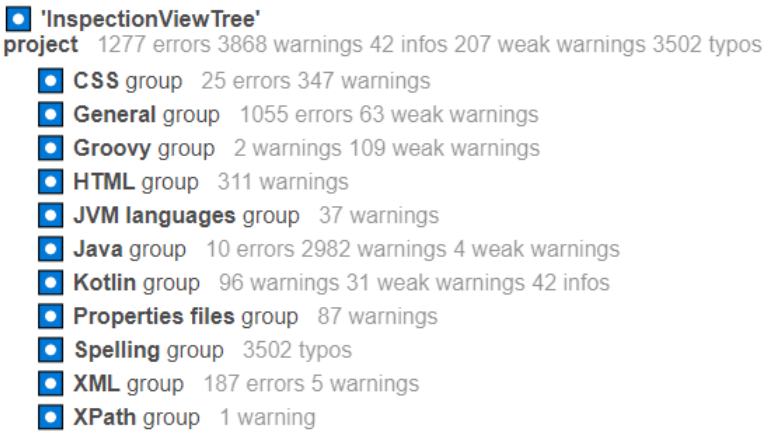


Figura 6.10: Inspección del paquete «ganttproject»

Se aprecia que en la clase «GanttTreeTableModel» se instancia un atributo privado de «GanttLanguage» al que no se le da uso. En vez de usar el atributo, se instancia en 212 ocasiones de forma aislada durante el proyecto.

[9]Enlace al pull request: https://github.com/bardsoftware/ganttproject/pull/1730

También se aprecian numerosos *import* en desuso en la clase «ChartSelection», por lo que se procede a eliminarlos.

Por último, existe una clase llamada «MathUtil» con un único método en ella. La clase no es usada para ningún fin, por lo que se procede a eliminarla.

6.3.2. Paquete «ganttproject-builder»

A continuación se analiza el paquete «ganttproject-builder» y se extrae la información del análisis.

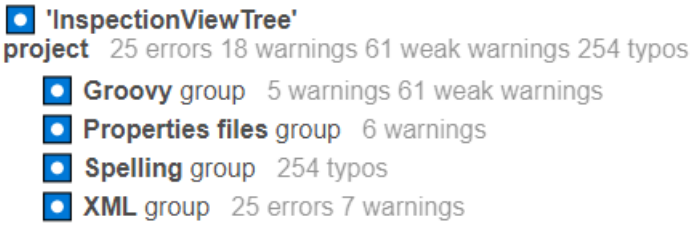


Figura 6.11: Inspección del paquete «ganttproject-builder»

Este paquete está centrado en la «construcción» del proyecto (ver Sección 5.1.3), por lo que no se procede a retocar ninguna de las divisiones de *Groovy* ni *XML* ya que exceden del conocimiento actual.

En cambio, si se aprecian ciertos puntos de mejora en el apartado de "Properties files" —como entradas de propiedades repetidas— pero siendo una clase editable por el usuario, se decide no modificarla.

6.3.3. Paquete «ganttproject-tester»

Por último, se analiza el paquete «ganttproject-tester».

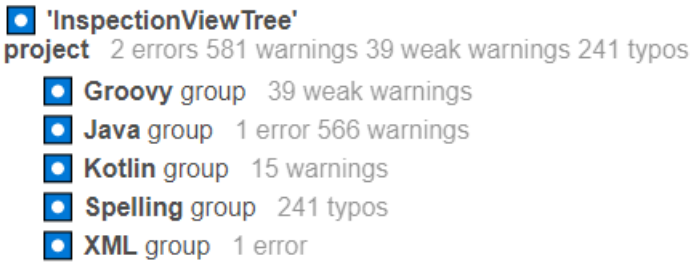


Figura 6.12: Inspección del paquete «ganttproject-tester»

De nuevo, existen *import* en desuso en las clases «TestTaskBounds», «WeekendCalendarImplTest», «ChangeTaskProgressRulerTest» y la de

«TestTaskDependencyCommon» que se proceden a eliminar.

A continuación, se filtran creaciones de *Array* redundantes en la clase «TestResourceAssignments».

```
42 Set<HumanResource> expectedResources =
43  new HashSet<HumanResource>(
44   // previo
45   Arrays.asList(new HumanResource[] { res1, res2 }));
46   // posterior
47   Arrays.asList(res1, res2));
```

Y en la clase «TestTaskHierarchy»:

```
48 assertEquals("Unexpected nested tasks of task=" + task1,
      Arrays
49   // previo
50   .asList(new Task[] { task2 })
51   // posterior
52   .asList(task2)
```

Por último se modifica la clase «ImportTasksTestCase». En ella, se eliminan "boxing" innecesarios. La técnica de "boxing" —también conocida como *wrapping* o envolver— es el proceso de colocar un tipo primitivo (valor) dentro de un objeto con el fin de ser usado como referencia.

En versiones de *Java* anteriores a J2SE 5.0 era necesario realizar este proceso para evitar errores de compilación.

```
53 // versiones previas a 5.0
54 Integer i = new Integer(9);
55 // posteriores
56 Integer i = 9;
```

Sin más cambios, se realiza el *pull request* dividido en *commit* diferentes para facilitar su corrección.

6.3.4. Feedback del desarrollador

Tras revisar los cambios realizados, el desarrollador principal le da el visto bueno y procede a realizar el *merge* hacia el código original de GanttProject[10].

[10] Enlace al commit aprobado: https://github.com/bardsoftware/ganttproject/pull/1730

7. Pruebas

En este capítulo se estudia si el funcionamiento de los *issues* implementados es el correcto. Para ello, se analizará cada uno de forma independiente y se corroborará que funcionan sin arrojar errores.

GanttProject contiene una gran cantidad de test unitarios para cada clase implementada, pero después de analizarlos y ver que la gran mayoría de ellos falla, se procede a probar su funcionalidad de forma manual.

7.1. Pruebas de funcionalidad de Issue: tkt_1659_decimales

Tras implementar la nueva funcionalidad que permite ver 2 decimales en la columna de coste, se comprueban varias funcionalidades que ha de cumplir:

- Se crea correctamente la columna de coste
- Si se importa un proyecto que la contiene, la columna se carga correctamente
- Se detecta que los valores son del tipo «Double»
- Utilizando números enteros, se muestran 2 decimales
- Utilizando un solo decimal, se muestran 2 decimales
- Utilizando 2 decimales, no cambia el formato
- Utilizando más de 2 decimales, se muestran únicamente 2

Todos los test se cumplen correctamente, por lo que se puede concluir que la implementación cumple con su cometido.

7.2. Pruebas de funcionalidad de Issue: tkt_1596_recursos

El objetivo de este *issue* trata de añadir la funcionalidad de ordenar la tabla de recursos alfabéticamente de forma ascendente o descendente. También debe de ordenarse el gráfico correspondiente a cada recurso.

Para comprobar el correcto funcionamiento de la implementación, se procede a confirmar si los objetivos preestablecidos se cumplen:

- Sin pulsar la cabecera, los recursos se encuentran en su orden original
- Al pulsar la cabecera, si los recursos están ordenados de forma ascendente, se ordenan de forma descendente
- Al pulsar la cabecera, si los recursos están ordenados de forma descendente, se ordenan de forma ascendente
- Cuando los recursos son ordenados, cada gráfico correspondiente lo hace también
- No se mezclan las tareas de cada recurso cuando son ordenados

En esta ocasión, se cumplen de nuevo todos los test predefinidos, por lo que se concluye que esta implementación cumple con su cometido.

7.3. Pruebas de funcionalidad de *refactor*

Por último, se realizan las pruebas de la refactorización realizada en los paquetes «ganttproject», «ganttproject-builder» y «ganttproject-tester».

Tras la modificación de cada uno de los paquetes, se han colocado *breakpoints* en las líneas afectadas, y se ha procedido a utilizar un *debugger* para comprobar que no surgen problemas.

Una vez comprobado que no surgen problemas, se concluye que este último *issue* también cumple con su cometido.

8. Conclusiones

A lo largo de la duración del proyecto, se ha establecido un ritmo de trabajo continuo que ha permitido realizar las tareas predefinidas antes de la fecha límite de la entrega del mismo. Se han estudiado las diferentes aproximaciones hacia los proyectos *open source*, como analizar y colaborar en ellos, así como a desarrollar un proyecto «real» y utilizable fuera del paraguas de la universidad.

Pese a realizar una planificación previa calculando y analizando las tareas del proyecto y el tiempo dedicado a cada una de ellas, se han sufrido retrasos por las complicaciones con la configuración y actualizaciones del proyecto, y en los tiempos de respuesta por parte del desarrollador principal. No se debe olvidar que se trata de un proyecto en constante desarrollo, por lo que los contratiempos pueden estar presentes más frecuentemente.

Este último capítulo de la memoria se centrará en la comparativa de la planificación estimada y la real, en las lineas futuras que puede presentar este proyecto y en una reflexión personal de la alumna Oihane Albizuri Silguero, que dará finalización a sus estudios del «Grado de Ingeniería Informática de Gestión y Sistemas de Información» con la finalización de esta memoria.

8.1. Análisis entre planificación estimada y real

Realizar una planificación exacta es muy difícil y casi imposible, más aún cuanto existen potenciales riesgos que pueden afectar en el desarrollo de un proyecto. En esta ocasión, se han sufrido contratiempos que se han podido solventar antes de la fecha de finalización del mismo.

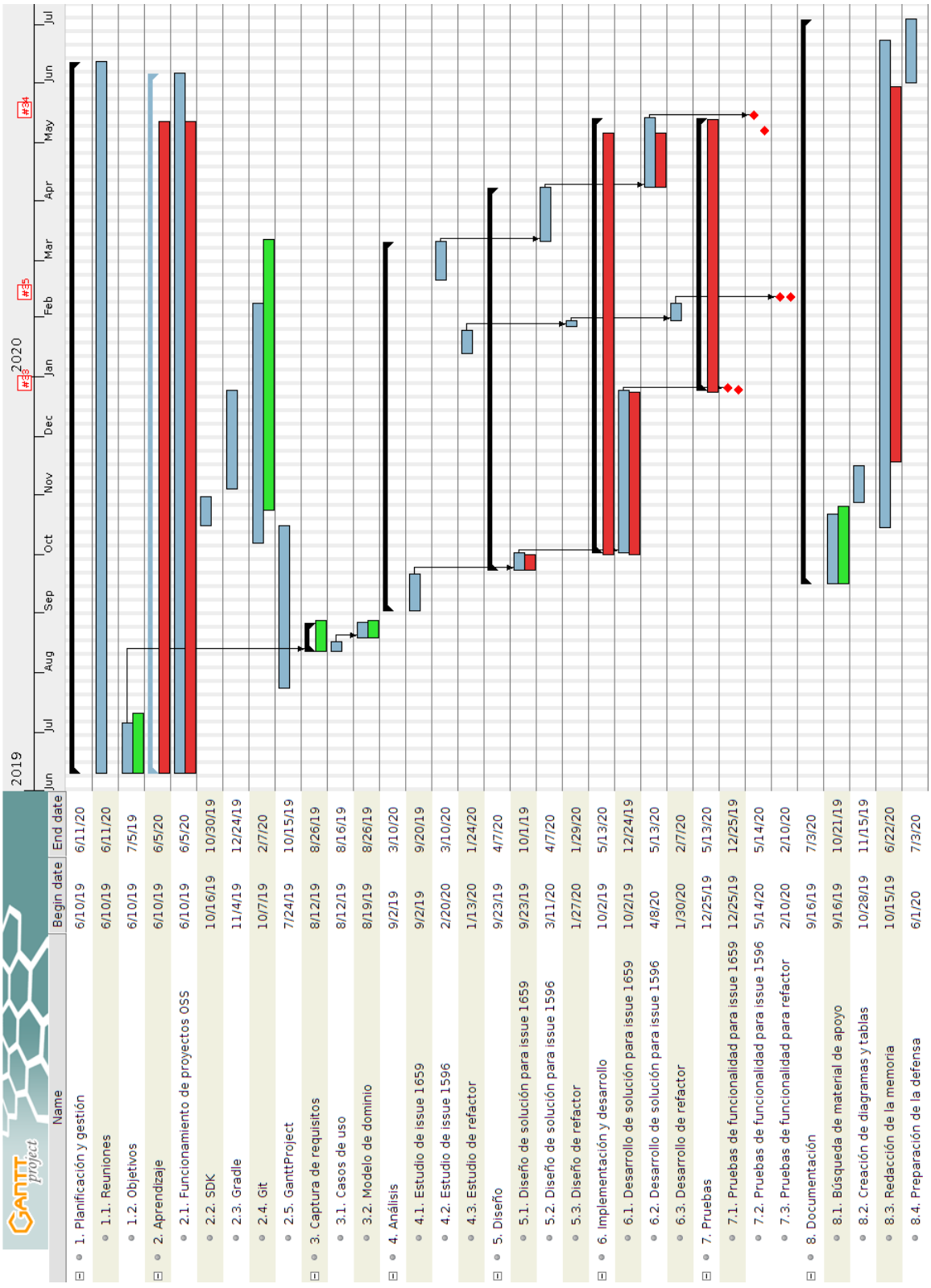


Figura 8.1: Comparativa de planificaciones

La Figura 8.1 muestra la evolución de la planificación estimada que se realizo en el inicio del proyecto y la real.

Por cada tarea se muestran dos líneas de tiempo: la superior indica la real, mientras que la inferior indica la preestablecida. En caso de mostrarse una única línea en la tarea, indica que no ha sufrido ningún cambio.

En cambio, algunas de las tareas se muestran en verde y rojo, e indican que han sido afectadas. Los dos colores, verde y rojo, indican que se ha sufrido un adelanto o retraso respectivamente, y mayormente se encuentran en la tarea de desarrollo, que afectan directamente a la de aprendizaje de funcionamiento de proyectos OSS y de redacción de memoria.

Gracias al control de horas realizado por la herramienta Toggl, se ha calculado que la suma de horas reales asciende a 553 repartidas en las tareas previamente definidas, lo que supone un aumento del 11,7 %.

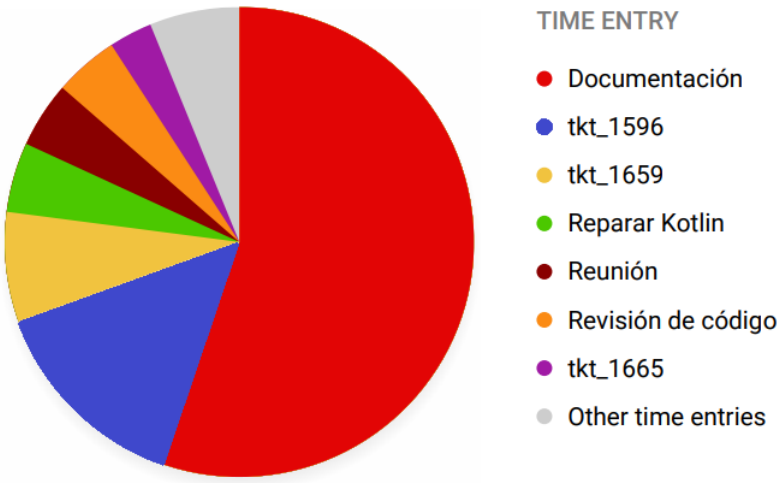


Figura 8.2: Gráfico circular de tareas

Más de la mitad del tiempo invertido en el proyecto se adjudica a la etiqueta de documentación. Esto se debe a que se ha invertido una cantidad considerable de tiempo en la búsqueda de recursos y artículos con el fin de lograr la formación necesaria para desarrollar el proyecto.

8.2. Líneas futuras

Este año se han implementado pequeñas funcionalidades y arreglo de errores. Con la experiencia adquirida y la documentación generada, se estima que se podría tomar como base o apoyo práctico para la docencia de otras asignaturas: «Análisis y Diseño de Sistemas de Información», «Ingeniería del Software», Programación...

Por otro lado, de cara a futuros TFG, podrían plantearse tareas más complejas, como el soporte de planificación hasta nivel de horas —un *issue*[1] que lleva abierto varios años sin resolverse, debido a la complejidad del mismo, pues tendría impacto en muchas de las clases que forman GanttProject—.

8.3. Licencias

Siguiendo la idea principal de este TFG de aprender y contribuir en *Open Source Software*, toda la información recopilada en el documento seguirá la misma filosofía; cualquier desarrollador tiene permiso para leerlo, modificarlo y redistribuirlo como considere, siempre que mantenga la misma licencia CC-by-sa. Con respecto a las contribuciones al proyecto GanttProject, se aplica la licencia *GNU General Public License v3.0* previamente mencionada en la Sección 1, que permite usar, estudiar, compartir y modificar el código fuente siempre que se mantenga la licencia.

8.4. Reflexión personal

En lo referente a la valoración personal, se ha de mencionar que los inicios del desarrollo fueron cautelosos y desconfiados. La idea de embarcarse en un proyecto tan inmenso y desarrollar junto a programadores experimentados resultaba del todo foráneo. El concepto de ser valorado por ellos y no por docentes de la universidad era algo aterrador.

A medida que se adquirían conocimientos y ser realizaban contribuciones previas a las mencionadas en esta memoria —con el fin de obtener la práctica necesaria—, se comprendió que no había tanto que temer. En caso de no ser aceptado el *pull request*, siempre había opción de corregir los errores y volver a presentar una nueva implementación.

La experiencia y conocimientos adquiridos durante el transcurso de este proyecto se valora de manera muy positiva, y se puede asegurar que se continuará con las contribuciones en un futuro.

[1] Enlace al issue: https://github.com/bardsoftware/ganttproject/issues/225

Anexos

Anexo I: Inspección de paquetes usando IntelliJ

Para realizar el desarrollo de la aplicación se ha utilizado el entorno de desarrollo IntelliJ. Esta aplicación permite inspeccionar el código de forma general o por paquetes —por ejemplo, sólo código en producción, o archivos modificados—.

En el desarrollo de refactorización se ha decidido hacerlo por paquetes (ver Sección 5.1.3) por el volumen de clases que GanttProject maneja y para que los PR pudieran ser revisados —y por tanto aceptados— de forma más fácil. Para realizar la inspección hay dos opciones disponibles:

- **Desde el panel principal**: En el menú, se selecciona "Analizar" —o *Analyze* en inglés— y a continuación "Inspeccionar Código..." —*Inspect Code...*—.

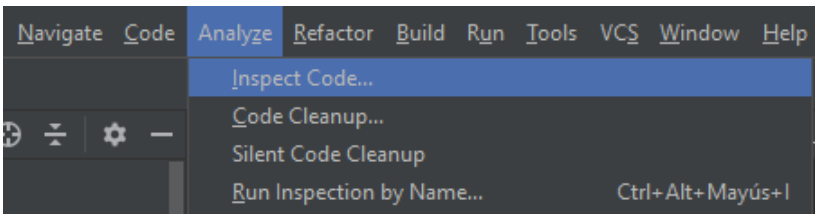


Figura A.1: Opciones a seleccionar en el menú

- **Seleccionando el paquete**: El la zona izquierda del editor se muestran todos los archivos que forman el proyecto.

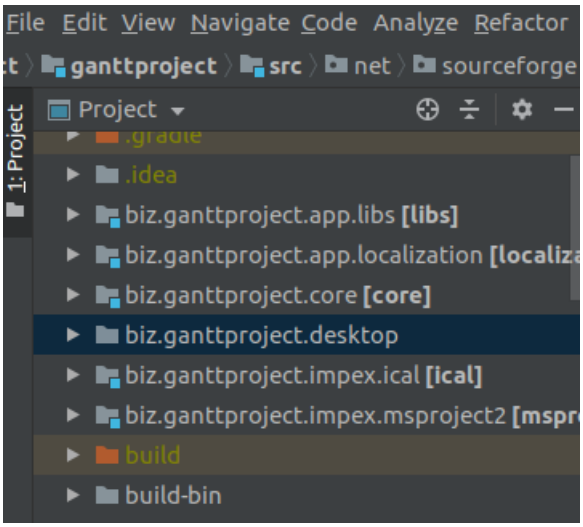


Figura A.2: Archivos en IntelliJ

Pulsando sobre el paquete o archivo con el botón derecho del ratón se despliega el mismo menú que en el paso anterior (ver Figura A.1).

Tras escoger cualquiera de las dos opciones para inspeccionar archivos, aparece en pantalla una nueva ventana para la configuración del análisis.

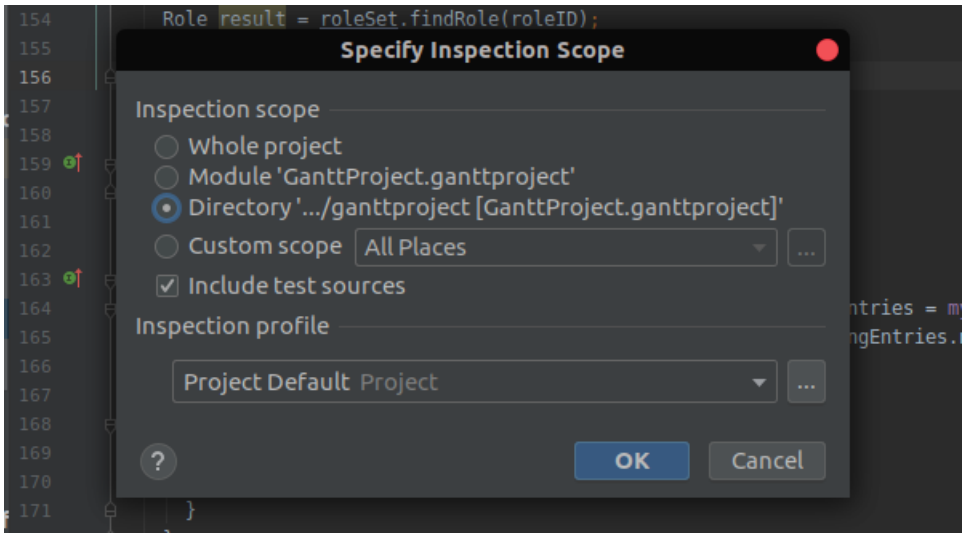


Figura A.3: Configuración del análisis a realizar

En la misma, se detallan el alcance de la inspección —proyecto completo, módulos, directorio o personalizada— junto al perfil de inspección.

Cada inspección realizada tiene un nivel de severidad. Cada una de ellas se muestra de diferente manera en el editor, lo que facilita distinguir rápidamente entre errores críticos y no tan importantes como pueden ser errores tipográficos.

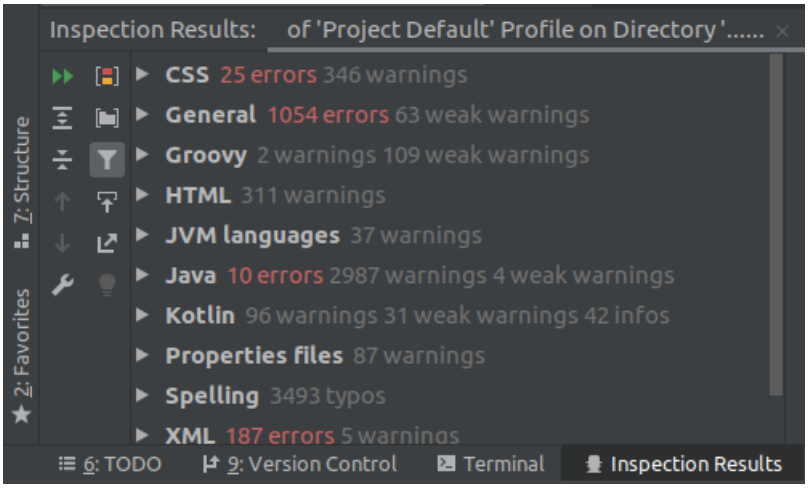


Figura A.4: Resultado de inspección

En la Figura A.4 se pueden ver numerosos errores del paquete «ganttproject». Están divididos por categorías junto con la cantidad de errores y advertencias que tiene cada una de ellas.

Como ejemplo, se ha seleccionado la categoría *Java*, y se ha recorrido en busca de errores que se consideran de más importancia.

En el desarrollo del *issue* relacionado con los recursos (ver Sección 6.3) se elimina la clase «MathUtil» ya que solo consta de un método llamado `signum` no usado en ninguna ocasión. La herramienta IntelliJ facilita la eliminación del método de forma segura (ver Figura A.5).

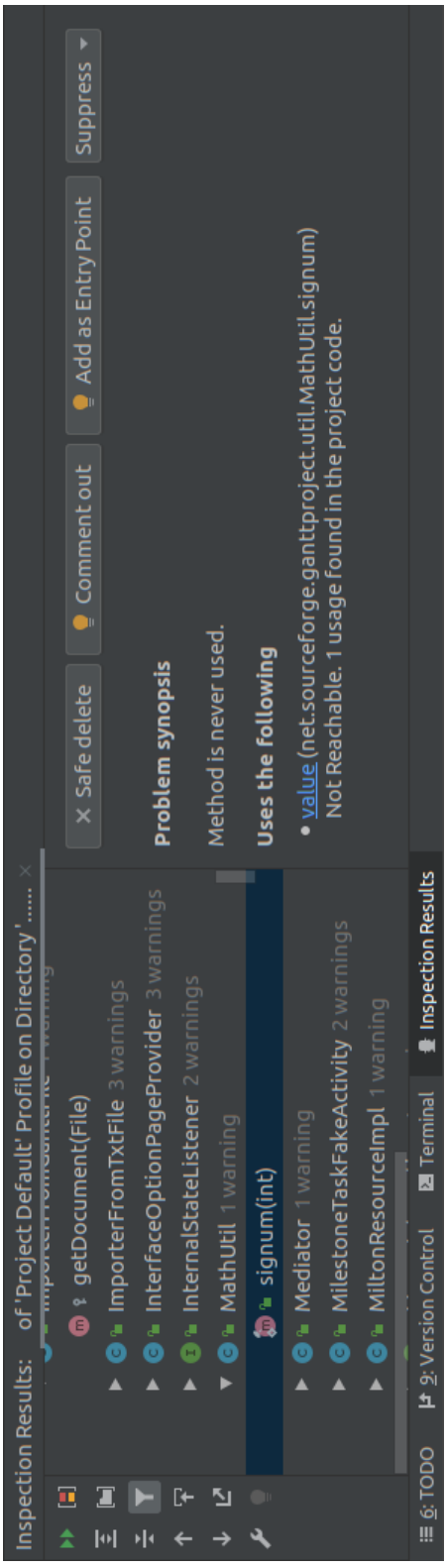


Figura A.5: Inspección de la clase «MathUtil»

Acrónimos

BOE

Boletín Oficial del Estado. 32

OS

Operating System. 10

OSS

Open Source Software. 1, 4, 6, 7, 16, 33

RAE

Real Academia Española. 99

Glosario

API

Conocido también por ser las siglas de "Application Programming Interface" —o Interfaz de Programación de Aplicaciones—, se trata de procesos, funciones y métodos proporcionados por una librería con el fin de ser utilizado por otro software. 10

artefacto

Se trata de un objeto tangible resultante de un proceso de desarrollo de software. 54

branch

Una rama —o branch en inglés— en Git es un puntero para el rastreo de commits. 47, 64

breakpoint

Punto de ruptura en castellano, se trata de un punto crítico en el que se pausa el programa con el fin de ser depurado. 81

bug

En informática, agujero o brecha con falta de seguridad de un programa de computación. 3, 14, 19

casting

Trata de alterar una variable primitiva de un tipo a otro. 75

code review

Code review —o revisión de código en castellano— es un examen del código fuente de un programa informático. 61

Command Line Interface (CLI)

Interfaz de linea de comandos en castellano, se trata de una interfaz basada en texto usada para ver y gestionar ficheros. 10

commit

En Git, acción de grabar los cambios realizados en el proyecto en el repositorio. 78

debugger

También conocido como depurador, es un programa usado para probar y depurar los errores de software. 81

diagrama de Gantt

Se trata de una herramienta gráfica cuyo objetivo es exponer el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado. 9

Diagrama EDT o Estructura de Descomposición de trabajo

Es un diagrama que consiste en la descomposición, de manera jerárquica, de las tareas de un trabajo o proyecto. 9

feedback

La retroalimentación es un mecanismo por el cual cierta porción de la salida de un sistema es redirigida a la entrada del mismo. 61, 67

Git

Es un software de control de versiones diseñado por Linus Torvalds y utilizado con frecuencia para el desarrollo de proyectos software. 7, 19

GitHub

Servicio de hosting open source para Git que permite alojar proyectos en remoto. 4, 7, 19, 45, 47, 64

Gradle

Es un sistema de automatización de construcción de código abierto construido sobre Apache Ant y Apache Maven. 7, 10, 72

hacker

Persona con grandes habilidades en el manejo de computadoras que investiga un sistema informático para avisar de los fallos y desarrollar técnicas de mejora. 1

hardware

Admitida por la RAE por no poseer una traducción adecuada al contexto, se trata del conjunto de aparatos de una computadora. 10, 33

hosting

Alojamiento web en castellano, es un servicio en línea que permite publicar un sitio o aplicación web en Internet. 7, 9

issue

A diferencia del bug —o error en castellano—, un issue esta directamente relacionado con un proceso. Cuando se espera que la aplicación o programa funcione de cierta manera, pero no lo hace, se trata de un issue. 3, 7, 9, 11, 14, 18, 19, 22, 24, 47–49, 59, 61, 64, 66, 67, 74, 76, 79–81

Java

Es un lenguaje de programación orientado a objetos incorporado en los años noventa a la informática. 7, 10

kernel

El núcleo o kernel es un software que constituye la base para todo sistema operativo. 45

Kotlin

Lenguaje de programación de tipado estático que corre sobre la máquina virtual de Java y que también puede ser compilado a código fuente de JavaScript. 69

lenguaje de programación interpretado

Es un tipo de lenguaje de programación que necesita un programa informático (interprete) para ser analizado y ejecutado. Entre los más conocidos están Java, Python o Ruby. 10

Linux

Se trata de la distribución basada en sistemas Unix y en el kernel Linux. 45

macros

Secuencia de acciones dentro de la aplicación que el usuario activa pulsando una combinación específica de teclas. 10

merge

En Git, combinar dos ramas. 7, 63, 64, 66, 75, 78

Open Innovation (OI)

Se trata de nueva estrategia de innovación mediante la cual las empresas combinan su conocimiento interno con el externo para sacar adelante los proyectos de estrategia y de I+D junto a otras empresas. 43

open source

Software en el que el código fuente esta completamente disponible y puede ser redistribuido y modificado. 2, 4, 8, 44, 45, 64, 83

Open Source Software

Software en el que el propietario de los derechos de autor permite a los usuarios utilizar, cambiar y redistribuir el software, a cualquiera, para cualquier propósito, ya sea en su forma modificada o en su forma original. También es conocido como Open Source Software (OSS). 1, 3–5, 43–45, 86

origin

En Git, término usado para referenciar el repositorio remoto del que originalmente se clona. 64, 65

pull

En Git, extracción del código de otro repositorio para posteriormente ser integrado en local u otro repositorio. 65

pull request

Cuando se colabora en proyectos open source, lo más común es realizar cambios en local, y a posterior presentar dichos cambios a un proyecto remoto donde serán revisados antes de ser implementados o unidos al original. A esta acción se le llama realizar un pull request. 24, 61, 63, 72, 75, 78, 86

refactor

Se trata de un anglicismo proveniente de la palabra refactor. Es una técnica de la ingeniería del software que consiste en reestructurar el código fuente sin alterar el comportamiento externo. 21, 22, 24

renderizador

De trata de un anglicismo para la representación gráfica usado en áreas de la informática. 57, 62–64, 66

repositorio

En Git, localización de los archivos relacionados con un projecto. 45, 61, 64

sistema operativo

Software primario que controla todo el hardware y el resto de software de una computadora. 1

software

Admitida por la RAE por no poseer una traducción adecuada al contexto, se trata de un conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora. 1, 8, 33, 43, 44, 49, 54

standalone

Del anglicismo "stand alone", es la habilidad de operar independientemente de otro hardware o software. 54

tethering

Proceso por el cual un dispositivo móvil con conexión a Internet actúa como pasarela para ofrecer acceso a la red a otros dispositivos, asumiendo dicho dispositivo móvil un papel similar al de un módem o enrutador inalámbrico. 40

tkt

Formato en el que se realizan los pull request. 47

Unix

Sistema operativo análogo a DOS y Windows. 10

upstream

En Git, término usado para referenciar el repositorio remoto original del que se realiza el fork —o bifurcación—. 64, 65

Bibliografía

Sogol Balali, Igor Steinmacher, Umayal Annamalai, Anita Sarma, and Marco Aurelio Gerosa. Newcomers' barriers... is that all? an analysis of mentors' and newcomers' barriers in oss projects. *Computer Supported Cooperative Work (CSCW)*, 27(3-6):679–714, 2018.

Julia Beckhusen. Occupations in information technology. https://www.census.gov/content/dam/Census/library/publications/2016/acs/acs-35.pdf, 2016.

William H Brown, Raphael C Malveau, Hays W McCormick, and Thomas J Mowbray. *AntiPatterns: refactoring software, architectures, and projects in crisis*. John Wiley & Sons, Inc., 1998.

Peter Burns. How to resolve merge conflicts in git. https://stackoverflow.com/a/163659, 2012.

Andrea Capiluppi, Maurizio Morisio, and Patricia Lago. Evolution of understandability in oss projects. In *Eighth European Conference on Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings.*, pages 58–66. IEEE, 2004.

Brian W Carver. Share and share alike: Understanding and enforcing open source and free software licenses. *Berkeley Technology Law Journal*, 20 (1):443–481, 2005.

Director de Comunicaciones Red Comisión Europea. Women in digital. https://ec.europa.eu/digital-single-market/en/women-ict, 2019.

Vincent Driessen. A successful git branching model. https://nvie.com/posts/a-successful-git-branching-model/, 2010.

Pankaj Kamthan. On the prospects and concerns of integrating open source software environment in software engineering education. *Journal of Information Technology Education: Research*, 6(1):45–64, 2007.

Andrew M St Laurent. *Understanding open source and free software licensing: guide to navigating licensing issues in existing & new software*. .ºReilly Media, Inc.", 2004.

J. Linåker, H. Munir, K. Wnuk, and C.E. Mols. Motivating the contributions: An open innovation perspective on what to share as open source software. *Journal of Systems and Software*, 135:17 – 36, 2018. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2017.09.032. URL http://www.sciencedirect.com/science/article/pii/S0164121217302169.

Michael Pergamenshik. Project comprehension: Understanding java projects efficiently. https://dzone.com/articles/project-comprehension-understanding-java-projects, 2017.

Stormy Peters and Nithya Ruff. Participating in open source communities. https://www.linuxfoundation.org/resources/open-source-guides/participating-open-source-communities/, 2020.

Sulayman K Sowe and Ioannis G Stamelos. Involving software engineering students in open source software projects: Experiences from a pilot study. *Journal of Information Systems Education*, 18(4):425, 2007.