



ugr

Universidad
de Granada

TRABAJO FIN DE GRADO
INGENIERÍA INFORMÁTICA

Detección de errores en bases de datos químicas

Autor

Jesús Navarro Merino

Directora

Rocío Celeste Romero Zaliz



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

Granada, julio de 2023

Detección de errores en bases de datos químicas

Jesús Navarro Merino

Palabras clave: palabra_clave1, palabra_clave2, palabra_clave3,

Resumen

Poner aquí el resumen.

Project Title: Project Subtitle

First name, Family name (student)

Keywords: Keyword1, Keyword2, Keyword3,

Abstract

Write here the abstract in English.

Yo, **Jesús Navarro Merino**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 15429457E, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Jesús Navarro Merino

Granada a X de MES de 201 .

Dña. **Rocío Celeste Romero Zaliz**, Profesora del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado *Detección de errores en bases de datos químicas*, ha sido realizado bajo su supervisión por **Jesús Navarro Merino**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a X de mes de 201 .

La directora:

Rocío Celeste Romero Zaliz

Agradecimientos

Poner aquí agradecimientos...

Índice general

1. Introducción	1
1.1. Motivación	2
1.2. Objetivos	7
1.3. Estructura de la memoria	7
2. Estado del arte y fundamentos teóricos	9
3. Gestión y Planificación del proyecto	11
3.1. Metodología	11
3.1.1. Scrum	13
3.1.2. XP	15
3.1.3. Aplicación de Scrum/XP al proyecto	16
3.2. Planificación	17
3.3. Gestión de la configuración	20
3.3.1. Gestión del código	20
3.3.2. Gestión de la documentación	20
3.4. Gestión de recursos	21
3.4.1. Recursos humanos	21
3.4.2. Recursos materiales	21
3.4.3. Recursos software	21
3.5. Gestión de costes	23
3.5.1. Coste de recursos humanos	24
3.5.2. Costes de recursos materiales	25
3.5.3. Costes software	26
3.5.4. Otros costes	26
3.5.5. Presupuesto final	26
3.6. Análisis de riesgos	27
3.6.1. Riesgos materializados	29
4. Diseño	31
4.1. Diagrama de Clases	31
4.2. Clases modificadas	33
4.3. Clases nuevas	37

5. Implementación y resultados	43
---------------------------------------	-----------

Índice de figuras

1.1. Distintas cadenas SMILES válidas para el 1-methyl-3-bromo-ciclohexeno. (a) Considera el ciclo como la rama principal y el bromo como ramificación. (b) Hace el recorrido que marca la flecha, dejando parte del ciclo como una ramificación. Imagen extraída de [3]	3
3.1. Ciclo de vida iterativo de Scrum.	13
3.2. Diagrama de Gantt sobre la planificación inicial estimada . .	18
3.3. Diagrama de Gantt sobre la planificación temporal real . . .	19
3.4. Registro de horas dedicadas al proyecto a través de Clockify .	25
3.5. Riesgos del proyecto, causas, y planes de actuación	28
3.6. Matriz de probabilidad-impacto de riesgos	29
4.1. Diagrama de clases	32

Índice de tablas

1.1. Códigos SMILES y sus representaciones visuales según Sigma-Aldrich	5
1.2. Códigos SMILES y sus representaciones visuales según Sci-Finder	6
3.1. Tabla comparativa de las principales características entre las metodologías tradicionales y ágiles	12
3.2. Tabla de los costes materiales	26
3.3. Tabla de costes adicionales	26
3.4. Presupuesto total del proyecto	27

Capítulo 1

Introducción

La Química estudia la composición y estructura de la materia, sus propiedades y transformaciones. Estudia las sustancias, la energía y sus cambios durante las reacciones. Desde que se tienen registros, la química ha sido fundamental para el desarrollo de la humanidad, ya que ha permitido la producción de materiales, alimentos, medicamentos y energía, entre otros. Esto ha sido un proceso lento y exhaustivo a través de la experimentación. Por ejemplo, en 1881, Beilstein publica su Manual de Química Orgánica, que recogía 15000 compuestos orgánicos con sus propiedades [2]. Conforme la química se iba expandiendo, también lo hacía el volumen de datos que se generaban, siendo cada vez mas frecuentes preguntas como "¿alguien habrá sintetizado ya este compuesto?" [11]

Eventualmente, hace unas cuantas décadas, se pensó que la cantidad de información que cada químico por su cuenta había acumulado, se podía compartir y hacer accesible a la comunidad científica a través de su almacenamiento en bases de datos [4]. Con el desarrollo de técnicas de manipulación y tratamiento de esos datos surgió el término *chemoinformatics* (en español, quimioinformática).

Las *chemoinformatics* han cobrado gran importancia en los últimos años debido al aumento exponencial de datos experimentales generados en la investigación biomédica y química, y a la necesidad de manejar y analizar esta información de manera eficiente. Esta disciplina ha sido influenciada por diversas áreas, como la química, matemáticas, estadística, biología y ciencias de la computación entre otras. Al parecer, su origen se remonta a la década de 1940, habiendo ya algunas investigaciones en el área, pero el término 'chemoinformatics' se lleva utilizado desde 1998 [6]. Como tal, aun no hay un acuerdo en cuanto a su definición, seguramente por su carácter interdisciplinar, ni si quiera en cómo deletrearlo, pudiendo aparecer también como *cheminformatics*, *chemical informatics*, *chemi-informatics*, y *molecular informatics* entre otras [6, 8]. En la literatura se discuten varias inter-

pretaciones sobre su definición, unas más precisas y otras más generales: [6, 7, 8, 4]

La mezcla de recursos de información para transformar datos en información, y la información en conocimiento, con el fin de tomar decisiones más rápidas y efectivas en la identificación y optimización de fármacos [Brown 1998]

Chem(o)informatics es un término genérico que abarca el diseño, la creación, la organización, la gestión, la recuperación, el análisis, la difusión, la visualización y el uso de la información química. [G. Paris 1999]

La aplicación de métodos informáticos para resolver problemas de química [J. Gasteiger and T. Engel 2006]

A pesar de ello, son a día de hoy un componente esencial en el descubrimiento de sustancias químicas; sin duda es un área en constante evolución y su importancia solo aumentará en los próximos años, tanto en el descubrimiento de fármacos —que es como originariamente surgió y donde más impacto tiene en la sociedad— como en otros campos de la química.

Una herramienta de vital importancia en este ámbito son los sistemas de representación lineal. Algunos formatos que usan este tipo de representación son Wiswesser Line Notation (WLN), Sybyl Line Notation (SLN), Representation of structure diagram arranged linearly (ROSDAL), Simplified Molecular-Input Line-Entry System (SMILES) o IUPAC Chemical Identifier (InChI). Surgieron a medida que la química y la tecnología computacional avanzaban, y nos permiten codificar moléculas para su análisis y almacenamiento en bases de datos. En el siglo XIX, se desarrollaron varias formas de representación visual de moléculas, como las fórmulas estructurales que permitieron a los químicos dibujar y visualizar moléculas de manera más efectiva. Sin embargo, estas formas de representación no son adecuadas para su uso en la computación, ya que no son fácilmente legibles para los programas informáticos. Nosotros, los humanos, cuando vemos una estructura molecular dibujada la entendemos directamente, obtenemos una visión global de los símbolos que representan los enlaces y la distribución espacial de los átomos que la componen, pero los computadores no tienen esa facilidad. Por ello, se desarrollaron sistemas de notación lineal que permitían describir de manera más precisa y eficiente la estructura molecular, trabajando con tipos de datos sencillos, usando por ejemplo cadenas de caracteres.

1.1. Motivación

Los formatos de representación lineal llevan siendo un tema de interés e investigación para los científicos desde mediados del siglo XIX, evolucionando poco a poco y desarrollándose nuevas representaciones en función de

las necesidades —principalmente computacionales— del momento y las limitaciones que se iban descubriendo [1]. En la actualidad las más usadas SMILES, InChI, y SELFIES [18]. Como comenté antes, una forma muy potente de representar moléculas y compuestos químicos es mediante cadenas de caracteres, y de esto justamente se encargan las representaciones lineales: traducir una molécula, con sus átomos, enlaces entre ellos, ciclos y otras propiedades, en una cadena de caracteres que la represente, y que la máquina y los propios químicos puedan entender. Sin embargo, hay diferencias notables entre las representaciones, tanto en la sintaxis de las cadenas que se generan como en las aplicaciones que se le puede dar a cada una de ellas.

SMILES, ideada por David Weininger, sale a la luz en 1988 satisfaciendo con creces las necesidades de procesamiento de información química que había, desbancando a la representación estandarizada del momento, Wiswesser Line Notation (WLN). Desde ese entonces SMILES se convirtió —y sigue siendo a día de hoy— en el estándar de representación lineal, ya que permite describir estructuras moleculares de una forma sencilla en un formato fácil de leer, lo que ha hecho que sea una herramienta popular en la química computacional, siendo la más usada entre investigadores y químicos. Pese a esto, SMILES tiene dos grandes inconvenientes: una misma molécula puede escribirse con varias cadenas SMILES distintas válidas, es decir, tiene sinónimos (Figura 1.1); y no es robusto ni sintáctica ni semánticamente. En este sentido se podría generar un string que no represente una molécula válida, como lo es CC(CCCC, el cual tiene un paréntesis sin cerrar (lo que implica que no se delimita cuándo acaba la rama). O generar una molécula que no sea químicamente viable como CO=CC, que muestra un átomo de oxígeno neutro formando tres enlaces (superando el límite de enlaces covalentes que un oxígeno neutro puede tener) [18].

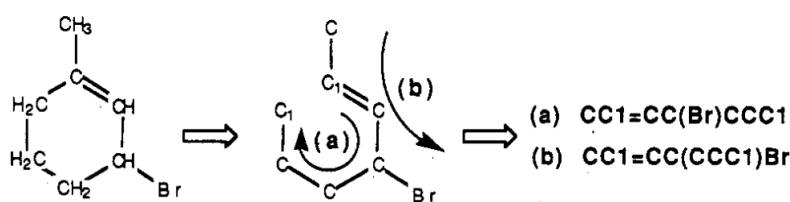


Figura 1.1: Distintas cadenas SMILES válidas para el 1-methyl-3-bromociclohexeno. (a) Considera el ciclo como la rama principal y el bromo como ramificación. (b) Hace el recorrido que marca la flecha, dejando parte del ciclo como una ramificación. Imagen extraída de [3]

Esto tiene especial relevancia en el ámbito del Machine Learning (ML) o Aprendizaje Automático. Aunque se sale del alcance de este trabajo, uno de los grandes objetivos de la química computacional es la creación o diseño de nuevas moléculas. Se podrían crear modelos de ML, en particu-

lar de redes neuronales, capaces de generar moléculas ficticias válidas, para posteriormente ver sus propiedades, valorarlas energéticamente para ver cuán estables son, y estudiar su viabilidad en distintas aplicaciones, entre otras cosas. SMILES dificulta esta tarea, y por ello, aparece en 2020, SELFIES (SELF-referencIng Embedded Strings), una nueva representación lineal 100 % robusta, muy usada actualmente para modelos generativos (ver [18, 14] para más detalles de cómo soluciona los problemas de robustez y otras características de la representación). SELFIES es relativamente reciente y continuamente está ampliando sus funcionalidades, mejorando en simplicidad y facilidad de uso [19]. Aun así, no se termina de instaurar entre la comunidad investigadora. Por último, InChI es creado en 2013 por la IUPAC (International Union of Pure and Applied Chemistry) como un proyecto para estandarizar el proceso de búsqueda de estructuras moleculares entre distintas bases de datos. Esto es porque InChI (International Chemical Identifier) genera una cadena canónica única para cada molécula, de manera que cada molécula tiene una sola representación, y dicha representación solamente hace referencia a esa molécula. La principal desventaja radica en su sintaxis y su estructura jerárquica, haciéndola complicada de leer y utilizar por los humanos. Por esto mismo, no es la mejor opción para usar en modelos generativos, pues tiene una serie de reglas y normas gramaticales y aritméticas que son complejas de aplicar al generar moléculas a través de modelos de ML[10].

Por todo lo anterior, me centraré en la notación SMILES durante el desarrollo de este trabajo. Dicho esto, existen diversas bases de datos en química donde se recoge gran cantidad de información acerca de los compuestos. Entiéndase esto como una colección estructurada y organizada que contiene datos sobre compuestos químicos, sus propiedades y relaciones con otros compuestos. Se utilizan para almacenar y recuperar información sobre moléculas, sustancias, reacciones, propiedades fisicoquímicas, e incluso literatura científica relacionada. Mencionaré ahora las más importantes y las que serán objeto de interés. *PubChem*, una base de datos abierta que sirve información a millones de usuarios en todo el mundo, desde investigadores y estudiantes hasta el público general. Recogen para cada compuesto, información sobre su estructura, representaciones 2D y 3D, identificadores, propiedades químicas y físicas, patentes, avisos de toxicidad, etc. [22]

SciFinder, una herramienta de investigación muy potente que permite explorar las bases de datos de CAS (American Chemical Society) las cuales contienen literatura sobre Química y otras disciplinas afines como Física, Biomedicina, Geología, Ingeniería Química, etc. Incluye referencias bibliográficas y resúmenes de artículos, informes, y libros entre otras cosas. Permite realizar búsquedas por estructura, nombres de sustancias o identificadores, reacciones en la que participa dicha sustancia, artículos y publicaciones que nombren el compuesto en cuestión, e incluso proveedores de

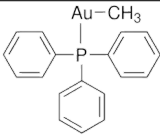
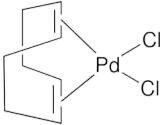
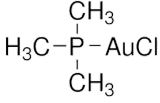
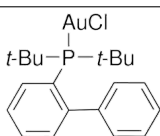
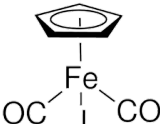
Código SMILES	Representación 2D
<chem>C[Au].c1ccc(cc1)P(c2ccccc2)c3ccccc3</chem>	
<chem>Cl[Pd]Cl.C1CC=CCCC=C1</chem>	
<chem>Cl[Au].CP(C)C</chem>	
<chem>Cl[Au].CC(C)(C)P(c1ccccc1-c2ccccc2)C(C)(C)C</chem>	
<chem>[Fe]I.[C-]#[O+].[C-]#[O+].[CH]1[CH][CH][CH][CH]1</chem>	

Tabla 1.1: Códigos SMILES y sus representaciones visuales según Sigma-Aldrich

compra [23]. Para el uso de esta herramienta es necesario acceder mediante la red de una institución autorizada (en este caso trabajo mediante VPN de la UGR) y seguir los pasos para registrarte ¹. Y *Sigma-Aldrich*, una compañía de ciencia, química y biotecnología que se dedica a la producción y venta de productos químicos, reactivos, equipos y materiales de laboratorio. Ofrece herramientas, servicios, artículos y una gran variedad de productos químicos que se utilizan en investigación, biofarmacéutica, e industria entre otros ámbitos [25]. A través de su página web se enfocan al comercio electrónico pudiendo buscar y comprar productos, compuestos orgánicos e inorgánicos, agentes reactivos, isótopos para síntesis químicas, proteínas, enzimas, etc. De cada producto muestra información relevante como la ficha de datos de seguridad, detalles de las propiedades físicas y químicas así como algunas representaciones lineales del compuesto y la representación molecular en 2D.

Desde la Universidad de Granada, la tutora de este TFG colabora con el grupo de investigación del ICIQ (Instituto Catalán de Investigación Química) liderado por la profesora Mónica H. Pérez-Temprano. Su foco de investigación gira en torno al entendimiento de transformaciones catalíticas en

¹Pasos para el registro en SciFinder https://bibliotecaugr.libguides.com/scifinder_scholar

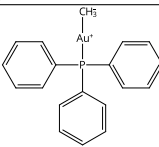
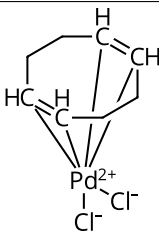
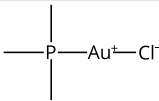
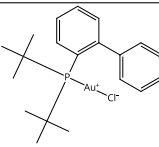
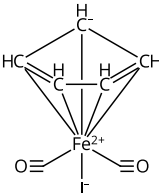
Código SMILES	Representación 2D
<chem>[Au+](CH3-)[P](C=1C=CC=CC1)(C=2C=CC=CC2)C=3C=CC=CC3</chem>	
<chem>[Cl-][Pd+2]123([Cl-])[CH]=4CC[CH]3=[CH]2CC[CH]41</chem>	
<chem>[Cl-][Au+][P](C)(C)C</chem>	
<chem>[Cl-][Au+][P](C=1C=CC=CC1C=2C=CC=CC2)(C(C)(C)C)C(C)(C)C</chem>	
<chem>O#C[Fe+2]1234([I-])(C#O)[CH]=5[CH]4=[CH]3[CH-]2[CH]51</chem>	

Tabla 1.2: Códigos SMILES y sus representaciones visuales según SciFinder

las que participan compuestos organometálicos, descubriendo y diseñando reacciones más eficientes basadas en catalizadores metálicos (para más detalle sobre el grupo de investigación y sus ámbitos de trabajo, ver su sitio web [21]). En resumen, intentan desarrollar enfoques más sostenibles para la síntesis de moléculas orgánicas usando la química organometálica. Como tal, necesitan codificar correctamente una molécula de organometálica en pos de trabajar con ella adecuadamente y utilizar todas las herramientas, para, entre otras cosas, poder dibujarla y entenderla mejor.

Uno de los principales problemas que se detectan en este ámbito es la heterogeneidad en las distintas bases de datos para un mismo compuesto o molécula. Para ilustrar esto, presento las Tablas 1.1 y 1.2. Ambas tablas comparan las mismas moléculas, mostrando el código SMILES y la representación visual que ofrecen las bases de datos Sigma-Aldrich y SciFinder respectivamente. Vemos diferencias claras en el tratamiento de los ciclos aromáticos, la especificación de las cargas de los átomos y la posición de

algunas ramificaciones. Utilizo un subconjunto de 5 moléculas pertenecientes a la organometálica, seleccionadas desde un conjunto de datos de 30 moléculas considerados de interés por los químicos del ICIQ (disponible para su consulta en GitHub). En el Apéndice ??, se puede consultar una tabla comparativa con el set de moléculas al completo.

1.2. Objetivos

Por tanto, el objetivo principal de este Trabajo Fin de Grado es mejorar las herramientas existentes para trabajar con chemoinformatics, adaptándolas a moléculas organometálicas. Para ello, se establecen los siguientes subobjetivos:

- Analizar las distintas bases de datos químicas disponibles y evaluar similitudes y diferencias en el almacenado de moléculas organometálicas.
- Diseñar una metodología para representar de forma canónica una molécula organometálica en formato SMILES.
- Mejorar la visualización de moléculas organometálicas representadas en formato SMILES.

1.3. Estructura de la memoria

esperar a tenerla mas avanzada para completar esto

Capítulo 2

Estado del arte y fundamentos teóricos

Quizas sea mejor mover esta seccion justo despues de la introduccion para seguir con la tematica de la motivacion, y ya luego me meto con la gestion y planificacion

Puedo hacer una revision de la literatura existente hasta dia de hoy sobre el tema Usar SCOPUS para esto, con terminos tipo: "SMILESmolecule organometalic" (juntarlos o separarlos segun vea)

Hablar por aqui de la organometalica, representacion de moléculas, Hablar mas extendido de SMILES, SELFIES, e INCHI; cosas de dibujado de moleculas (los paquetes que hay),) No se si meterlo aqui o en otro apartado, el diagrama de clases

la historia de la humanidad está marcada por la búsqueda de materiales que mejoren su calidad de vida, y los metales han sido parte crucial de esos cambios

La materia que forma los seres vivos tiene en su composición sustancias cuya base principal es el carbono. El estudio de estos compuestos constituye una rama de la química llamada química orgánica. La abundancia del carbono en el planeta es relativamente pequeña: aproximadamente un 0,03 %; sin embargo, da lugar a millones de sustancias diferentes, mientras que los compuestos inorgánicos son solo unos pocos miles. ¿Qué hace a este elemento tan especial? Su estructura singular, que le permite formar largas cadenas en las cuales una pequeña variación da lugar a un compuesto distinto al anterior.

(del archivo de informe Reuniones puedo ir sacando cosas para meter en el estado del arte) Openbabel como tal no soporta el dibujado en 3D de las moléculas, por lo que los únicos dibujos que puede hacer son en 2D.

Openbabel tampoco soporta el uso de wedge ni hash bonds para el dibujo de moléculas en 2D con perspectiva (es curioso porque en el código sí que hay funciones dedicadas a esto, pero luego cuando le metes símbolos SMILES de @@ y demás, los medio ignora. Sigo ejemplos del tutorial de Daylight, pero no salen los mismos dibujos. Otros símbolos más dedicados a geometría que viene en Daylight o en OpenSMILES, tipo @SP, @TB, @OH, los ignora por completo). Pero sí es capaz de generar archivos .sdf con información 3D, que se pueden usar en otros softwares de dibujo específicos como Avogadro (y no están mal, mucho mejor que los 2D desde luego)

Capítulo 3

Gestión y Planificación del proyecto

3.1. Metodología

En el pasado, el desarrollo de software seguía un enfoque ad hoc (software a medida) y poco estructurado, lo que llevaba a problemas como retrasos, presupuestos desbordados, productos finales que no cumplían con las expectativas o proyectos inmanejables y difíciles de mantener. Era frecuente por tanto proyectos fallidos o de mala calidad. Como resultado, surgió la necesidad de establecer un marco de trabajo más formal y disciplinado para el desarrollo de software. Las metodologías de desarrollo proporcionan pues, un marco de trabajo y un conjunto de métodos que guían a los equipos de desarrollo a lo largo del ciclo de vida del proyecto. Es por tanto, a día de hoy, fundamental elegir una metodología que se adapte bien al proyecto. Por lo general, las metodologías de desarrollo se clasifican en dos grandes grupos, tradicionales y ágiles. Se resume en la Tabla 3.1, las diferencias entre ambas metodologías.

Para este proyecto hay varias razones por las cuales elegir una metodología ágil:

- Se trata de un proyecto de investigación, donde se intentarán desarrollar varios algoritmos para la resolución de un problema. Por lo que, a priori no se conoce la calidad de los resultados que se obtendrán. Se deberán realizar experimentos iniciales para detectar los problemas, desarrollar una posible solución y volver a realizar experimentos para evaluar los resultados hasta alcanzar unos que cumplan con los objetivos o sean lo suficientemente satisfactorios.
- Se hará uso de herramientas en las que no se posee experiencia pre-

	Tradicionales	Ágiles
Enfoque	Secuencial	Iterativo e incremental
Planificación	Detallada y exhaustiva	Adaptativa y flexible
Gestión cambios	Difícil de manejar	Fomenta la adaptabilidad
Requisitos	Definidos desde el inicio	Evolucionan con el tiempo
Entrega software	Al final del proyecto	Continua, en incrementos
Colaboración	Menos énfasis	Fomenta la colaboración
Equipos	Mejor en equipos grandes	Mejor en equipos pequeños
Adaptabilidad	Menor flexibilidad	Mayor flexibilidad
Retroalimentación	Al final del proyecto	Constante y temprana

Tabla 3.1: Tabla comparativa de las principales características entre las metodologías tradicionales y ágiles

via, lo que podrá originar retrasos en la planificación o modificaciones frecuentes.

- Las metodologías ágiles se ajustan mejor en equipos pequeños como indico en la Tabla 3.1. En este caso el equipo de trabajo solo tiene una persona.
- La figura del tutor y la experta en química del ICIQ serán importantes para dar feedback continuo durante el proyecto y opinar sobre la calidad de los resultados.

Por las anteriores razones, una metodolgia clásica no se adaptaría bien al proyecto, prefiriendo una ágil. Permite una mayor flexibilidad con las tareas, y un desarrollo incremental en base a los resultados intermedios que se vayan obteniendo.

Dentro de las ágiles existen varias metodologías como Scrum, eXtreme Programming, Lean Development, Kanban, Crystal, DSDM, etc. Las más comunes hoy día son Scrum y XP, al ser las que mejores resultados obtienen en proyectos de desarrollo software [15, 13, 9, 16]. Hay que tener en cuenta que la mayoría de metodologías imponen una serie de prácticas y principios para guiar al equipo de desarrollo durante el proyecto. A algunos autores como Mike Cohn (fundador de la Scrum Alliance) les gusta tratar las metodologías como marcos de trabajo y no como una serie de métodos y reglas estrictas que haya que seguir al pie de la letra [5].

El perfil del proyecto no terminaba de encajar con ninguna metodología al completo, por lo que se ha aplicado un híbrido entre Scrum y XP, mez-

clando aspectos y técnicas de ambas según se ajustaban al proyecto. Scrum maneja la parte administrativa del proyecto, definiendo cómo se especifica el trabajo y el proceso de entrega de características, mientras se aplican algunas prácticas propias de XP para la parte de codificación[12, 24]

3.1.1. Scrum

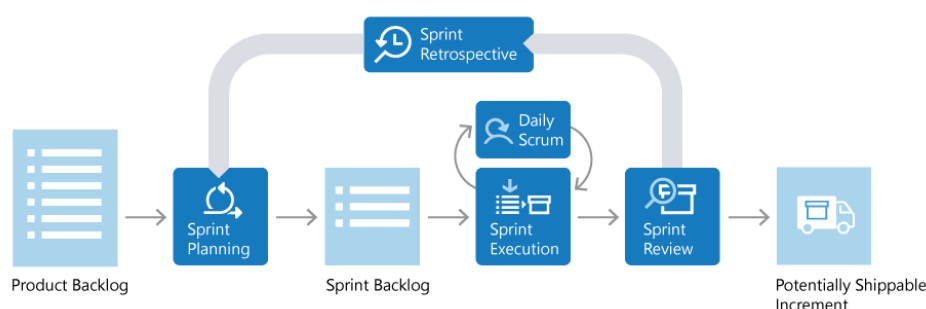


Figura 3.1: Ciclo de vida iterativo de Scrum.

El ciclo de vida de Scrum (Figura 3.1, extraída de microsoft learn¹) se basa en un enfoque iterativo e incremental que permite a los equipos adaptarse a los cambios y entregar productos de alta calidad en un tiempo reducido. Los principios de transparencia, inspección y adaptación son fundamentales para alcanzar los valores centrales de Scrum, la calidad, flexibilidad, mejora continua, compromiso, coraje, ritmo y responsabilidad. Una de las características principales de Scrum es su enfoque en ciclos de desarrollo cortos llamados *sprints*. Estos sprints suelen tener una duración de una a cuatro semanas, durante los cuales se planifican, desarrollan, prueban y entregan incrementos de software funcionales. Cada sprint comienza con una reunión de planificación en la que el equipo selecciona un conjunto de tareas para ser completados durante el sprint. Podemos definir Scrum según los elementos que lo componen:

■ Artefactos

- **Product Backlog (Pila del producto):** es una lista ordenada y priorizada de todas las funcionalidades, características y mejoras que podrían ser necesarias para el producto. Es responsabilidad del Product Owner y se actualiza constantemente a medida que se obtiene nueva información o se generan cambios en los requisitos.
- **Sprint Backlog (Pila del sprint):** es un conjunto de elementos seleccionado del Product Backlog para el sprint. Se descomponen

¹<https://www.scrum.org/>

en tareas de desarrollo más pequeñas junto con estimaciones de tiempo, expresando los requisitos en lenguaje técnico.

■ **Reuniones**

- **Sprint Planning:** marca el inicio de cada sprint y se realiza con el propósito de identificar el objetivo principal del sprint y las tareas concretas que se van a desarrollar en él. Como resultado, se genera el Sprint Backlog.
- **Daily Meetings:** reunión diaria de unos 15 minutos donde participan los miembros del Equipo de Desarrollo y el Scrum Master. Es una manera de estar al tanto del trabajo realizado y cuáles son los siguientes pasos. Es una oportunidad de identificar rápidamente obstáculos o problemas.
- **Sprint Review:** al final del sprint se pone en común todo el trabajo realizado durante el Sprint. Sirve para recoger información o feedback sobre el estado del proyecto.
- **Sprint Retrospective:** el equipo dedica tiempo a reflexionar sobre los aspectos positivos y las áreas que requieren mejoras. Como resultado de la retrospectiva, se generan acciones específicas para implementar en el siguiente sprint.

■ **Roles:** representan responsabilidades dentro del proyecto.

- **Stakeholders:** son todos aquellos interesados en el proyecto, tanto personas como organizaciones (gente de marketing, comerciales, usuarios, etc).
- **Product Owner (PO, propietario):** debe conocer perfectamente el entorno de negocio del cliente, las necesidades y el objetivo que se persigue con el sistema que se está construyendo. Debe conocer también como funciona Scrum para desempeñar bien su rol. Su responsabilidad principal es la de crear, administrar, y priorizar el P.Backlog, así como validar o rechazar el incremento resultado de cada iteración.
- **Scrum Master (director del proyecto):** garantiza el correcto funcionamiento de los procesos y metodologías que se empleen en el equipo. Gestiona el proceso e intenta mejorar la productividad del equipo. Promueve los valores y prácticas de Scrum, elimina impedimentos, facilita la colaboración entre los roles, actúa como escudo ante cosas externas. Se asegura de que el PO sepa cómo ordenar la pila de producto para maximizar el valor generado en cada sprint.
- **Equipo de desarrollo:** es el que se encarga de desarrollar el producto y hacer los entregables en incrementos. Los miembros del

equipo necesitan ser auto-organizados, multidisciplinares, multifuncionales, con un alto compromiso y sin jerarquías internas. Son los verdaderos responsables de que el producto salga adelante y se completen los incrementos. Se encargan de estimar el tamaño de los ítems del backlog. Es importante que el equipo de desarrollo comprenda bien la visión que tiene el PO acerca del producto. Suelen estar formados de entre 5 a 9 personas.

3.1.2. XP

La programación extrema (XP) es una metodología ágil que se centra en la velocidad y la simplicidad con ciclos de desarrollo muy cortos. En XP se promueven una serie de valores: comunicación, simplicidad, retroalimentación, coraje y respeto. Diseñada para entornos dinámicos con requisitos cambiantes y orientado fuertemente hacia la codificación, reduciendo considerablemente la documentación. En XP, las tareas que se terminan son susceptibles de ser modificadas durante el transcurso del proyecto, incluso después de que funcionen correctamente, por lo que son importantes las siguientes prácticas.

- **Prácticas:** XP propone una serie de prácticas a nivel técnico que se debería adoptar para el desarrollo del proyecto. Las más importantes a mi parecer son: [20].
 - **Programación a pares:** los programadores trabajan en parejas, mientras uno escribe el código, el otro proporciona comentarios y realiza revisiones en tiempo real. Esto promueve el intercambio de conocimientos, la revisión de código constante y la minimización de errores.
 - **Propiedad colectiva del código:** la propiedad colectiva anima a todos a aportar nuevas ideas sobre todos los segmentos del proyecto. Cualquier desarrollador puede cambiar cualquier línea de código para añadir funcionalidad, corregir errores, mejorar diseños o refactorizar.
 - **Estándares de codificación:** el código ajustarse a las normas de codificación acordadas. Estas hacen que el código sea consistente y fácil de leer y refactorizar para todo el equipo. Un código con el mismo aspecto o que sigue unas normas fomenta la propiedad colectiva.
 - **Marcha sostenible:** encontrar un ritmo de trabajo para el equipo de desarrollo donde todos los miembros se sientan cómodos. Las horas extra acaban con el espíritu y la motivación del equipo. A veces, menos es más.

- **Integración continua:** los equipos de XP no esperan a que se completen las iteraciones, sino que se integran constantemente. Se cuenta con un repositorio de código donde los desarrolladores envían el código cada poco tiempo (pocas horas a veces).
- **Refactorización:** reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo. Evita la complejidad innecesaria.
- **Desarrollo orientado a pruebas:** las pruebas son frecuentemente repetidas y automatizadas cada vez que se haga un cambio, por pequeño que sea, antes de desplegar la nueva versión. Se suelen escribir antes que el propio código.

■ Roles

- **Programador:** encargados de escribir y probar el código.
- **Cliente:** representa los intereses del usuario y es responsable de proporcionar las necesidades, requisitos del software y establecer prioridades.
- **Entrenador (Coach):** es el líder del equipo, actúa como facilitador y promotor de las prácticas y valores de XP, y ayuda al equipo a mejorar y adaptarse.
- **Consultor:** miembro externo al equipo de desarrollo con conocimiento específico en un tema necesario.
- **Rastreador (Tracker):** se encarga de gestionar la planificación y llevar un seguimiento del proyecto detectando los problemas en él.

3.1.3. Aplicación de Scrum/XP al proyecto

Algunas de las características y prácticas mencionadas de cada una de las metodologías no son aplicables al proyecto dada su naturaleza e integrantes, como por ejemplo, la programación por parejas propia de XP. Se describe ahora, el enfoque que se le ha dado del híbrido Scrum/XP al proyecto.

Se lleva a cabo una planificación por *Sprints* de entre 2 a 4 semanas, dependiendo de las tareas a realizar. Entre medias y al final de cada sprint, se agendará una *reunión con la tutora* en la que se hará retrospectiva del mismo, para comprobar el estado y avance del proyecto. Se revisará qué se ha hecho durante el sprint y cómo se ha hecho, repasando las novedades desde la última iteración y puntos a mejorar o pulir, priorizando las siguientes tareas

a realizar en base a los resultados. Igualmente, se mantendrá el contacto con el tutor de manera constante vía correo electrónico.

Mediante la *integración continua*, los cambios se envían con frecuencia a un repositorio compartido con un sistema de control de versiones. Cada vez que se realiza un cambio o se desarrolla una nueva funcionalidad, se ejecutan *pruebas* en el dataset completo de moléculas, comprobando rápidamente si los resultados son los esperados (parcial o totalmente), y detectar cualquier error antes de que se convierta en un problema más grave. Se sigue por tanto un *desarrollo incremental*, añadiendo pequeñas funcionalidades o porciones de código funcional, centrándose en un aspecto específico en cada commit subido a github.

Se asignarán los roles propios de Scrum. El papel de *Equipo de desarrollo* recae sobre el estudiante, y al ser únicamente una persona, también actuará como *Scrum Master* siendo responsable de la correcta aplicación de la metodología y prácticas al proyecto. La tutora actuará como *Product Owner*, conoce las necesidades del proyecto y el dominio del problema, hace de intermediaria con el cliente y ayuda al equipo de desarrollo a priorizar las tareas. Como posibles clientes o personas interesadas (*stakeholders*) podemos incluir al grupo de investigación del ICIQ, con el que se mantiene el contacto frecuentemente a través del Product Owner.

Al ser OpenBabel una librería existente es importante mantener unos *estándares de codificación* y un estilo consistente. Teniendo eso en mente, y usando el *Refactoring*, se realizan cambios manteniendo el código limpio y legible. Se han usado las guías propias de OpenBabel y la documentación oficial orientada a desarrolladores para añadir nuevas funcionalidades ², ³. Además, se busca ante todo la simplicidad a la hora de programar, a través de la eliminación de funcionalidades innecesarias y la adopción de soluciones sencillas.

3.2. Planificación

Inicialmente se elaboró una planificación estimada general sin mucho detalle, dividida en bloques grandes de trabajo, para establecer marcos de tiempo y controlar el ritmo del proyecto. Se puede ver en la Figura 3.2

Conforme se iba profundizando en el dominio del problema y entendiendo más sobre la problemática, se fueron subdividiendo y priorizando las tareas. A continuación, se desglosan las tareas definidas para el proyecto, separados en sprints:

²<https://openbabel.org/docs/current/OpenBabel.pdf>

³https://acortar.link/Openbabel_Adding_Plugins

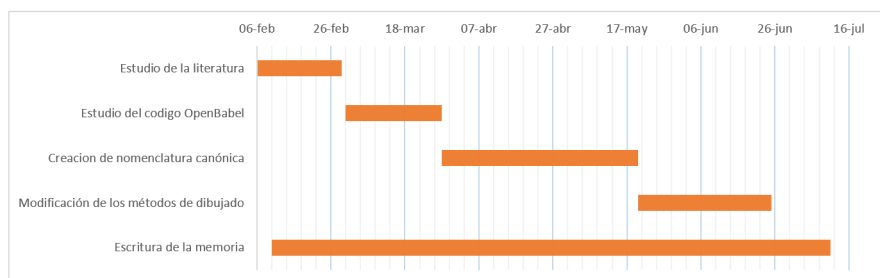


Figura 3.2: Diagrama de Gantt sobre la planificación inicial estimada

- **Bloque 1:** Investigación y aprendizaje previo. Estudio del estado del arte.
 - Lectura artículos y publicaciones sobre Chemoinformatics
 - Repaso de química general y estudio de química organometálica
 - Experimentación y pruebas iniciales con distintas herramientas
- **Bloque 2:** Estudio del paquete OpenBabel
 - Lectura detallada del código OpenBabel
 - Experimentación moléculas usando el dibujo de OpenBabel
- **Bloque 3:** Proceso de mejora del sistema de dibujo
 - Detección de estructuras de ciclopentadienilo (Cp)
 - Modificación dibujo moléculas con Cp individuales
 - Detección de múltiples Cp en la misma molécula usando bloques
- **Bloque 4:** Creación de nomenclatura canónica
 - Detección de bloques en el SMILES y creación del árbol genérico
 - Desarrollo del algoritmo canónico para moléculas con 1 metal
 - Algoritmo canónico para moléculas con 2 o más metales
- **Bloque 5:** Documentación del proyecto
 - Redacción de la memoria

En el siguiente Diagrama de Gantt (Figura 3.3) se presenta la planificación real del proyecto:

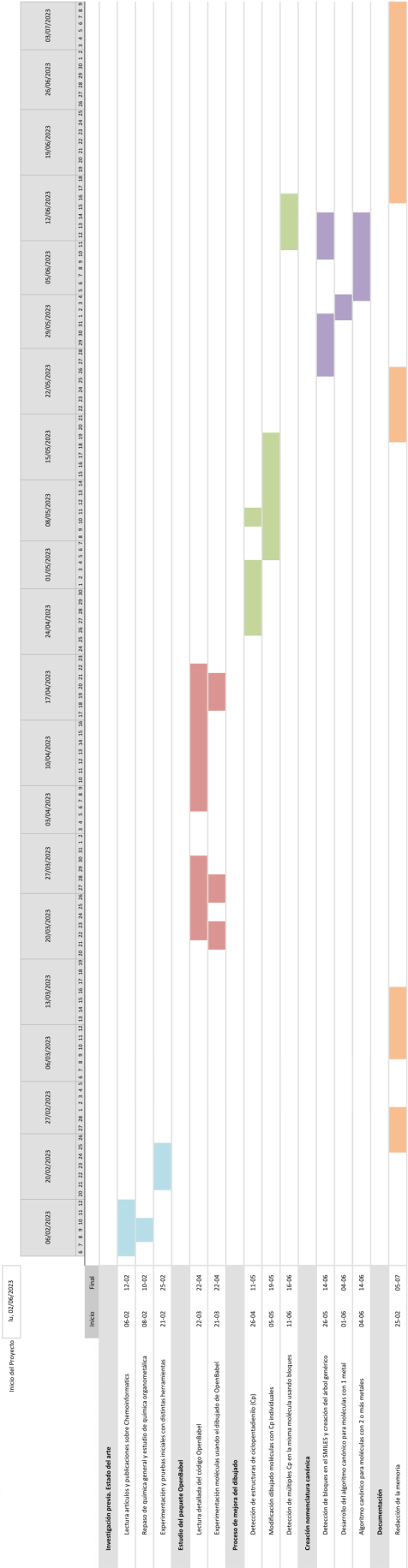


Figura 3.3: Diagrama de Gantt sobre la planificación temporal real

Como describo en la Sección 3.6.1, el proceso de entendimiento del código base se alargó más de lo esperado, retrasando un poco el resto de tareas. En contraparte, la canonización y dibujado de las moléculas me llevó menos tiempo del estimado. Además, se invirtió el orden entre la modificación del sistema de dibujado y la creación del algoritmo de canonización con respecto a la planificación inicial. A priori era más sencillo alterar esa parte del código, y se podían obtener resultados más visuales.

3.3. Gestión de la configuración

En este apartado se describirá cómo se ha llevado a cabo la gestión de los activos de este proyecto, es decir, el código desarrollado y la documentación generada. Dado que son partes fundamentales del trabajo, atendiendo al análisis de riesgos descrito en la sección 3.6 y el plan de actuación frente a la pérdida de información, se detalla a continuación la gestión de ambas.

3.3.1. Gestión del código

Para la gestión del código se ha usado un control de versiones a través de Git y Github. Ambas herramientas en conjunto permiten ir creando versiones intermedias del código conforme se va desarrollando y hacer copias de seguridad en la nube. También son muy útiles para proyectos colaborativos, donde varias personas del equipo pueden combinar fácilmente su parte del código desarrollado y revisar el progreso subido hasta el momento. El procedimiento a seguir en la mayoría de casos es bastante similar. Se creará un repositorio remoto en la plataforma de GitHub con el nombre de *TFG* —o el que uno prefiera— donde se irán almacenando los cambios⁴. En la carpeta de trabajo local de nuestra computadora, carpeta que contendrá en mi caso todos los elementos de los que quiera llevar un control, se creará un repositorio local usando Git, que habrá que vincular con el remoto para poder ir sincronizando los cambios.

3.3.2. Gestión de la documentación

En lo relativo a la documentación, el proceso de gestión será similar ya que también se ha usado GitHub para su control de versiones. Esta se ha redactado en LaTeX usando el servicio online Overleaf. Overleaf cuenta con una opción para sincronizar el proyecto con un repositorio de GitHub, pero es una opción de pago. En su lugar, descargo el proyecto en el repositorio

⁴<https://github.com/Jesnm01/TFG>

local y desde ahí ya realizo dicha sincronización para subir los cambios en el remoto.

Además, dada la naturaleza del proyecto y de la metodología de desarrollo utilizada, se ha llevado un registro de las reuniones con la tutora que también se ha ido actualizando periódicamente. Este documento pretende recoger los contenidos más relevantes de las reuniones: preguntas, comentarios, anotaciones, tareas que hacer, cosas pendientes de una reunión a otra, revisiones, y puntos a mejorar, entre otras cosas.

3.4. Gestión de recursos

3.4.1. Recursos humanos

- **Dña. Rocío Celeste Romero Zaliz**, profesora del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada en calidad de tutora del proyecto. A cargo de la supervisión y guía del alumno durante su desarrollo del trabajo.
- **Jesús Navarro Merino**, estudiante del grado en Ingeniería Informática en la Escuela Técnica Superior de Ingenierías Informática y Telecomunicación.

3.4.2. Recursos materiales

Para este proyecto no se han necesitado recursos adicionales, habiéndose usado únicamente los siguientes recursos materiales, ya existentes:

- **Portátil personal**: Portátil ACER Aspire A515-51G-8907 con un procesador Intel Core i7 8550U 1.8GHz, 20GB de memoria RAM y una arquitectura de 64 bits. Se ha usado durante todo el proyecto, para labores de programación y redacción de la memoria.
- **Pantalla**. Monitor utilizado de apoyo a la pantalla propia del portátil. De la marca AOC, de 24 pulgadas con una resolución de 1920x1080.

3.4.3. Recursos software

En esta sección describiré todas aquellas herramientas software empleadas durante la realización del proyecto. Todas y cada una de ellas son herramientas de software libre, gratuitas o disponibles a través de licencias de estudiantado. A continuación se lista el software usado:

- **Sistema operativo:** Windows 10 Home. Aunque por lo general Windows no es gratuito, al estar utilizando la típica licencia OEM que trae preinstalada el ordenador al comprarlo, la considero como tal.
- **Visual Studio C++:** es un IDE muy potente de Microsoft orientado a crear aplicaciones .NET y C++ para Windows. Se ha usado su versión gratuita Visual Studio Community 2022. Permite editar, depurar, realizar pruebas de testing, además de tener control de versiones integrado, entre otras cosas. Aquí se ha llevado a cabo todo el desarrollo del código.
- **OpenBabel:** Open Babel es una biblioteca de código abierto multi-plataforma utilizada en química computacional y ciencias relacionadas para la conversión y manipulación de estructuras químicas en varios formatos. He trabajado con la versión 3.1.1 disponible en su repositorio de GitHub oficial⁵.
- **CMake:** es una herramienta de generación de archivos de compilación que simplifica el proceso de compilación y construcción de proyectos, permitiendo una configuración flexible e independiente de la plataforma. CMake utiliza archivos de configuración llamados CMakeLists.txt para describir la estructura del proyecto y las dependencias necesarias. Se ha usado en su versión 3.25.2 para la compilación y creación de soluciones de OpenBabel.
- **Git:** software de código abierto para el control de versiones de un proyecto.
- **Github:** es una plataforma donde se alojará el código y la documentación del proyecto. Utiliza Git por debajo y es una de las plataformas gratuitas para alojamiento de código mas empleadas a nivel mundial.
- **Google Colab:** es una plataforma en línea gratuita ofrecida por Google que permite a cualquier usuario escribir y ejecutar código en el navegador. Es una herramienta basada en la nube que proporciona un entorno de ejecución como si fueran notebooks de Jupyter, es decir, se puede escribir, editar y ejecutar código en bloques/celdas interactivos. Se ha usado durante las etapas iniciales del proyecto para la experimentación con diversas moléculas.
- **Zotero:** software de gestión de referencias bibliográficas que permite recopilar, organizar, citar y generar fácilmente una bibliografía en varios estilos de formato estándares según los documentos, páginas webs, artículos o archivos PDF guardados. Facilita la creación de referencias

⁵<https://github.com/openbabel/openbabel>

y citas en documentos académicos, ahorrando tiempo y asegurando un uso correcto de las fuentes consultadas.

- **Google Meet:** servicio de videoconferencias de Google. Plataforma utilizada para las reuniones con la tutora.
- **Google Drive Sync:** ahora llamada Google Drive para PC, es una aplicación de Google que permite sincronizar los archivos y carpetas de la computadora con la cuenta de Drive. Usado para realizar copias de seguridad —adicionales a lo almacenado en GitHub— de algunos archivos importantes.
- **Overleaf:** herramienta online para la redacción de documentos en LaTeX usada para la documentación de este proyecto.
- **Clockify:** herramienta online que te permite registrar las horas dedicadas a un proyecto.
- **Umbrello:** herramienta que combina funciones de modelado y generación de código para el lenguaje unificado de modelado (UML). Usado para la elaboración de los diagramas de clases.
- **Correo UGR:** servicio de correo electrónico institucional de la UGR.
- **Microsoft Word:** procesador de textos de Microsoft usado para apuntes personales y documentación en sucio. Disponible a través de la cuenta de Microsoft Office 365 que ofrece la universidad.
- **Microsoft Excel:** utilizado para la creación de algunas tablas y gráficos incluidos en la memoria. Disponible a través de la cuenta de Microsoft Office 365 que ofrece la universidad.
- **Visual Studio Code y LaTeX:** VSCode es un editor de texto, que a través de algunas extensiones, permite editar, compilar y visualizar ficheros LaTeX. Es la alternativa a Overleaf según lo descrito en la Sección 3.6.

3.5. Gestión de costes

TERMINAR esto cuando vaya acabando el trabajo

En esta sección se realizará una estimación de los costes asociados al proyecto, atendiendo a los recursos descritos en la Sección 3.4. La elaboración de un presupuesto preciso en muchos casos puede suponer un desafío, ya que los proyectos software a menudo están sujetos a cambios y variables imprevistas, pero es una tarea importante para estudiar su viabilidad.

3.5.1. Coste de recursos humanos

Si actuáramos como una empresa en un proyecto software al uso, existen muchos componentes que tener en cuenta para montar esta sección del presupuesto. Aspectos como el salario de los trabajadores y compensaciones varias, el coste de los procesos de selección y contratación, formación y desarrollo del personal, costes laborales adicionales (seguros, vacaciones, etc), o costes de personal externo (consultores o subcontratistas) entre otras cosas. Dada la naturaleza de este proyecto, en relación a los gastos asociados a recursos humanos contamos con un equipo de desarrollo formado por una persona que tendrá el papel de Ingeniero Informático. Para estimar el costo de su trabajo, se indica el número total de horas dedicadas.

El intervalo de tiempo que está pensado para el proyecto son 5 meses, desde febrero hasta junio.

$$Dias\ totales = 5\ meses * 30\ dias/mes = 150\ dias$$

A este tiempo hay que descontarle los días no laborables de fines de semana más los días de vacaciones correspondientes por mes trabajado:

$$Dias\ no\ laborables = \left(\frac{8\ dias}{mes} + \frac{22\ dias\ vacaciones}{12\ meses} \right) * 5\ meses \approx 50\ dias$$

Eso nos deja un total de 100 días aproximadamente de trabajo. Teniendo en cuenta una media semanal de 30 horas trabajadas repartidas en 6 horas diarias, tendríamos el siguiente total de horas trabajadas:

$$Horas\ trabajadas = 100\ dias * 6\ horas/dia = 600\ horas$$

Esta cifra duplicaría la cantidad de horas que le corresponderían a un TFG según sus créditos asignados por la universidad. Estos cálculos como tal serían una estimación, pero se ha utilizado durante el desarrollo del proyecto la herramienta Clockify para el registro real de las horas dedicadas. Como se ve en la figura 3.4, han sido **XXXXX** horas en total, por lo que usaré ese dato para calcular el coste.

Según varias fuentes^{6,7,8}, el salario medio de un ingeniero informático recién graduado está en torno a 21.000€. Siendo equivalente, unos 1750€ mensuales, que son 13,46€/hora. Con todo eso, tenemos que:

$$Coste\ total\ recursos\ humanos = XXXXhoras * 13,46\ €/hora = XXX\ €$$

⁶<https://www.jobted.es/salario/ingeniero-informtico>

⁷https://acortar.link/infojobs_salario

⁸https://acortar.link/uax_salario

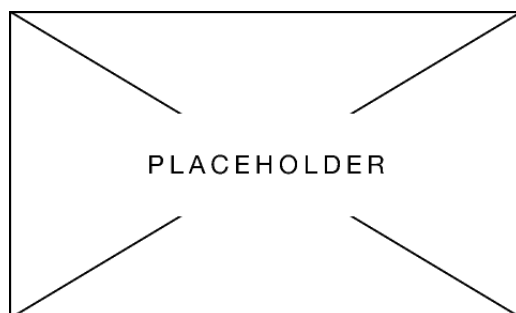


Figura 3.4: Registro de horas dedicadas al proyecto a través de Clockify

3.5.2. Costes de recursos materiales

Dado que no se han adquirido expresamente para este proyecto, ya que se poseían con anterioridad, no se valora su precio de compra como tal sino su valor de depreciación. Los productos electrónicos experimentan un proceso llamado depreciación, que conlleva una devaluación gradual a lo largo de su vida útil. Es importante tener esto en cuenta para estimar su valor actual y el coste del recurso. Esta estimación refleja el valor de un activo desde un punto de vista contable.

Al referirnos a la depreciación de un activo a lo largo de su vida útil, no se incluyen situaciones en las que sufra daños debido a accidentes, desastres naturales u otros eventos similares. En cambio, estamos hablando del desgaste del uso cotidiano, así como los impactos derivados de las innovaciones tecnológicas que surgen durante ese período que puedan dejar obsoleto el dispositivo.

Para calcular el valor actual de los activos utilizaré el método de depreciación lineal, que considera un desgaste uniforme durante su uso, mostrando el resultado del gasto anual de depreciación[17]. Necesitamos lo primero, hallar el valor residual del activo, es decir, el valor que se estima que tendrá cuando llegue al final de su vida útil. Usaré para los dispositivos electrónicos una vida útil de 8 años. Haré un ejemplo con el coste del portátil.

$$\text{Valor residual} = \frac{\text{Coste inicial}}{\text{Vida útil (años)}} = \frac{689 \text{ €}}{8 \text{ años}} = 86,125 \text{ €}$$

Con el valor residual estimado, se puede calcular la depreciación lineal anual con la siguiente fórmula.

$$\text{Depreciación} = \frac{\text{Coste inicial} - \text{Valor residual}}{\text{Vida útil (años)}} = \frac{689 \text{ €} - 86,125 \text{ €}}{8 \text{ años}} = 75,36 \text{ €}$$

Teniendo estos 2 datos, se puede calcular el valor actual del activo tenien-

do en cuenta sus años de antigüedad. Se muestran todos los datos relativos a los costes materiales en la Tabla 3.2.

Recurso	Valor inicial (€)	Depreciación anual (€)	Antigüedad (años)	Valor actual (€)
Portátil personal	689	75,36	5	312,2
Pantalla AOC	230	25,15	2	179,69
Total:				491,89

Tabla 3.2: Tabla de los costes materiales

3.5.3. Costes software

Los costes relacionados con los recursos software son nulos. Como indico en el listado de recursos de la Sección 3.4.3, son herramientas de software libre o utilizadas mediante licencias gratuitas, por lo que no suponen coste alguno en el desarrollo de este proyecto.

3.5.4. Otros costes

Aquí se incluyen todos los demás gastos que han sido necesarios para el desarrollo del proyecto y que no pertenecen a los apartados anteriores. Principalmente son los gastos vinculados a las facturas de la luz e Internet. El coste de la tarifa de Internet contratada es de 40€/mes, que a lo largo de los 5 meses el total asciende a 200€. Para la luz, una estimación posible serían 19,58€, teniendo en cuenta el gasto que suponen los recursos materiales en base a una factura trimestral de 11,80€.

Recursos	Importe (€)
Luz	19,58
Internet	200

Tabla 3.3: Tabla de costes adicionales

3.5.5. Presupuesto final

Se presenta por tanto el presupuesto completo asociado al proyecto, dividido en cada una de las secciones tratadas anteriormente. Se ve el desglose en la Tabla 3.4.

Detalle	Importe
Costes de recursos humanos	X €
Trabajo autónomo	X €
Costes de recursos materiales	491,89 €
Portátil personal	312,2 €
Pantalla de apoyo	179,69 €
Costes de recursos software	0,00 €
Windows 10	0,00 €
Visual Studio C++	0,00 €
OpenBabel	0,00 €
CMake	0,00 €
Git	0,00 €
GitHub	0,00 €
Google Colab	0,00 €
Zotero	0,00 €
Google Meet	0,00 €
Google Drive Sync	0,00 €
Overleaf	0,00 €
Clockify	0,00 €
Correo UGR	0,00 €
Microsoft Word	0,00 €
Microsoft Excel	0,00 €
Visual Studio Code y LaTeX	0,00 €
Costes adicionales	219,58 €
Internet	40€ x 5 meses = 200€
Factura de la luz	19,58 €
Total:	X €

Tabla 3.4: Presupuesto total del proyecto

3.6. Análisis de riesgos

El análisis de riesgos desempeña un papel fundamental en la planificación y ejecución exitosa de proyectos. A veces surgen imprevistos que pueden afectar en mayor o menor medida a la correcta evolución de estos. Por ello, la aplicación de este proceso resulta esencial para minimizar la incertidumbre, intentar evitar la aparición de esos riesgos, y en caso de que se materialicen, paliarlos o mitigarlos de manera efectiva mediante unos planes de actuación.

En este apartado por tanto, se analizarán los riesgos potenciales del proyecto, incluyendo sus causas y el plan de acción para resolverlos o mitigar

ID	Riesgo	Causa	Plan de acción
R.1	Pérdida de documentación y/o código	Eliminación accidental de ficheros clave	Realizar copias de seguridad y herramientas de control de versiones para su recuperación.
R.2	Fallo en el hardware de trabajo	Fallo irreparable por edad de uso, sobrecalentamiento, golpe o rotura.	Adquisición de nuevo portátil. Instalación y configuración del entorno de trabajo cuanto antes.
R.3	Falta de conocimiento de los paquetes de chemoinformatics	Nula experiencia con toolkits químicos y sus prestaciones	Invertir tiempo en el estudio y comprensión del código base.
R.4	Cambios inesperados y tareas nuevas no contempladas	Planificación poco adecuada a las necesidades del proyecto	Replanificación de las tareas existentes y nuevas, con una adecuada asignación de tiempo
R.5	Problemas en la instalación y configuración de las herramientas	Incompatibilidad de versiones, la máquina en donde se va a realizar la instalación o complejidad del proceso	Consulta de las guías de instalación, búsqueda de herramientas alternativas viables o uso de máquinas virtuales/entornos de desarrollo distintos
R.6	Falta de feedback directo con el tutor	Incompatibilidades horarias, o enfermedad de alguna de las partes	Agendar una reunión en otra fecha, y si fuera necesario o viable, enviar el material a revisar/preguntas por correo para feedback asíncrono.
R.7	Los objetivos del proyecto no son realistas	Se ha infravalorado el alcance del proyecto	Realizar una planificación coherente con el tiempo y recursos disponibles. Si fuera necesario, adaptar el contenido del trabajo original y proponer los cambios
R.8	Los resultados del TFG finalmente no son satisfactorios o los esperados	A la tutora (y los químicos colaboradores) no les gusta el resultado	Recibir feedback periódicamente de los cambios realizados y opiniones sobre los resultados intermedios.
R.9	Dificultades para crear un sistema canónico viable para cualquier molécula de entrada	La tarea de programación para llevar esto a cabo es más compleja de lo que se creía	Proponer, aunque sea un inicio de canonización y detallar formalmente las reglas que lo definen. O, sabiendo cómo funcionan internamente las librerías, buscar un sistema canónico más sencillo en términos de programación.
R.10	Imposibilidad de acceso al servicio de Overleaf para la redacción de la memoria	Caída temporal de los servidores	Si es muy urgente, uso de Visual Studio Code para compilado local de la documentación.
R.11	Finalización del uso gratuito de alguna herramienta utilizada	Conclusión de las licencias gratuitas o periodos de prueba	Extensión de la licencia en caso de ser posible o búsqueda de herramientas alternativas parecidas/compatibles.

Figura 3.5: Riesgos del proyecto, causas, y planes de actuación

su impacto al máximo (Figura 3.5). Además, se realizará una evaluación de la probabilidad de ocurrencia y del impacto asociado a cada riesgo, que se puede ver en la Figura 3.6. Esto se basa en una matriz con 2 dimensiones: la probabilidad de ocurrencia de un riesgo y el impacto que tendría en el proyecto si se materializa. Se valorarán los riesgos según aspectos técnicos, de recursos humanos o complejidad y naturaleza del proyecto, y preguntas del tipo, ¿qué tan difícil sería recuperarse del riesgo?, ¿cuál es el resultado más negativo que podría originarse como consecuencia? o ¿ha sucedido este riesgo o alguno similar anteriormente?

	Probabilidad				
Impacto	Muy improbable (0,1)	Poco probable (0,3)	Moderada (0,5)	Probable (0,7)	Casi seguro (0,9)
Muy bajo					
Bajo	R.11	R.12	R.4		
Medio	R.6	R.9	R.8		
Alto	R.1			R.3	
Muy alto	R.2		R.5, R.10		

Figura 3.6: Matriz de probabilidad-impacto de riesgos

3.6.1. Riesgos materializados

completar esto al final del trabajo, o conforme se vayan ocurriendo

Los riesgos materializados han estado relacionados principalmente con aspectos técnicos. Primeramente, el R.5. Tuve problemas para instalar OpenBabel en mi máquina por problemas de versiones, por lo que acabé utilizando Google Colab como entorno virtual e instalar ahí algunas librerías necesarias para las primeras experimentaciones con moléculas. Esto tampoco era muy útil a largo plazo puesto que tenía que poder acceder al código fuente para modificarlo, añadir las funcionalidades y compilarlo manualmente para probar los cambios. Por lo que finalmente con ayuda de las guías, se pudo ejecutar localmente. Instantáneamente después, se materializó el riesgo R.3. La falta de conocimiento ante una librería tan grande ya existente retrasó considerablemente el proceso de modificación del código. **Añadir mas riesgos conforme se vayan materializando**

Finalmente, se consiguieron solventar los riesgos manifestados mediante los planes de actuación descritos en cada uno de ellos.

Capítulo 4

Diseño

En esta sección se describirán las clases y todos los métodos que se han añadido y modificado durante el proceso de implementación.

4.1. Diagrama de Clases

Al compilar los archivos fuente y generar el proyecto, los archivos de configuración de CMake generan una serie de *soluciones*. Hay soluciones que consisten únicamente en el archivo *'main'*, que representa el ejecutable al que llamamos por línea de órdenes desde la terminal. El resto de soluciones forman la propia API de OpenBabel, teniendo como clases principales *'OBMol'*, *'OBAtom'*, y *'OBBond'*, que permiten almacenar la información de una molécula, un átomo, o un enlace entre átomos respectivamente; y otras clases más orientadas a la conversión entre formatos como *'OBConversion'* u *'OBFormat'* (se explica más en detalle la estructura del proyecto OpenBabel y el proceso de compilación en el Anexo ??).

Para este trabajo, se ha necesitado modificar algunas clases de la API y añadir otras nuevas. En el siguiente Diagrama de clases (Figura 4.1) se muestran tanto las clases que se han visto modificadas (en color anaranjado), las creadas desde cero (en color más verdoso) y las demás clases importantes que interactúan con las anteriores pero no se han visto alteradas (en amarillo).

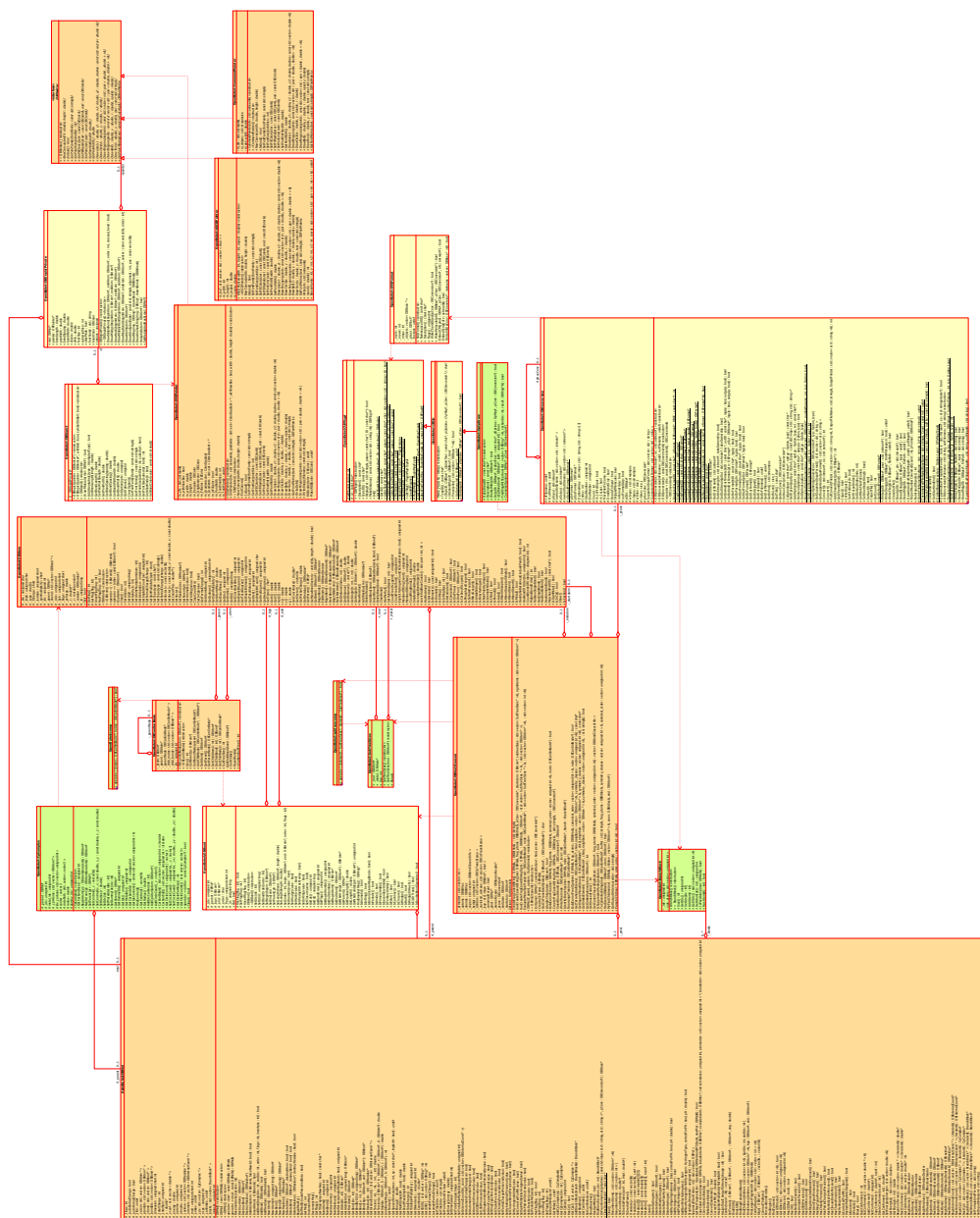


Figura 4.1: Diagrama de clases

Se pasa a detallar ahora cada una de ellas, tanto las modificadas como las nuevas, para qué sirven, y en qué consisten sus métodos brevemente. Las clases que no se han alterado, al igual que el resto de clases que no se incluyen en el diagrama se puede consultar su documentación en la página oficial ¹. Puntualizar que existe una enorme cantidad de clases en la librería de OpenBabel, no tendería sentido añadirlas todas en el diagrama. Además, no todas poseen de documentación, por lo que la mayoría no aparecerán en la página anterior.

4.2. Clases modificadas

- **OBPainter**: clase base abstracta para las clases de representación gráfica 2D. Se han añadido el siguiente método para poder utilizarlo en las clases que implementan esta interfaz:

```
public:

virtual void DrawPolygonLine(const std::vector<std::pair<double,
double> >& points) = 0;
```

- **SVGPainter**: clase que hereda de OBPainter y genera representaciones 2D en el formato de gráficos vectoriales SVG.

```
public:

//Inserts the necessary xml code in the .svg output file to draw
a polygon according to the vector of points specified by @p
points
void DrawPolygonLine(const std::vector<std::pair<double, double>
>& points);
```

- **ASCIIPainter**: clase que hereda de OBPainter.

```
public:

//The method is declared empty to avoid compilation errors due to
interface implementation. It has no use
void DrawPolygonLine(const std::vector<std::pair<double, double>
>& points);
```

- **CommandPainter**: clase que hereda de OBPainter.

```
public:

//The method is declared empty to avoid compilation errors due to
interface implementation. It has no use
void DrawPolygonLine(const std::vector<std::pair<double, double>
>& points);
```

¹<https://openbabel.github.io/api/3.0/index.shtml>

- **OBAtom**: clase central, contiene la información relativa a un átomo. Se han añadido los siguientes métodos:

```
public:

//\return Is this a metal commonnly present in organometallic
  compounds?
bool IsOgmMetal();

//\return Is atom part of a Cp ring?
bool IsInCp() const;

//\return Is this atom a Carbon (atomic number == 6)?
bool IsCarbon();

//Debug method. Displays on basic output simple data to identify
  the atom
void Show();

//Mark an atom as part of a Cp ring
void SetInCp(bool value = true);
```

- **OBMol**: clase central, almacena toda la información básica relacionada con una molécula. Se han añadido las siguientes variables y métodos:

```
private:

std::string _smiles;           //!< Input smiles string for
  the molecule
std::vector<CpComplex*> _cps;   //!< Cp information
unsigned int _ncps;           //!< Number of cps complexes
  detected
std::string _canSmiles;        //!< Canonical smiles based
  on Ogm canonicalization
std::vector<BranchBlock*> _blocks; //!< Branches information
unsigned int _nblocks;         //!< Number of blocks

public:

//! Set the input smiles string of this molecule to @p smi
void SetInputSmiles(std::string smi);

//! \return the input smiles string of this molecule
std::string GetSmiles();

//! Add a new CpComplex specified by @p cp
void AddCpComplex(CpComplex& cp);

//! \return the cp at index @p idx or NULL if none exists.
CpComplex* GetCpComplex(int idx);

//! \return number of cp in the molecule
unsigned int GetCpSize();

//! \return whether the molecule has cps or not
bool HasCp();
```

```

    //! \return the whole container of cps of this molecule
    std::vector<CpComplex*> GetCps();

    //! Add a new block to the molecule, specified by @p branch
    BranchBlock* AddBranchBlock(BranchBlock& branch);

    //! \return the number of blocks in the molecule
    unsigned int GetBlockSize();

    //! \return the canonical smiles string generated by the ogm
    canonicalization methods
    std::string GetCanSmiles();

    //! Set the canonical smiles string of this molecule to @p smi
    void SetCanSmiles(std::string smi);

    //! Debug method. Displays on basic output all molecule blocks
    with basic information of the atoms.
    void ShowBranches();

    //! \return If this molecule has any Ogm metal or not
    bool HasOgmMetal();

    //! \return the block of which the carbon at index @p carbon_idx
    is part, or NULL if no such block exists
    BranchBlock* FindBranch(int carbon_idx);

    //! Set the iterator to the beginning of the Cp list
    //! \return the first Cp structure, or NULL if none exist
    CpComplex* BeginCp(std::vector<CpComplex*>::iterator & i);

    //! Advance the iterator to the next Cp record
    //! \return the next first Cp record, or NULL if none exist
    CpComplex* NextCp(std::vector<CpComplex*>::iterator& i);

    //! Set the iterator to the beginning of the BranchBlock list
    //! \return the first BranchBlock structure, or NULL if none
    exist
    BranchBlock* BeginBranchBlock(std::vector<BranchBlock*>::iterator
    & i);

    //! Advance the iterator to the next BranchBlock record
    //! \return the next first BranchBlock record, or NULL if none
    exist
    BranchBlock* NextBranchBlock(std::vector<BranchBlock*>::iterator&
    i);

```

- **OBMol2Cansmi**: clase que maneja la conversión del smiles de entrada a un smiles canónico. Se han añadido los siguientes métodos:

```

    private:

    //Only changed visibility to private, since
    CreateFragCansmiStringOgm was created. Selects the "root"
    atom, which will be first in the SMILES, then builds a tree
    in canonical order, and finally generates the SMILES.
    void CreateFragCansmiString(OBMol&, OBBitVec&, std::string&);

    //Auxiliary private methods for SelectRootAtomOgm

```

```

//Shortened version of the CreateCansmiString method. Create the
    necessary variables to call AuxCreateFragCansmiStringOgm.
void AuxCreateCansmiString(OBMol& mol, OBBitVec& frag_atoms,
    OBConversion* pConv, OBAtom* startatom, std::vector<
    SubTreeSizes*>& subtreesizes, std::vector<OBAtom*> ogmAtoms);

//Shortened version of the CreateFragCansmiStringOgm method.
    Create the necessary variables to build a new canonical tree
    using as root @p startAtom
void AuxCreateFragCansmiStringOgm(OBMol&, OBBitVec&, OBAtom*, std
    ::vector<SubTreeSizes*>&, std::vector<OBAtom*>);

//Once the tree is built, this method runs through it in DFS
    evaluating the subtrees hanging from the other ogm metals.
    Use the auxiliary struct SubTreeSizes for this.
void EvaluateMetalSubTrees(OBCanSmiNode* root, OBCanSmiNode* node
    , std::vector<SubTreeSizes*>&, std::vector<OBAtom*>&, std::
    vector<int>&);

    public:
//Method based on CreateFragCansmiString. Share much of the code,
    with some additional methods specifically for my own
    canonical form designed for organometallic molecules.
void CreateFragCansmiStringOgm(OBMol&, OBBitVec&, std::string&,
    OBConversion*);

//If more than 1 Ogm metal is present in the molecule, this
    method chooses one of them, based on some rules and the
    connectivity of the metal within the molecule and the rest of
    the atoms
OBAtom* SelectRootAtomOgm(OBMol&, OBConversion*);

//Debug method for writing in basic output the tree with
    hierarchy formatting
void WriteTree(OBCanSmiNode* node, int level = 0);

//Adds information to the molecule of the blocks that form it.
    Being a block, each set of atoms that, due to their bonds,
    are within the same parenthesis in the original input Smiles.
    Or, according to the OBMol2Cansmi::BuildCanonTree method,
    the parent-child relationship between atoms.
void IdentifyBranches(OBMol& mol, OBCanSmiNode* node, BranchBlock*
    branch = nullptr);

//Modifies the tree built by BuildCanonTree based on the length
    of the branches identified in IdentifyBranches. This is a
    canonical rule designed for a little more consistency in the
    output canon smiles.
void RearrangeTree(OBCanSmiNode* node);

//Builds the SMILES tree, in canonical order, for the specified
    molecular fragment. Based on the BuildCanonTree method.
    Shares much of the code, with some changes in the neighbour
    selection algorithm.
bool BuildCanonTreeOgm(OBMol& mol, OBBitVec& frag_atoms, vector<
    unsigned int>& canonical_order, OBCanSmiNode* node);

```

- **OBCanSmiNode**: clase que representa un nodo. En conjunto se forma una estructura de árbol, cada nodo es un átomo del árbol para luego escribir el SMILES canónico. Se han añadido los elementos:

```

    private:

    OBCanSmiNode* _parentNode;          //!< Pointer to the parent node

    //!< Add a child bond to the node, specified by @p bond. Should
    only be used in the ResetBonds method as a part of the
    OBmol2Cansmi::RearrangeTree algorithm.
    //!< Otherwise, use addChildnode to add both the child node and
    its respective bonds
    void AddChildBond(OBBond* bond);

    public:

    //!< Set the parent node to @p parent
    void SetParentNode(OBCanSmiNode* parent);

    //!< \return the parent node
    OBCanSmiNode* GetParentNode();

    //!< Traverses the tree in dfs from the node calling the method
    //!< \return the number of total children (counting himself)
    int SubTreeSize();

    //!< Sort a node's child_nodes using a std::sort operation an a
    custom comparator 'mycomp'
    void SortChilds();

    //!< When added at the same time in the addchildnode method, the
    child with its bond have a 1 to 1 index correspondence. When
    reordering the children, in OBmol2Cansmi::RearrangeTree, the
    indices of the bonds are lost. This method clears and adds
    the bonds back in order.
    void ResetBonds();

    //!< \return the total number of carbons in this node subtree
    int nCarbonsSubTree();

```

4.3. Clases nuevas

- **CpComplex**: clase que maneja y permite almacenar estructuras de ciclopentadienilo. Se han creado las siguientes variables y métodos:

```

    protected:

    OBmol* _parent;                      //!< Parent molecule
    unsigned int _idx;                   //!< Cp identifier within
    the molecule
    unsigned int metal_idx;              //!< Atom idx of central
    metal
    std::vector<OBAtom*> _cpAtoms;        //!< Atoms for the
    carbons of the Cp structure
    std::vector<unsigned int> idx_carbons; //!< Atom indexes for the
    carbons of the Cp structure
    vector3 center;                     //!< Cp center, for
    normal bond connection with metal atom, and aromatic circle
    position
    std::vector<vector3> circlePath;      //!< Coordinates for the
    cp circle (needed to achieve a perspective circumference)

```

```

double radius;                                //!< Cp's aromatic circle
    radius
unsigned int dummy_idx;                        //!< Dummy central atom
    idx

    public:

    //!< Default constructor
    CpComplex();

    //!< \name Methods to modify internal information
    //@{
    //!< Attach an OBmol @p ptr as the parent container for this Cp
    void SetParent(OBMol* ptr);
    //!< Set the center point of the Cp, sprecified by @p _v. It is
        equidistant to every carbon in th Cp, as they are disposed in
        a regular polygon
    void SetCentroid(vector3& _v);
    //!< Set the center point of the Cp, sprecified by @p v_x, v_y,
        v_z. It is equidistant to every carbon in th Cp, as they are
        disposed in a regular polygon
    void SetCentroid(const double v_x, const double v_y, const double
        v_z);
    //!< Set the radius of the Cp circle
    void SetRadius(double r);
    //!< Set the Cp identifier
    void SetIdx(int idx);
    //!< Set the idx of the central metal to which this Cp is attached
    void SetMetalIdx(int midx);
    //!< Dummy atom is created to make a perpendicular bond between
        the metal and the Cp drawing
    //!< Set the atom idx of the dummy atom created for this Cp
    void SetDummyIdx(int idx);
    //!< Set the point of the Cp circle at index @p i to the
        coordinates specified by @p _v
    void SetCircleCoord(unsigned int i, vector3 _v);
    //!< Set the point of the Cp circle at index @p i to the
        coordinates specified by @p _vx, _vy, _vz
    void SetCircleCoord(unsigned int i, double _vx, double _vy,
        double _vz = 0.0);
    //@}

    //!< \name Methods to retrieve information
    //@{
    //!< \return number of carbon atoms in the cp
    unsigned int GetCarbonsSize();
    //!< \return the molecule which contains this Cp, or NULL if none
        exists
    OBMol* GetParent();
    //!< \return dummy atom idx for this Cp structure, or 0 if none
        exists
    unsigned int GetDummyIdx() const;
    //!< \return Cp identifier
    unsigned int GetIdx() const;
    //!< \return Central metal identifier
    unsigned int GetMetalIdx() const;
    //!< \return carbon idx at position @p i in tha container. Zero
        based access method to vector
    unsigned int GetCarbonIdx(int i) const;
    //!< \return the whole contanier of carbon idx

```



```

const std::vector<unsigned int>& GetIdxCarbons();
///! \return the centroid of this Cp in a coordinate vector
vector3& GetCentroid();
///! \return the radius of the Cp circle
double GetRadius();
///! \return the coordinate vector for the Cp circle point at
    position @p i in the container. Zero based access method to
    vector
vector3 GetCircleCoord(unsigned int i);
///! \return the number of points of the Cp circle
int GetCirclePathSize() const;
///! \return the whole container of coordinates of the Cp circle
std::vector<vector3> GetCircleCoords() const;
//@}

///! \name Addition of data for a Cp
//@{
///! Adds a new atom idx to this Cp
void AddIdxCarbon(int idx);
///! Adds a new atom to this Cp
void AddCpAtom(OBAtom* atom);
///! Adds a new point to the coordinate vector that forms the Cp
    circle
void AddCircleCoord(vector3 _v);
//@}

///! \name Iteration methods
//@{
///! Set the iterator to the beginning of the Cp atom list
///! \return the first atom, or NULL if none exist
OBAtom* CpComplex::BeginAtomCp(OBAtomIterator& i);
///! Advance the iterator to the next atom in the Cp
///! \return the next first atom record, or NULL if none exist
OBAtom* CpComplex::NextAtomCp(OBAtomIterator& i);
//@}

///! \name Other operations
//@{
///! Calculate and set the centroid of this Cp, taking into
    consideration all atoms stored in _cpAtoms
void FindCentroid();
///! Equivalence operator
bool operator==(const CpComplex* other) const;
//@}

```

- **BranchBlock**: clase que representa un bloque definido dentro de la molécula, p.ej. un ciclo de benceno, un Cp, o toda una rama de un átomo. Se han añadido las siguientes variables y métodos:

```

private:

unsigned int _idx;                                //!< Block identifier
std::vector<unsigned int> vidx_atoms;              //!< Vector idx of
    the atoms that are part of the block.

public:

```

```

///! Default constructor
BranchBlock();

///! Destructor
~BranchBlock();

///! \return the size of the block (number of atoms in the block)
int Size();

///! \return the block identifier
unsigned int GetIdx();

///! Set the block identifier
void SetIdx(int idx);

///! Add an atom's idx to the block
void AddAtom(int i);

///! \return the idx of the atom at position @p i. Zero based
access.
unsigned int GetAtomIdx(int i);

///! \return Whether the @p idx exists within the atoms already
inserted in the block
bool HasAtom(int idx);

///! Cp will be possible if all the elements in the block are
carbons up to that point and have a bond with an ogm metal.
///! \return whether or not it appears to be a Cp block
bool IsPossibleCp(OBMol &mol)

```

- **OpCpDraw**: clase plugin que hereda de OBOp. Contiene el algoritmo de detección, identificación, y almacenamiento en la molécula de estructuras tipo Cp.

```

///! Default constructor
OpCpDraw(const char* ID);

///! Inherited method.
///! Display through the output stream a brief description of the
plugin.
const char* Description();

///! Inherited method.
///! \return true if this op (plugin operation) is designed to
work with the class of @p pObj, e.g. OBMol
virtual bool WorksWith(OBBase* pObj) const;

///! Inherited method. Required function that does the work.
Normally return true, unless object is not to be output.
virtual bool Do(OBBase* pObj, const char* OptionText = nullptr,
OpMap* pOptions = nullptr, OBConversion* pConv = nullptr);

///! \return If @p bond is likely to be a cp-bond like
bool isCpBond(OBBond* bond, unsigned int idxM);

///! Finds the ring of which the carbon with idx @p carbonIdx is a
part of, among the rings of @p rlist (obtained from a SSSR
perspective), and stores it in @p result.

```

```
//! \returns whether it was found or not
bool FindRingWithCarbon(vector<OBRing*>& rlist, int carbonIdx,
    OBRing*& result);

//! Canonize the input SMILES and identify blocks
void CanonizeOgm(OBMol* mol, OBConversion* pConv);
```

- **SubTreeSizes**: struct auxiliar creado para la selección del primer metal durante la canonización (se profundiza sobre esto en la sección ??). Contiene las siguientes variables y métodos:

```
OAtom* _root;           //!< Tree root
OAtom* _metal;          //!< Metal to evaluate
int size;               //!< Size of the subtree for the _metal to
    evaluate

//! Default constructor
SubTreeSizes();

//! Parameter constructor. Creates a new object with _root as @p
    root
SubTreeSizes(OAtom* root);

//! Debug method. Displays through basic output the struct
    information.
void Show();
```

- **subtreecomp**: objeto comparador que prioriza unos metales sobre otros en el proceso de selección del átomo raíz para el árbol (más en detalle en la Sección ??).

```
bool operator() (SubTreeSizes* element1, SubTreeSizes* element2)
    const;
```

- **mycomp**: objeto comparador que prioriza las ramas del árbol canónico durante el proceso de reordenación.

```
bool operator() (OBCanSmiNode* node1, OBCanSmiNode* node2);
```


Capítulo 5

Implementación y resultados

Aquí la idea es ir poniendo las pruebas que vaya haciendo de las moléculas, y lo que vaya descubriendo.

En otro apartado, explicar el sistema de canonización (y los cambios en el dibujado, esto no se si es mejor directamente en resultados, puesto que tendré que poner fotos de cómo ha quedado, y explicarlo sin fotos es medio raro) al que he llegado y sus reglas. Ya en la sección de experimentación, expondré los resultados.

- espaciado de los bonds aumentado para mayor claridad y separación entre los átomos. Hace que no se vea todo tan pegado. Útil para moléculas planas o sin estructuras especiales (cps, o geometrías)

- Se ha puesto el foco /centrado los esfuerzos de mejora de dibujado en estructuras Cp, muy comunes en organometálica. Describir cómo se detectan los Cp (hablar de los problemas con los anillos SSSR, peculiaridades, casos específicos, problemas y proceso de implementación (de pensado del algoritmo o de cómo funciona a rasgos generales sin entrar en tema de código), como 1º trabajé con moléculas con 1 solo Cp, y luego con varias, una vez tuviera los bloques (hablar por tanto de la necesidad de distinguir/detectar no solo las estructuras, sino cuántas había y cuando empieza y acaba cada una, y qué carbonos forman parte de cada Cp)) Ilustrar todo esto con imágenes es bueno.

- Canonizado: comentar no se si aquí o en el estado del arte el sistema de canonizado propio de openbabel. se quería idear un sistema canónico para que independientemente de la forma en la que se escriba el SMILES de la molécula, siempre obtuviéramos el mismo SMILES resultado. Para esto, es necesario definir una serie de reglas que den prioridad a según qué átomos (manteniendo obviamente la correspondencia de los enlaces entre ellos) para luego mostrarlos en ese orden. Describir esas reglas detalladamente y con ejemplos a ser posible (al menos 1 para ilustrar el caso).

Despues de describir el proceso de cada cosa, mostrar sus resultados

En otro apartado describir el testing. La mayoria de tests que he realizado son tests funcionales completos. Hacer test de unidad para los metodos importantes es complicado ya que necesitan una serie de parámetros y variables que se van seteando/creando sobre la marcha en multitud de metodos previos.

Bibliografía

- [1] William J. Wiswesser. “107 Years of Line-Formula Notations (1861-1968)”. en. En: *Journal of Chemical Documentation* 8.3 (ago. de 1968), págs. 146-150. ISSN: 0021-9576, 1541-5732. DOI: 10.1021/c160030a007. URL: <https://pubs.acs.org/doi/abs/10.1021/c160030a007>.
- [2] Reiner Luckenbach. “The Beilstein Handbook of Organic Chemistry: the first hundred years”. en. En: *Journal of Chemical Information and Computer Sciences* 21.2 (mayo de 1981), págs. 82-83. ISSN: 0095-2338. DOI: 10.1021/ci00030a006. URL: <https://pubs.acs.org/doi/abs/10.1021/ci00030a006>.
- [3] David Weininger. “SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules”. en. En: *Journal of Chemical Information and Modeling* 28.1 (feb. de 1988), págs. 31-36. ISSN: 1549-9596. DOI: 10.1021/ci00057a005. URL: <https://pubs.acs.org/doi/abs/10.1021/ci00057a005> (visitado 20-11-2022).
- [4] Johann Gasteiger y Thomas Engel. *Chemoinformatics: A Textbook*. en. John Wiley & Sons, dic. de 2006. ISBN: 978-3-527-60650-4.
- [5] Mike Cohn. *Differences Between Scrum and Extreme Programming*. en. Abr. de 2007. URL: <https://www.mountaingoatsoftware.com/blog/differences-between-scrum-and-extreme-programming> (visitado 14-05-2023).
- [6] Andrew R. Leach y V. J. Gillet. *An Introduction to Chemoinformatics*. en. Google-Books-ID: 4z7Q87HgBdwC. Springer, sep. de 2007. ISBN: 978-1-4020-6291-9.
- [7] Johann Gasteiger. “The Scope of Chemoinformatics”. en. En: *Handbook of Chemoinformatics*. Ed. por Johann Gasteiger. Weinheim, Germany: Wiley-VCH Verlag GmbH, mayo de 2008, págs. 3-5. ISBN: 978-3-527-61827-9 978-3-527-30680-0. DOI: 10.1002/9783527618279.ch1. URL: <https://onlinelibrary.wiley.com/doi/10.1002/9783527618279.ch1>.

- [8] Nathan Brown. “Chemoinformatics—an introduction for computer scientists”. en. En: *ACM Computing Surveys* 41.2 (feb. de 2009), págs. 1-38. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/1459352.1459353. URL: <https://dl.acm.org/doi/10.1145/1459352.1459353>.
- [9] M. Despa. “Comparative study on software development methodologies”. En: *Database Systems Journal* (dic. de 2014). URL: <https://www.semanticscholar.org/paper/Comparative-study-on-software-development-Despa/28d32b9375e9125624ec614a2cec85c9f0716b13>.
- [10] Stephen R. Heller et al. “InChI, the IUPAC International Chemical Identifier”. En: *Journal of Cheminformatics* 7.1 (mayo de 2015), pág. 23. ISSN: 1758-2946. DOI: 10.1186/s13321-015-0068-4. URL: <https://doi.org/10.1186/s13321-015-0068-4>.
- [11] Thomas Engel y Johann Gasteiger. *Applied Chemoinformatics. Achievements and Future Opportunities*. en. Wiley-VCH, 2018. ISBN: 978-3-527-34201-3.
- [12] Juan Camilo Salazar et al. “Scrum versus XP: similitudes y diferencias”. es. En: *Tecnología Investigación y Academia* 6.2 (dic. de 2018), págs. 29-37. ISSN: 2344-8288. URL: <https://revistas.udistrital.edu.co/index.php/tia/article/view/10496>.
- [13] Flaviu Fuior. “Key elements for the success of the most popular Agile methods”. En: *Revista Română de Informatică și Automatică* 29 (dic. de 2019), págs. 7-16. DOI: 10.33436/v29i4y201901.
- [14] Mario Krenn et al. “Self-referencing embedded strings (SELFIES): A 100% robust molecular string representation”. en. En: *Machine Learning: Science and Technology* 1.4 (dic. de 2020), pág. 045024. ISSN: 2632-2153. DOI: 10.1088/2632-2153/aba947. URL: <https://iopscience.iop.org/article/10.1088/2632-2153/aba947>.
- [15] Digital.ai. *15th State of Agile Report*. 2021.
- [16] Alok Mishra et al. “Organizational issues in embracing Agile methods: an empirical assessment”. En: *International Journal of System Assurance Engineering and Management* 12 (oct. de 2021). DOI: 10.1007/s13198-021-01350-1.
- [17] Kevin Darza. *¿Cómo calcular la depreciación de un equipo de cómputo?* Ago. de 2022. URL: <https://www.oliversoft.mx/depreciacion-de-un-equipo-de-computo/> (visitado 25-05-2023).
- [18] Mario Krenn et al. “SELFIES and the future of molecular string representations”. en. En: *Patterns* 3.10 (oct. de 2022). arXiv:2204.00056 [physics], pág. 100588. ISSN: 26663899. DOI: 10.1016/j.patter.2022.100588. URL: <http://arxiv.org/abs/2204.00056> (visitado 20-11-2022).

- [19] Alston Lo et al. *Recent advances in the Self-Referencing Embedding Strings (SELFIES) library*. arXiv:2302.03620 [physics]. Feb. de 2023. DOI: 10.48550/arXiv.2302.03620. URL: <http://arxiv.org/abs/2302.03620> (visitado 26-02-2023).
- [20] *Extreme Programming: A Gentle Introduction*. URL: <http://www.extremeprogramming.org/> (visitado 07-05-2023).
- [21] ICIQ. Prof. Mónica H. Pérez-Temprano, Research Group. en. URL: https://www.iciq.org/research/research_group/dr-monica-h-perez-temprano/section/research_overview/ (visitado 10-03-2023).
- [22] PubChem. *PubChem*. en. URL: <https://pubchem.ncbi.nlm.nih.gov> (visitado 01-02-2023).
- [23] SciFinder. *SciFinder-help*. en. URL: https://scifinder-n.cas.org/help/#t=Working_with_Search_Results%5C%2FAll_Answer_Types_screen.htm (visitado 01-02-2023).
- [24] *Scrum and Extreme Programming (XP) - a perfect hybrid model for software development projects*. es. URL: <https://www.linkedin.com/pulse/scrum-extreme-programming-xp-perfect-hybrid-model-raghavan> (visitado 08-05-2023).
- [25] SigmaAldrich. *SigmaAldrich - About us*. es. URL: <https://www.sigmaaldrich.com/ES/es/life-science/about-us/expertise> (visitado 01-02-2023).