



ESCUELA DE INGENIERÍA DE FUENLABRADA

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE
TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

EVOLUCIÓN DE LA PARTICIPACIÓN VOLUNTARIA DE
PROYECTOS DE SOFTWARE LIBRE: EVIDENCIAS DE
DEBIAN

Autor : Pablo Cabeza Portalo

Tutor : Dr. Gregorio Robles

Curso académico 2023/2024

Trabajo Fin de Grado

Reproducción de un trabajo con evidencias de Debian

Autor : Pablo Cabeza Portalo

Tutor : Dr. Gregorio Robles

La defensa del presente Proyecto Fin de Carrera se realizó el día de
de 2024, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2024

*Dedicado a
mi familia, mis amigos y a mi pareja.*

Agradecimientos

Este es el fin de una de las etapas más importantes de mi vida y por ello quiero agradecer a varias personas que me han acompañado en este proceso. Primero quiero agradecer a mi madre Antonia todo el apoyo que me ha dado en cada una de mis decisiones, tanto personales como académicas. Eres quien me impulsa a lograr mis sueños. Segundo a mi padre Manuel que siempre ha sido mi ejemplo a seguir. Me ha aportado valores fundamentales como el trabajo y el esfuerzo que han sido imprescindibles para abordar esta carrera. Gracias a los dos por darme la vida. También agradecer a mi tutor, Gregorio, por darme este proyecto, tener paciencia y ayudarme en todo cuanto pudo para desarrollarlo. Por último, quiero agradecer a esa persona tan especial que conocí en esta universidad y que me dio la confianza y la fuerza necesaria para lograr mis metas. Paula, gracias por ser mi compañera de viaje y por aparecer en el momento que mas lo necesitaba. Hemos afrontado este desafío juntos y sin ti no hubiera sido igual.

Resumen

Este proyecto se basa en un estudio sobre la evolución de la distribución más importante de Linux llamada Debian, la cual es de software libre. Partimos de un estudio anterior llamado Evolution of Volunteer Participation in Libre Software Projects: Evidence from Debian [10] del cual se ha realizado una réplica veinte años después.

El objetivo principal es responder a una serie de preguntas planteadas en dicho estudio las cuales nos darán la información necesaria para comprender como evoluciona Debian y si cumplen las tendencias estudiadas en el artículo anterior. Dichas preguntas son: **¿Cuántos mantenedores tiene Debian y cómo cambia este número con el tiempo?, ¿Existe una tendencia hacia la formación de equipos de mantenedores?, ¿Cuántos mantenedores de versiones anteriores permanecen activos?, ¿Cuál es el aporte de los mantenedores que permanecen en versiones posteriores?, ¿Qué sucede con los paquetes mantenidos por los mantenedores que abandonan el proyecto? y ¿Los paquetes más importantes y de uso común son mantenidos por mantenedores más experimentados?.**

Para obtener las conclusiones deseadas se ha pasado por las etapas de **descarga de releases de Debian, análisis de los paquetes de los releases, creación de diagrama entidad-relación, creación de bases de datos junto con sus tablas SQL, parseo de la información de cada paquete, obtención de gráficos sobre la información, creación de Queries e inserción de la información en la base de datos.**

Para ello se usan tecnologías tales como **Pycharm y MySQL Workbench**. En ellas necesitamos el uso de lenguajes de programación tales como **Python y SQL** junto con librerías de Python como **MYSQL Connector o Matplotlib**.

Este proyecto se ha realizado como parte de la asignatura Servicios y Aplicaciones Telemáticas bajo la supervisión de mi tutor el Dr. Gregorio Robles.

Summary

This project is based on a study about the evolution of the most important Linux distribution called Debian, which is free software. We started from a previous study called *Evolution of Volunteer Participation in Libre Software Projects: Evidence from Debian*-[10], which has been replicated twenty years later.

The main objective is to answer the questions raised in that study, which will provide us with the necessary information to understand how Debian evolves and if it follows the trends from the previous study. These questions are: **How many maintainers does Debian have and how does this number change over time?, Is there a pattern of creating maintainer teams?, How many maintainers from previous versions remain active?, What is the contribution of maintainers who stay in later versions?, What happens to the packages maintained by maintainers who leave the project?, Are the most important and commonly used packages maintained by more experienced maintainers?**

To get the results we want, we have gone through the stages of **downloading Debian releases, analyzing the packages of the releases, creating an entity-relationship diagram, creating databases and their SQL tables, parsing the information of each package, obtaining graphs on the information, creating queries, and inserting the information into the database.**

For this purpose, technologies such as **PyCharm and MySQL Workbench** are used. In them, we use programming languages like **Python and SQL** and Python libraries like **MySQL Connector and Matplotlib**.

This project was done for the Telematic Services and Applications subject under my tutor Dr. Gregorio Robles.

Índice general

1. Introducción	1
1.1. Contexto	2
1.1.1. Proyecto Debian	2
1.1.2. Versiones Debian	2
1.1.3. Evolución de la Participación Voluntaria en Proyectos de Software Libre: Evidencia de Debian	4
1.2. Estructura de la memoria	6
2. Objetivos	9
2.1. Objetivo general	9
2.2. Objetivos específicos	9
2.3. Planificación temporal	10
3. Estado del arte	13
3.1. Python 3.12.0	13
3.2. MySQL Connector	14
3.3. Matplotlib	15
3.4. Pycharm	15
3.5. SQL	16
3.6. MySQL WorkBench	17
4. Diseño e implementación	19
4.1. Arquitectura general	19
4.2. Descarga de los releases de Debian	20
4.3. Análisis de los paquetes de cada release	20

4.4.	Creación diagrama Entidad - Relación	25
4.5.	Creación de BBDD junto con tablas SQL	28
4.6.	Parseo de la información de los paquetes	34
4.7.	Inserción de la información en la BBDD	36
4.8.	Creación de Queries SQL	39
4.9.	Obtención de gráficas informativas	41
5.	Resultados	45
5.1.	¿Cuántos mantenedores tiene Debian y cómo cambia este número con el tiempo?	45
5.2.	¿Existe una tendencia hacia la formación de equipos de mantenedores?	48
5.3.	¿Cuántos mantenedores de versiones anteriores permanecen activos?	51
5.4.	¿Cuál es el aporte de los mantenedores que permanecen en versiones posteriores?	53
5.5.	¿Qué sucede con los paquetes mantenidos por los mantenedores que abandonan el proyecto?	55
5.6.	¿Los paquetes más importantes y de uso común son mantenidos por mantenedores más experimentados?	57
6.	Conclusiones	63
6.1.	Conclusiones a las cuestiones planteadas	63
6.1.1.	Conclusiones: ¿Cuántos mantenedores tiene Debian y cómo cambia este número con el tiempo?	63
6.1.2.	Conclusiones: ¿Existe una tendencia hacia la formación de equipos de mantenedores?	64
6.1.3.	Conclusiones: ¿Cuántos mantenedores de versiones anteriores permanecen activos?	65
6.1.4.	Conclusiones: ¿Cuál es el aporte de los mantenedores que permanecen en versiones posteriores?	65
6.1.5.	Conclusiones: ¿Cuál es el aporte de los mantenedores que permanecen en versiones posteriores?	66
6.1.6.	Conclusiones: ¿Los paquetes más importantes y de uso común son mantenidos por mantenedores más experimentados?	66
6.2.	Consecución de objetivos	67

<i>ÍNDICE GENERAL</i>	XI
6.3. Aplicación de lo aprendido	68
6.4. Lecciones aprendidas	69
6.5. Trabajos futuros	69
Bibliografía	71

Índice de figuras

2.1. Diagrama de Gantt	11
3.1. Lenguajes de programación más usados - Fuente: Guadalupe Moreno.	14
3.2. Ejemplos gráficas Matplotlib	15
3.3. Pycharm	16
3.4. SQL Workbench. Fuente: elaboración propia.	17
4.1. Fases del tratamiento de datos	19
4.2. Diagrama de Flujo de la descarga de los Releases	21
4.3. Descarga de los datos de releases	22
4.4. Atributos de un paquete.	22
4.5. Flujo del estudio de los atributos de un paquete	23
4.6. Diagrama ER.	26
4.7. Leyenda del Diagrama ER.	27
4.8. Flujo de creación de diagrama Entidad - Relación	29
4.9. Flujo de creación de BBDD	30
4.10. Bases de datos	31
4.11. Tablas BBDD	33
4.12. Flujo de inserción de los datos en BBDD	37
4.13. Build Depends	38
4.14. Atributo Files de cada paquete en el que se muestran diferentes archivos contenidos en él.	39
4.15. Flujo de creación de queries	40
4.16. Flujo de creación de gráficas	43

5.1. Número de maintainers de cada release	47
5.2. Número de paquetes de cada release	47
5.3. Ejemplo de nombre de equipo de maintainers	48
5.4. Ejemplo de nombre de equipo de maintainers	49
5.5. Equipos formados en cada release	49

Capítulo 1

Introducción

Vivimos en un mundo totalmente digitalizado en el que la presencia de ordenadores está a la orden del día. Con esto no nos referimos únicamente a ordenadores de escritorio. A diario interactuamos con una amplia gama de dispositivos compuestos por ordenadores camuflados. Desde electrodomésticos inteligentes pasando por coches modernos hasta en tarjetas de crédito. Estos tienen incorporados ordenadores pequeños pero potentes que realizan una serie de tareas para el beneficio y la mejora de la vida humana.

En la mayoría de estos ordenadores de uso cotidiano, como portátiles o móviles, se aloja un Sistema Operativo. Un Sistema Operativo es el “intermediario” entre el usuario y el hardware del ordenador a partir de software. Gestiona los recursos del hardware proporcionando una interfaz al usuario. De esta forma pueden interactuar con dicho ordenador. Algunos de los sistemas operativos más usados son: iOS, Android, macOS, Microsoft Windows o Linux.

Linux es un sistema operativo de código abierto el cual es gratuito para cualquier usuario que quiera adoptarlo en su computadora. Este consta de muchas distribuciones. Las distribuciones son versiones del Sistema Operativo de Linux desarrolladas por diferentes individuos, equipos o empresas para mejorar la experiencia de los usuarios.

Una de estas distribuciones es “Debian” y es sobre la que tratará este estudio. Debian es un Sistema Operativo que trabaja con el Kernel (núcleo) de Linux y ha ido aportando distintas versiones desde 1993. Treinta años después nos preguntamos ciertas cosas como, ¿Los individuos que trabajaban en las primeras versiones siguen actualizando Debian? ¿Se trabaja individualmente o por equipos? ¿Cuántos paquetes sacan en cada versión? ¿Qué ocurre con ellos? Todo esto lo veremos a continuación.

1.1. Contexto

1.1.1. Proyecto Debian

El Proyecto Debian está formado por un grupo de voluntarios a nivel mundial que trabajan para producir una distribución del Sistema Operativo Linux basada en ‘software libre’.

Con el término ‘software libre’ no nos referimos a su coste. Este va enfocado a la ‘libertad real’ dentro del software, es decir, ‘software de código abierto’. Esto significa que cualquier usuario puede acceder al código fuente para estudiarlo, revisarlo, modificarlo o distribuirlo sin restricción alguna.

Debian es la distribución de Linux más relevante sin fines comerciales. En su comienzo, fue la única abierta a la participación de diferentes usuarios que quisieran aportar al proyecto con su trabajo.

Con el tiempo fue asentando un gran conjunto de directrices y procedimientos para el empaquetamiento y distribución de software. Esto les sirvió para poder alcanzar los estándares de calidad requeridos y con ello asegurar su buen funcionamiento.

1.1.2. Versiones Debian

Debian ha publicado varias versiones desde 1993 las cuales explicaremos a continuación:

- **Versiones 0.x (1993 - 1995):** estas versiones fueron las primeras y más rudimentarias pero dieron lugar a la creación de Debian gracias a su creador **Ian Murdock**.
 - **Debian 0.01 hasta 0.90.**
 - **Debian 0.91:** disponía de un sencillo sistema de empaquetamiento que permitía instalar y desinstalar paquetes.
 - **Debian 0.93R5:** se asignaron responsabilidades de cada paquete a cada uno de los desarrolladores. Se comenzó a usar el administrador de paquetes **dpkg** para la instalación de paquetes después de la instalación del sistema. base.
- **Versiones 1.x (1996 - 1997):** **Bruce Perens** fue designado como líder del proyecto después de que Ian lo designara.

- **Debian 1.0:** esta versión nunca fue publicada debido a una confusión al distribuir una versión en desarrollo con el nombre equivocado de Debian 1.0 que daría problemas en ejecución.
- **Debian 1.1 Buzz:** es la primera versión de Debian con un nombre en clave sacado de las películas de **‘Toy Story’**.
- **Debian 1.2 Rex:** esta versión estaba completamente en formato **ELF** y usaba el núcleo (kernel) Linux 2.0.

El formato **ELF** (Executable and Linkable Format) es un estándar. Se usa en sistemas operativos tipo **UNIX** (como Linux). Sirve para organizar y manejar archivos ejecutables, bibliotecas compartidas y otros objetos binarios.

- **Debian 1.3 Bo.**
- **Versiones 2.x (1998 - 2000): Ian Jackson** pasó a ser el líder del proyecto.
 - **Debian 2.0 Hamm:** fue la primera versión multiplataforma de Debian. Agregó soporte para arquitecturas de la serie **Motorola 68000**.
 - **Debian 2.2 Potato:** agregó soporte para las arquitecturas PowerPC y ARM (CPU's de arquitectura RISC creadas por diferentes empresas).
- **Versiones 3.x (2002 -2005):**
 - **Debian 3.0 Woody:** se agregaron más arquitecturas a esta versión y fue la primera en usar **software criptográfico**. Este se usa para codificar información y mantener la transferencia segura de datos.
 - **Debian 3.1 Sarge:** incluye un nuevo instalador llamado **debian-installer**. Contiene detección automática de hardware, instalación sin supervisión y está traducido a más de treinta idiomas.
- **Debian 4.0 Etch (2007):** se añadieron mejoras como un instalador gráfico o la verificación criptográfica de los paquetes descargados entre otras.
- **Debian 5.0 Lenny (2009):** añadió la arquitectura **ARM EABI** para dar soporte a los nuevos procesadores **ARM**.

- **Debian 6.0 Squeeze:** con esta versión fue la primera vez que una distribución de Linux se extendía para permitir también el uso de un núcleo no Linux.
- **Debian 7.0 Wheezy (2011):** se introdujo el soporte de **multiarquitectura**. Esto permitía que los usuarios instalaran en una misma máquina paquetes de múltiples arquitecturas.
- **Debian 8 Jessie (2013):** trajo importantes mejoras de seguridad, como un nuevo kernel que solucionaba varias vulnerabilidades (como **ataques de enlace simbólico**).
- **Debian 9 Stretch (2015):** se introdujeron paquetes para la depuración a través de un repositorio nuevo en el archivo. Facilitaría el proceso de depuración y solución de problemas relacionados con esos paquetes.
- **Debian 10 Buster (2019):** incluyó por primera vez un marco de control de acceso obligatorio. Restringe las acciones que pueden realizar los programas, limitando su acceso a ciertos recursos del sistema, como archivos, directorios, redes, etc.
- **Debian 11 Bullseye (2021):** introduce un nuevo paquete, `ipp-usb`, que utiliza el protocolo IPP-over-USB, independiente del fabricante y soportado por muchas impresoras actuales. Esto permite que un dispositivo USB sea tratado como un dispositivo de red.

1.1.3. Evolución de la Participación Voluntaria en Proyectos de Software Libre: Evidencia de Debian

Este proyecto es la réplica del artículo **Evolución de la Participación Voluntaria en Proyectos de Software Libre: Evidencia de Debian** [10].

En este artículo se analiza la evolución en el tiempo de los recursos humanos de uno de los proyectos de software libre más grandes y complejos compuesto principalmente por voluntarios, el proyecto Debian.

Se realiza una investigación cuantitativa de datos de casi siete años, estudiando cómo la participación de voluntarios ha afectado al software lanzado por el proyecto y a la propia comunidad de desarrolladores.

Los resultados que se obtuvieron fueron los siguientes:

Date	Release	Nº Maintainer	Packages	Packages/Maintainer
July-98	2.0	217	1,101	5.1
March-99	2.1	297	1,559	5.2
August-00	2.2	453	2,601	5.7
July-02	3.0	859	5,119	6.0
December-04	(3.1)	1,237	7,786	6.3

Cuadro 1.1: N° de maintainers, paquetes y paquetes por maintainer

- **Número de maintainers de cada versión:** se llegó a la conclusión de que aumenta el número de maintainers con el tiempo. También aumenta el número de paquetes por maintainer por lo que se supuso una mayor eficiencia de los maintainers debido a mejoras en las herramientas.
- **¿Existe una tendencia hacia la formación de equipos de mantenedores?:** los resultados que se obtuvieron marcan una clara tendencia a la formación de los mismos.

Date	Release	Group Packages	Quality Control Packages	Percentage
July-98	2.0	14	14	1.3 %
March-99	2.1	21	11	1.4 %
August-00	2.2	46	31	1.8 %
July-02	3.0	101	71	2.2 %
December-04	(3.1)	599	194	7.4 %

Cuadro 1.2: N° de grupos, paquetes de los grupos y porcentaje de paquetes asociados a grupos

- **Seguimiento de los mantenedores restantes de Debian:** se estudió el número de maintainers que había en la primera versión y como disminuye con el paso del tiempo debido a que abandonan el proyecto.
- **Investigar la experiencia del mantenedor:** se estudió si los mantenedores más experimentados tenían asociados mayor cantidad de paquetes pero resultó ser una proporción igualada. Aunque se debería estudiar más a fondo.
- **Paquetes de mantenedores que abandonaron el proyecto:** se observó que se asignaban los paquetes a otro maintainer cuando abandonaba el proyecto y tenía paquetes asociados.

Date	Release	Maintainers 1° version	Packages	Packages/Maintainer
July-98	2.0	216	1,101	5.1
March-99	2.1	207	1,086	5.2
August-00	2.2	188	1,040	5.5
July-02	3.0	147	870	5.9
December-04	(3.1)	121	729	6

Cuadro 1.3: N° de maintainers de la 1° versión, paquetes y paquetes por maintainer

- **Experiencia e importancia:** se llegó a la conclusión de que los paquetes con mayor número de instalaciones y más usados se asignaban a maintainers más experimentados.

Este artículo se publicó en 2005. A día de hoy, se han publicado ocho versiones nuevas de Debian y por ello podemos realizar un estudio actualizado de dicha distribución. Con la aportación de nuevos datos que ayuden a manejar una mayor exactitud en las conclusiones y a plantear nuevas cuestiones sobre Debian.

1.2. Estructura de la memoria

A continuación se exponen los capítulos en los que se organiza esta memoria y los puntos clave tratados en cada uno de ellos:

- **Capítulo 1: Introducción.** Se explica el contexto de Debian y las diferentes versiones distribuidas a lo largo de su historia.
- **Capítulo 2: Objetivos.** Se especifica cuales son los objetivos parciales para lograr el objetivo general.
- **Capítulo 3: Estado del arte.** Se muestran y explican las diferentes tecnologías usadas para la realización de dicho proyecto.
- **Capítulo 4: Diseño e implementación.** Se muestran las diferentes etapas que se han seguido en este proyecto a detalle.
- **Capítulo 5: Experimentos y validación.** Se indica el proceso seguido para alcanzar los diferentes objetivos usando diferentes tecnologías.

- **Capítulo 6: Resultados.** Se muestran los diferentes resultados de estos experimentos. También se comentan los diferentes patrones o tendencias procedentes del análisis de dichos resultados.
- **Capítulo 7: Conclusiones.** Se observan los resultados obtenidos y se comparan con lo que se esperaba obtener. Se realizan una serie de deducciones tras el tratamiento y el análisis de los datos. Se aplican los conocimientos adquiridos en la carrera y se plantean nuevas líneas de investigación.

Capítulo 2

Objetivos

2.1. Objetivo general

Este proyecto de fin de grado se basa en el análisis de los lanzamientos, paquetes y mantenedores presentes en la evolución de la distribución Debian a lo largo de su historia.

Se busca conocer, para cada versión (**release**), la evolución en el número de personas y equipos que lo mantienen, cuántos de ellos siguen en releases posteriores y qué ocurre con los paquetes de un mantenedor si este abandona el proyecto. Se ha podido llevar a cabo gracias al estudio de diferentes paquetes de datos aportados por Debian¹ en su página oficial. De esta forma se obtienen las diferencias y similitudes necesarias para la comparación de los releases de forma que se pueda comprender su evolución.

2.2. Objetivos específicos

Para poder llevar a cabo dicho objetivo se requiere:

- **Diseñar el diagrama entidad relación.** Se comprenden los diferentes campos que conforman un paquete y se crea un diagrama para el posterior diseño de la base de datos que los alojará. .
- **Crear la base de datos.** Se crea una base de datos con un buen diseño (basada en el diagrama anterior) para poder lanzar **Queries** con las que se obtienen la información que

¹<https://www.debian.org/releases/>

se requiere.

- **Inserción de los datos.** Se crea un script para parsear los datos de los paquetes e insertarlos en sus respectivas tablas junto a sus relaciones con otras tablas..
- **Formular las Queries.** Se analiza la información que se quiere extraer de la base de datos y se crean las llamadas necesarias para obtenerla.
- **Crear las tablas y gráficas.** Con la información obtenida se crean diferentes tablas y gráficas para ayudar a la comprensión de los datos de forma visual.

2.3. Planificación temporal

En el Diagrama de la Figura 2.1 se visualizan las diferentes tareas realizadas junto con la organización en días de las mismas.

Primero se marcaron los objetivos junto con las tecnologías a usar en este proyecto. Más tarde se realizó un análisis de los diferentes paquetes para poder crear el diagrama, el diseño de la base de datos y sus tablas correspondientes. Realizamos el parseo de las diferentes datos del paquete para poder insertarlos correctamente en la base de datos. La inserción de los datos fue lo más problemático en este proyecto. Cada release consta de muchos paquetes en los cuales hay que parsear, extraer e insertar todos los datos. Este estudio se realiza sobre 11 releases. Al tratarse de tanto volumen de datos, el tiempo de ejecución fue elevado. Al comprobar las bases de datos y hallar errores, se insertaban de nuevo los datos por lo que esto fue lo más complejo del proyecto.

Finalmente se realizaron una serie de llamadas SQL (Queries) con las que extraer la información necesaria para el proyecto junto con sus gráficas para obtener dicha información de forma visual.

Con ello pudimos obtener conclusiones sobre el estudio de estos datos y comenzar con la redacción de la memoria.

El Dr. Gregorio Robles, mi tutor, me propuso dicho proyecto en noviembre de 2023 y el tiempo desempeñado en él ha sido de 4h diarias reflejadas en los días laborales, de lunes a viernes. En los meses de abril y mayo se intensifico mi desempeño a 6h diarias para desarrollar la memoria con la mayor dedicación posible.

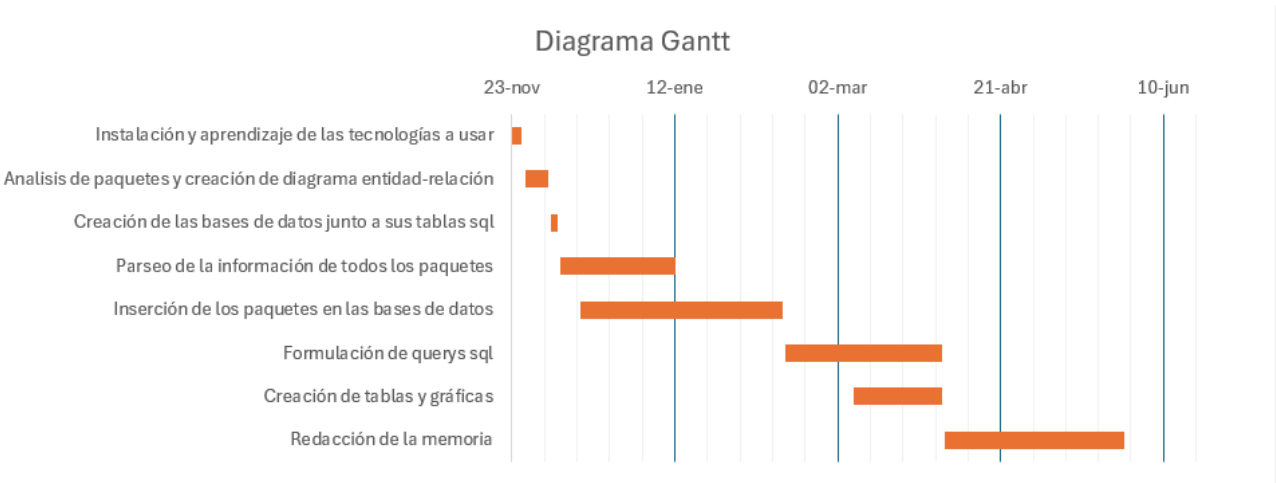


Figura 2.1: Diagrama de Gantt

Capítulo 3

Estado del arte

En este capítulo mencionamos las diferentes tecnologías usadas en este proyecto sin las que no hubiera sido posible su realización.

3.1. Python 3.12.0

Python [4] es un lenguaje de programación orientado a objetos de alto nivel con una sintaxis fácil de interpretar y leer. Tiene un amplio uso en computación científica, desarrollo web y automatización.

Peter Norvig, director de investigación de Google afirma que ‘Python ha sido una parte importante de Google desde el principio, y permanece así a medida que el sistema crece y evoluciona’.

Al ser un lenguaje popular tiene una mayor selección de bibliotecas, lo que ahorra a un desarrollador cantidades increíbles de tiempo y esfuerzo. También tiene más tutoriales y documentación. Esto aumenta las probabilidades de encontrar soluciones a los problemas.

Una curiosidad del lenguaje es que la empresa de mayor emprendimiento de la ‘inteligencia artificial’ **Open AI**¹ está diseñada con python en gran parte y sus bibliotecas son públicas para su uso.

En la figura 3.1 podemos observar que Python es el lenguaje más usado del mundo según sus últimos datos en 2019 [13].

¹<https://platform.openai.com/docs/libraries/python-library>



Figura 3.1: Lenguajes de programación más usados - Fuente: Guadalupe Moreno.

3.2. MySQL Connector

Esta es la librería más importante de Python para poder desarrollar este proyecto. MySQL Connector/Python [14] permite a los programas de Python acceder a las bases de datos MySQL, utilizando una API que cumple con la Especificación de API de Base de Datos de Python. Sus funciones más destacadas e importantes son:

1. **Conexión a la base de datos:** con la función `mysql.connector.connect()`.
2. **Ejecución de consultas SQL:** con la función `cursor.execute()`.
3. **Recuperación de resultados:** con la función `cursor.fetchall()`.
4. **Inserción, actualización y eliminación de datos:** con consultas SQL como INSERT, UPDATE y DELETE, ejecutadas con la función `cursor.execute()`.
5. **Gestión de errores:** se capturan y manejan errores en el código Python usando excepciones.
6. **Desconexión de la base de datos:** con la función `connection.close()`.

3.3. Matplotlib

Matplotlib [5] es una librería de **Python open source**. John Hunter, neurobiólogo, fue su desarrollador inicial en 2002. Su objetivo era visualizar las señales eléctricas del cerebro de personas epilépticas. Por ello intentó replicar las diferentes funcionalidades de MATLAB (gráficas) con Python.

Matplotlib ha sido mejorado a lo largo del tiempo por numerosos contribuidores de la comunidad open source. Se usa para crear gráficas y diagramas de gran calidad que aportan la visualización de los datos de forma detallada.

Es posible crear trazados, histogramas, diagramas de barras y cualquier tipo de gráfica como en la Figura 3.2 con unas líneas de código.

Esta librería es particularmente útil para las personas que trabajan con Python o NumPy.

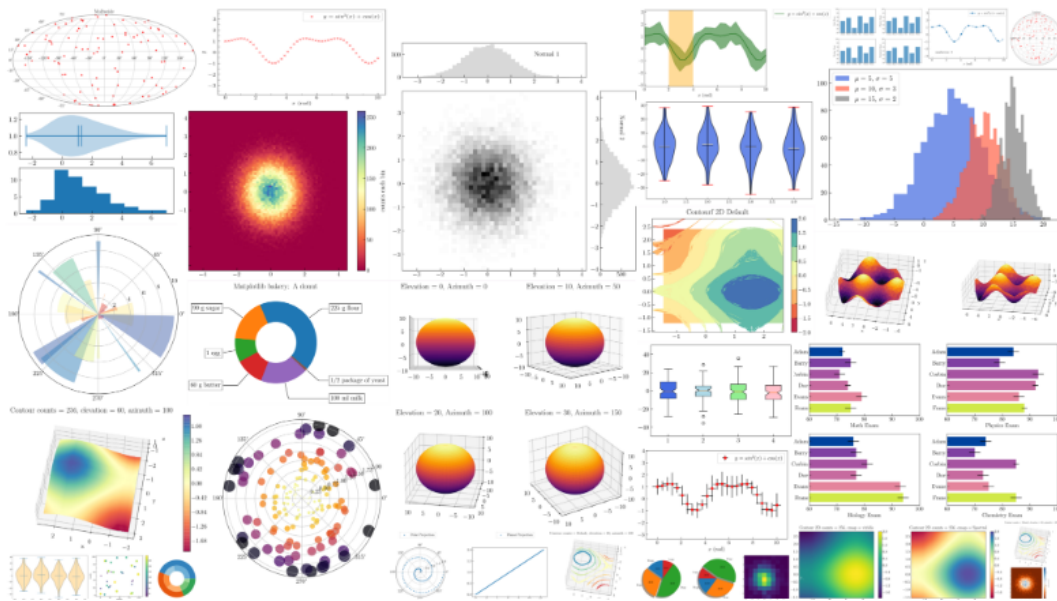


Figura 3.2: Ejemplos gráficas Matplotlib

3.4. Pycharm

PyCharm es el **IDE** [6] más popular para Python hasta la fecha. Esta plataforma híbrida se utiliza habitualmente para el desarrollo de aplicaciones en Python por grandes empresas como Twitter, Facebook, Amazon y Pinterest.

Un **Integrated Development environment (IDE)** o **Entorno de Desarrollo Integrado (EDI)** es un conjunto de herramientas necesarias para desarrollar software. Incluye un editor y un compilador.

Pycharm es compatible con Windows, Linux y macOS. Además contiene módulos y paquetes que ayudan a los desarrolladores a programar software con Python más rápido y con menos esfuerzo y se puede personalizar para responder a las necesidades específicas de un proyecto.



Figura 3.3: Pycharm

3.5. SQL

Es un lenguaje de consulta de bases de datos que **almacena y procesa información en una base de datos relacional**.

Una base de datos **relacional** almacena información en forma de tabla, con filas y columnas que representan diferentes atributos de datos junto con sus relaciones. Con ello se puede **almacenar, actualizar, eliminar, buscar y recuperar** información de la base de datos.

Es un lenguaje de consulta popular que se usa con frecuencia en todos los tipos de aplicaciones debido a su alta integración con el resto de lenguajes tales como **Java, Python, C#, PHP, etc.**

3.6. MySQL WorkBench

Es una herramienta visual ideal para **modelar, diseñar y administrar bases de datos MySQL** junto con el uso de código MySQL.

Se trata de una herramienta gráfica que fue creada por la compañía Oracle. Es un programa de cliente que, a través de un entorno de desarrollo integrado, facilita la creación, consulta y administración de bases de datos [3].

Este software tiene múltiples funcionalidades como:

- **Modelado de datos:** diseñar, modelar, gestionar y generar bases de datos de forma visual.
- **Ingeniería inversa:** recopilar información o datos a partir de un producto determinado para saber qué elementos lo componen.
- **Migrar bases de datos:** migrar desde Microsoft SQL Server, Microsoft Access, Sybase ASE y otros sistemas de gestión de bases de datos a MySQL.

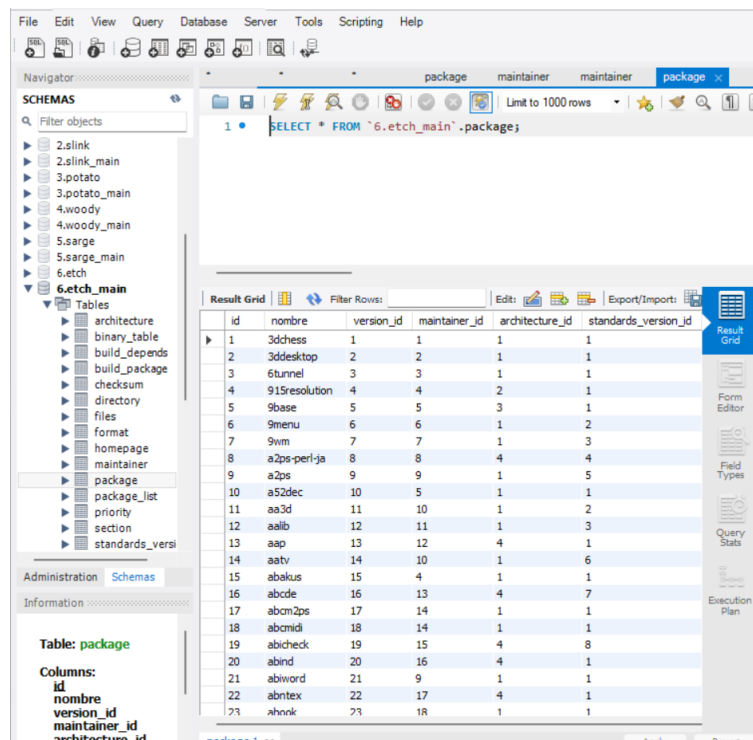


Figura 3.4: SQL Workbench. Fuente: elaboración propia.

Capítulo 4

Diseño e implementación

4.1. Arquitectura general

El objetivo de este proyecto es descubrir e investigar como evoluciona la distribución de Debian a lo largo de su historia y los factores que han cambiado o se han mantenido en releases posteriores junto con su explicación. Para ello hemos seguido estas fases:

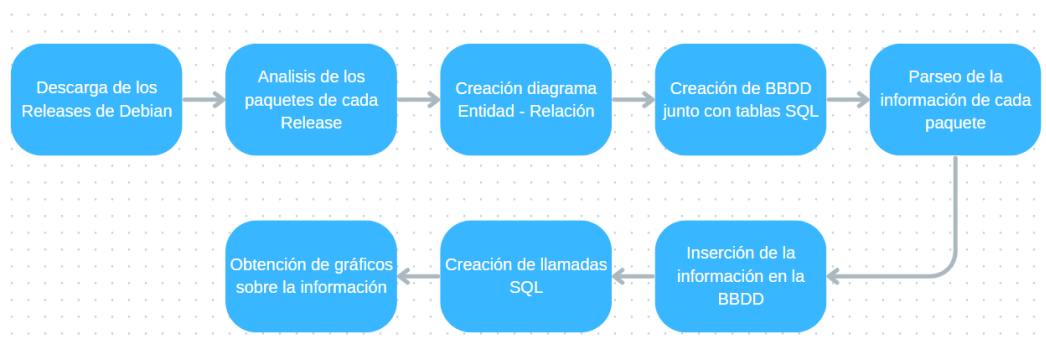


Figura 4.1: Fases del tratamiento de datos

Debemos analizar una gran cantidad de datos. Debian consta de 11 releases (lanzamientos) en los cuales se hallan un gran volumen de paquetes que va aumentando exponencialmente en releases posteriores.

Por ello, se necesita seguir una secuencia específica representada en la **Figura 4.1** para descargar, limpiar, ordenar y extraer estos datos de forma efectiva.

Primero debemos descargar los diferentes releases de Debian desde su página oficial [7]. Con ello ya podremos analizar los diferentes paquetes que contienen y así poder diseñar el

diagrama entidad - relación correspondiente.

Una vez tenemos el diseño completo, podremos comenzar con la creación de las diferentes BBDD (bases de datos) tal que cada release sea una BBDD con sus correspondientes tablas sql.

En este punto creamos un script cuya funcionalidad será parsear los diferentes paquetes de cada release con el fin de extraer la información necesaria.

Una vez extraída dicha información creamos otro script que, conectándose a la BBDD correspondiente, introduzca los datos parseados en la BBDD para su posterior análisis.

Se crean una serie de Queries o llamadas sql para poder obtener la información que necesitamos y así responder a los intereses del proyecto.

Por último, creamos una serie de gráficas y tablas para ayudar a la visualización de los resultados obtenidos en este estudio.

4.2. Descarga de los releases de Debian

Esta fase es la más rápida.

La **Figura 4.2** es una representación de la lógica seguida para realizar la descarga de dichos releases entrando en la página oficial de Debian y, dependiendo de la antigüedad del release, entramos en archive o no para encontrarlo.

Conociendo las páginas donde encontrar los releases retirados [8] y los que siguen en uso [7] (aportados por mi tutor Gregorio), podemos descargar sus diferentes archivos **main** en la **Figura 4.3** donde encontramos todos los paquetes de cada lanzamiento y con ello los datos que queremos tratar.

4.3. Análisis de los paquetes de cada release

En esta fase se requiere un estudio detallado de los diferentes atributos de cada paquete. Asignamos las entidades y las relaciones existentes entre los mismos para poder construir un diagrama valido para el diseño de las bases de datos posteriores.

Un paquete esta conformado por los atributos mostrados en la **Figura 4.4**.

En la **Figura 4.5** explicamos el flujo de como se ha realizado el estudio de los paquetes asignando las diferentes relaciones a las entidades.

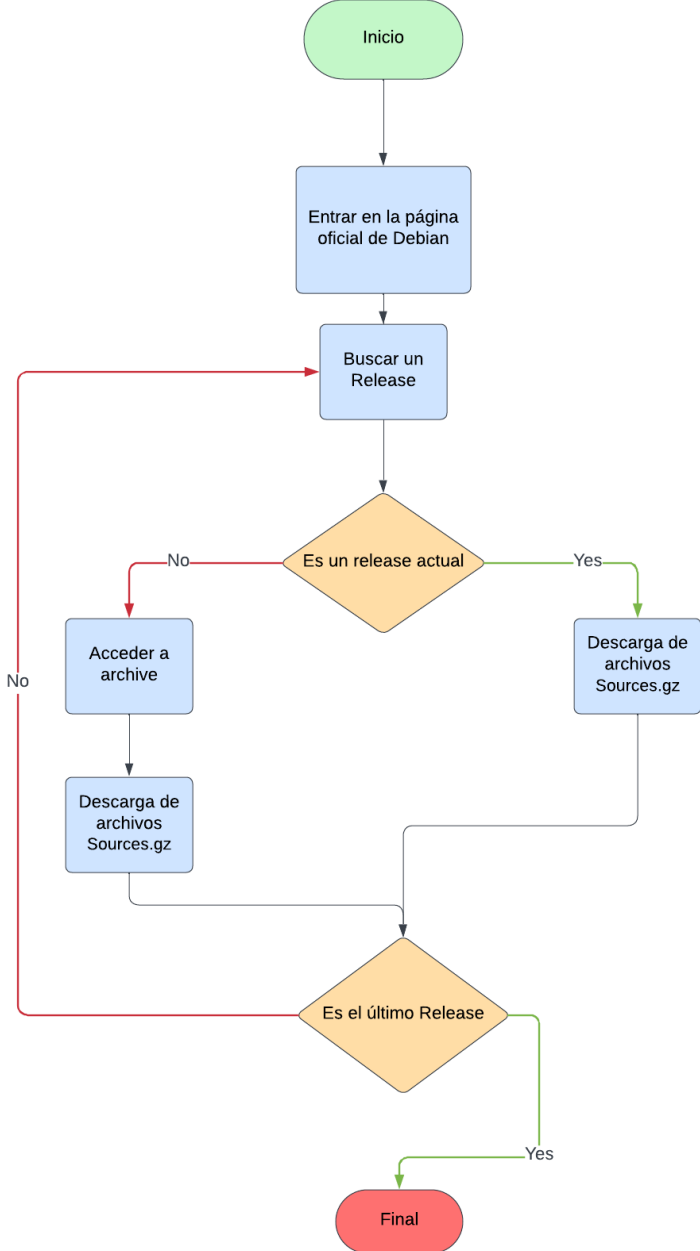


Figura 4.2: Diagrama de Flujo de la descarga de los Releases












 Debian-0.93R6/	2008-10-31 19:43	-
 bo/	1998-12-08 22:17	-
 buster-backports-sloppy/	2024-03-10 17:24	-
 buster-backports/	2024-03-10 17:24	-
 buster-proposed-updates/	2024-03-10 17:24	-
 buster-updates/	2024-03-10 17:24	-
 buster/	2024-03-10 17:24	-
 buzz/	2008-10-31 23:13	-
 etch-m68k/	2010-06-20 20:24	-
 etch/	2010-06-20 20:23	-
 hamm-proposed-updates/	2008-11-01 01:09	-

Figura 4.3: Descarga de los datos de releases

```

Package: abydos
Binary: python3-abydos, python-abydos-doc
Version: 0.5.0+git20201231.344346a-6
Maintainer: Debian Python Team <team+python@tracker.debian.org>
Uploaders: Julian Gilbey <jdg@debian.org>
Build-Depends: debhelper-compat (= 13), dh-sequence-python3, python3-all, python3-deprecation, python3-lzss <!nocheck>, python3-nltk
<!nocheck>, python3-numpy, python3-paq <!nocheck>, python3-setuptools, python3-sphinx-rtd-theme <!nodoc>, python3-sphinxcontrib.bibtex
<!nodoc>, python3-syllabipy <!nocheck>, sphinx <!nodoc>
Architecture: all
Standards-Version: 4.6.0
Format: 3.0 (quilt)
Files:
d52f9c58f1b5be8c12a554e72aa48ff5 2491 abydos_0.5.0+git20201231.344346a-6.dsc
2bbc1c5e23fc5778ad17f6ebf8bf919e 21335891 abydos_0.5.0+git20201231.344346a.orig.tar.gz
d0c4def13f06da9410fda77adc191f01 6284 abydos_0.5.0+git20201231.344346a-6.debian.tar.xz
Vcs-Browser: https://salsa.debian.org/python-team/packages/abydos
Vcs-Git: https://salsa.debian.org/python-team/packages/abydos.git
Checksums-Sha256:
926a65bc99c7cf4feb0c85aaec9d402e97f8234d653fc4f5df1907301ee53896 2491 abydos_0.5.0+git20201231.344346a-6.dsc
08e36ee602f03a5bd704d6a71ecb99de4713a4c483888a2abed65ed5a7ddc81c 21335891 abydos_0.5.0+git20201231.344346a.orig.tar.gz
9d93e6681a5c85923fafcecca1a8e63c1c4ab9bd74956e45bb6b10e5d9e1018cd 6284 abydos_0.5.0+git20201231.344346a-6.debian.tar.xz
Homepage: https://github.com/chrislit/abydos
Package-List:
python-abydos-doc deb doc optional arch=all profile=!nodoc
python3-abydos deb python optional arch=all
Testsuite: autopkgtest
Testsuite-Triggers: python3-all
Directory: pool/main/a/abydos
Priority: extra
Section: misc

```

Figura 4.4: Atributos de un paquete.

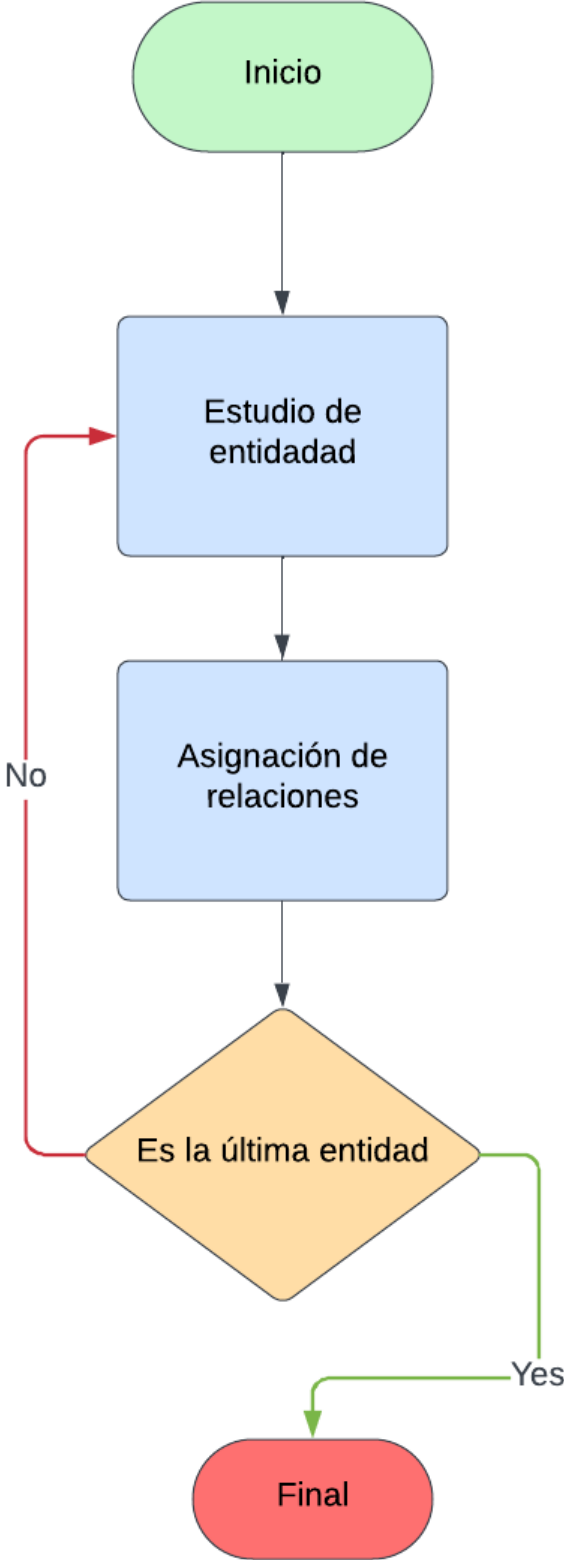


Figura 4.5: Flujo del estudio de los atributos de un paquete

El siguiente proceso consta de la comparación de diferentes paquetes para entender que entidades hay y como relacionarlas:

- **Package nombre:** siempre hay uno y es único.
- **Binary:** mínimo hay uno o más.
- **Version:** siempre hay uno y puede repetirse en varios paquetes.
- **Maintainer:** siempre hay uno y puede repetirse en varios paquetes.
- **Uploaders:** mínimo hay uno o más y puede repetirse en varios paquetes.
- **Build-Depends:** mínimo hay uno o más y puede repetirse en varios paquetes.
- **Architecture:** siempre hay uno y puede repetirse en varios paquetes.
- **Standards-Version:** siempre hay uno y puede repetirse en varios paquetes.
- **Format:** siempre hay uno y puede repetirse en varios paquetes.
- **Files:** mínimo hay uno o más y son únicos.
- **Vcs-Browser:** (no está en todos los paquetes) máximo uno y es único.
- **Vcs-Git:** (no está en todos los paquetes) máximo hay uno y es único.
- **Checksums-Sha256:** mínimo hay uno o más y son únicos.
- **Homepage:** (no está en todos los paquetes) máximo hay uno y puede repetirse en varios paquetes.
- **Package-List:** mínimo hay uno o más y son únicos.
- **Directory:** siempre hay uno y es único.
- **Dgit:** (no está en todos los paquetes) máximo hay uno y es único.
- **Testsuite:** (no está en todos los paquetes) máximo hay uno y puede repetirse en varios paquetes.

- **Testsuite-Triggers:** (no está en todos los paquetes) puede haber uno o más y puede repetirse en varios paquetes.
- **Priority:** siempre hay uno y puede repetirse en varios paquetes.
- **Section:** siempre hay uno y puede repetirse en varios paquetes.

Con esta información podemos comenzar con el diseño del diagrama.

4.4. Creación diagrama Entidad - Relación

Un diagrama **entidad-relación**, también conocido como **modelo entidad relación o ERD** [12], es un tipo de diagrama de flujo que ilustra cómo las entidades, como personas, objetos o conceptos, se relacionan entre sí dentro de un sistema.

Los diagramas ER se usan a menudo para diseñar o depurar bases de datos relacionales en los campos de ingeniería de software, sistemas de información empresarial, educación e investigación. También emplean un conjunto definido de símbolos, tales como rectángulos, diamantes, óvalos y líneas de conexión para representar la interconexión de **entidades, relaciones y sus atributos**.

En la Figura 4.6 se observa el diseño del diagrama junto con la Figura 4.7 que es su leyenda.

Cada rectángulo es una entidad que va unida a otras entidades mediante una serie de líneas que son los indicadores del tipo de relación que muestran.

En este caso todas las entidades tienen una relación de pertenencia debido a que un paquete contiene o no estas entidades.

En la **Figura 4.7** podemos ver el tipo de relación entre entidades:

1. Se muestra una relación de **1 a 1**, es decir, solo puede existir una entidad en el paquete obligatoriamente.
2. Se muestra una relación de **0 a 1**, es decir, pueden existir ninguna o una entidad en el paquete.
3. Se muestra una relación de **1 a infinito**, es decir, pueden existir una o muchas entidades en el paquete pero nunca puede no existir.

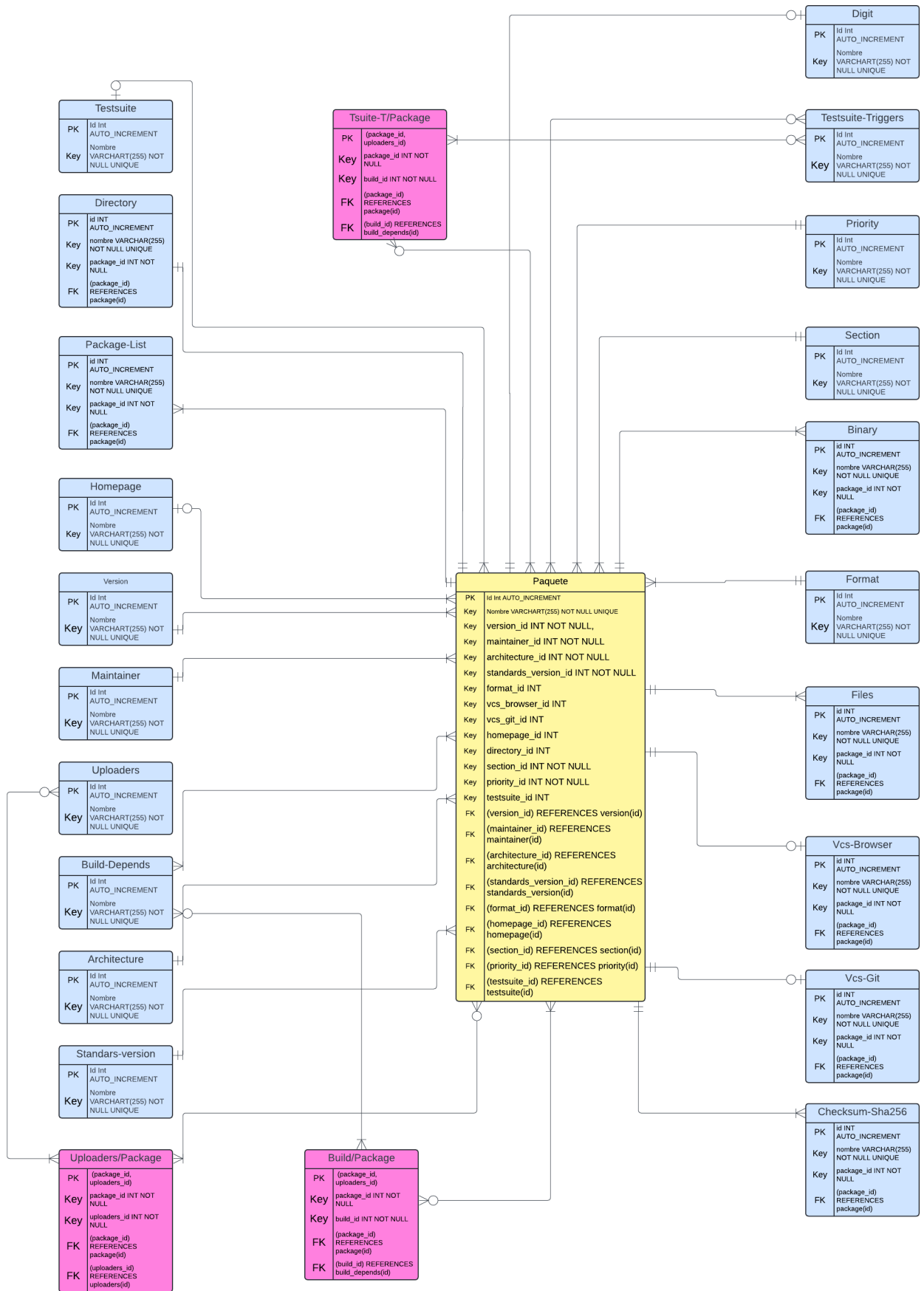


Figura 4.6: Diagrama ER.

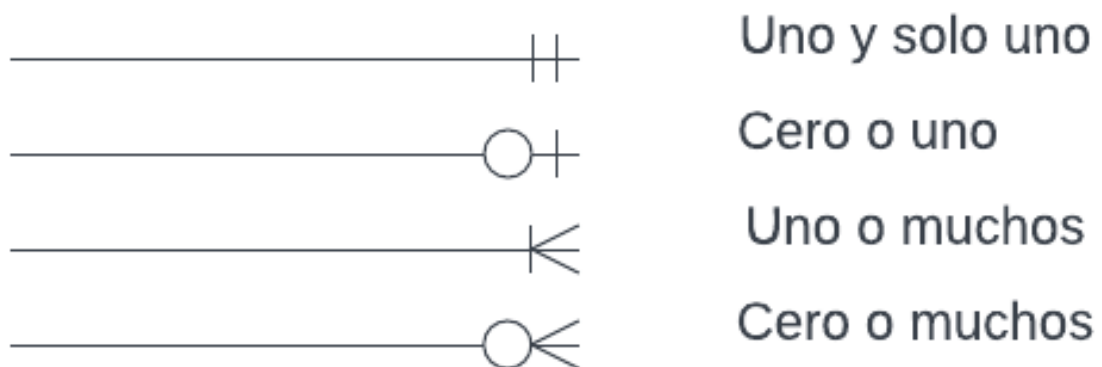


Figura 4.7: Leyenda del Diagrama ER.

4. Se muestra una relación de **0 a infinito**, es decir, pueden existir ninguna, una o muchas entidades en el paquete.

En este tipo de diagrama podemos observar como están diseñadas las tablas de la BBDD y como están relacionadas entre sí.

En la Figura 4.6 identificamos diferentes tablas:

- **Tabla amarilla:** esta tabla es la principal ya que es **Package**. Tiene relaciones con todas las demás tablas. En ella podemos observar su **Primary Key** que es su **Id o identificador**, una serie de campos que son atributos (id's del resto de tablas) y sus **Foreign Key** que permiten poder relacionar las demás tablas con ella misma y así obtener valores correctos al realizar llamadas **SQL mediante Queries**.
- **Tablas azules:** estas tablas son las diferentes entidades que se encuentran en un paquete y están unidas a la tabla principal mediante las líneas, explicadas anteriormente, que muestran las relaciones que tienen. Cada una tiene su **Primary Key** que es su identificador, los atributos como el nombre y algunas contienen **Foreign Key** debido al tipo de relación entre tablas.
- **Tablas rosas:** estas son tablas intermedias que son necesarias para conectar tablas en las cuales no se puede insertar una **Foreign Key** en ninguna de las dos y por lo tanto no pueden

relacionarse. Creando este tipo de tablas forzamos una conexión. Estas contienen una **Primary Key** especial conformada por los identificadores de ambas tablas y las **Foreign Key** asociadas a las tablas para poder relacionarlas. Son casos especiales en relaciones de **muchos a muchos**.

Una vez completadas las relaciones junto con las entidades se diseña la BBDD que va a acoger todos aquellos datos necesarios para el estudio.

En la **Figura 4.8** podemos visualizar el método empleado para diseñar este diagrama. Para ello partimos de las entidades y las relaciones estudiadas anteriormente. Después asignamos las diferentes **Primary Keys** y por último comprobamos que tipo de relación tienen las entidades para crear tablas intermedias o asignar una **Foreign key** directamente.

4.5. Creación de BBDD junto con tablas SQL

En este punto se usan las aplicaciones MySQL Workbench y Pycharm. Se crean las diferentes BBDD en MySQL para poder realizar la inserción de las tablas posteriormente.

En la **Figura 4.10** podemos visualizar el flujo de trabajo realizado para el diseño y creación de las BBDD. Primero creamos manualmente las BBDD en MySQL Workbench. Creamos un script donde introducimos las credenciales necesarias, diseñamos las tablas en lenguaje SQL e insertamos dichas tablas.

El diseño de la BBDD depende directamente del diagrama realizado anteriormente. Este diagrama tiene una traducción a lenguaje SQL:

- **Entidad 1 (uno y solo uno) Entidad 2:** cuando esto ocurre necesitamos crear una **foreign key** para relacionar las tablas de cada entidad. **Entidad 1** debe tener una foreign key para la **Entidad 2**.

```
"""  
CREATE TABLE IF NOT EXISTS package_list (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  nombre VARCHAR(255) NOT NULL UNIQUE,  
  package_id INT NOT NULL,  
  FOREIGN KEY (package_id) REFERENCES package(id)
```

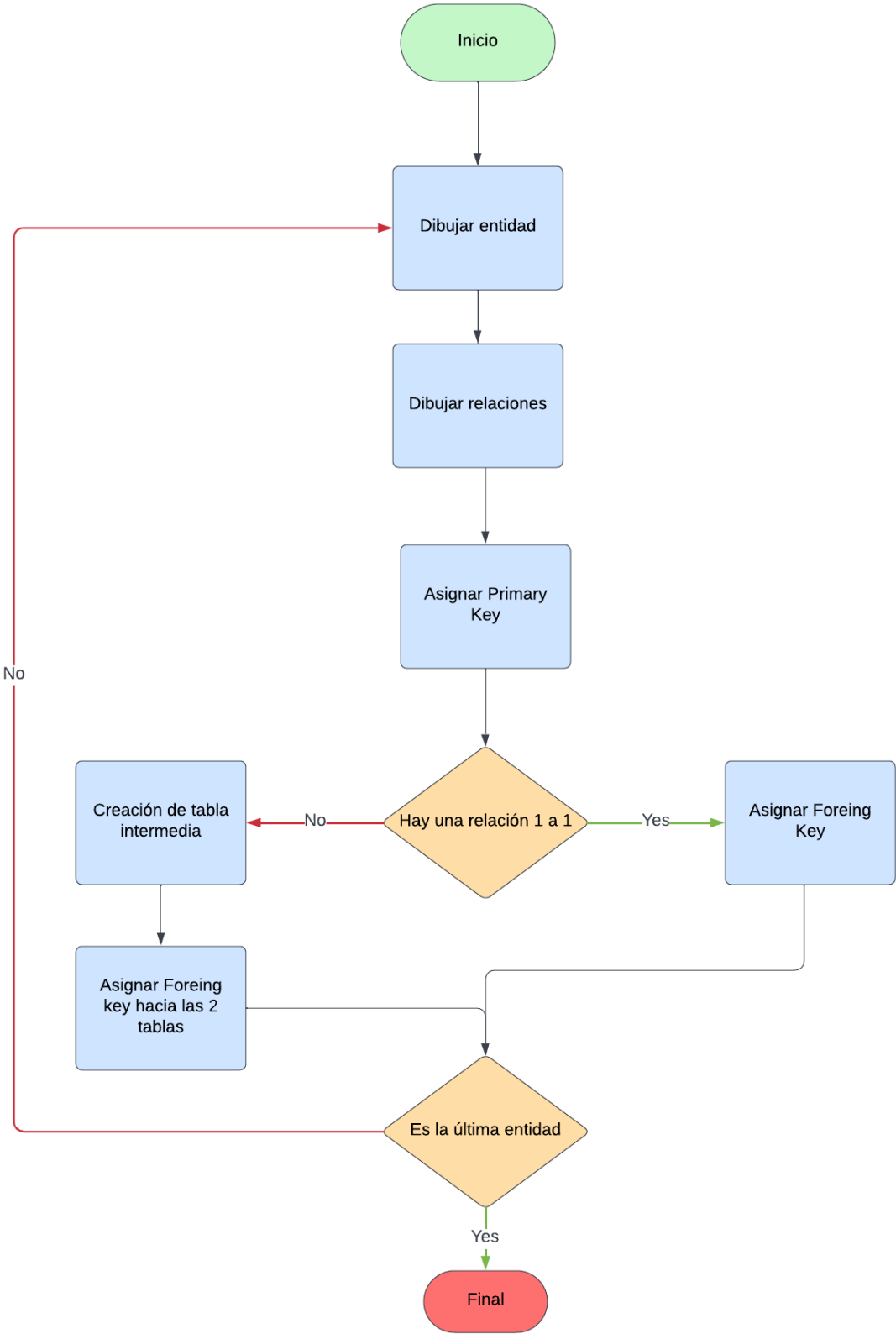


Figura 4.8: Flujo de creación de diagrama Entidad - Relación

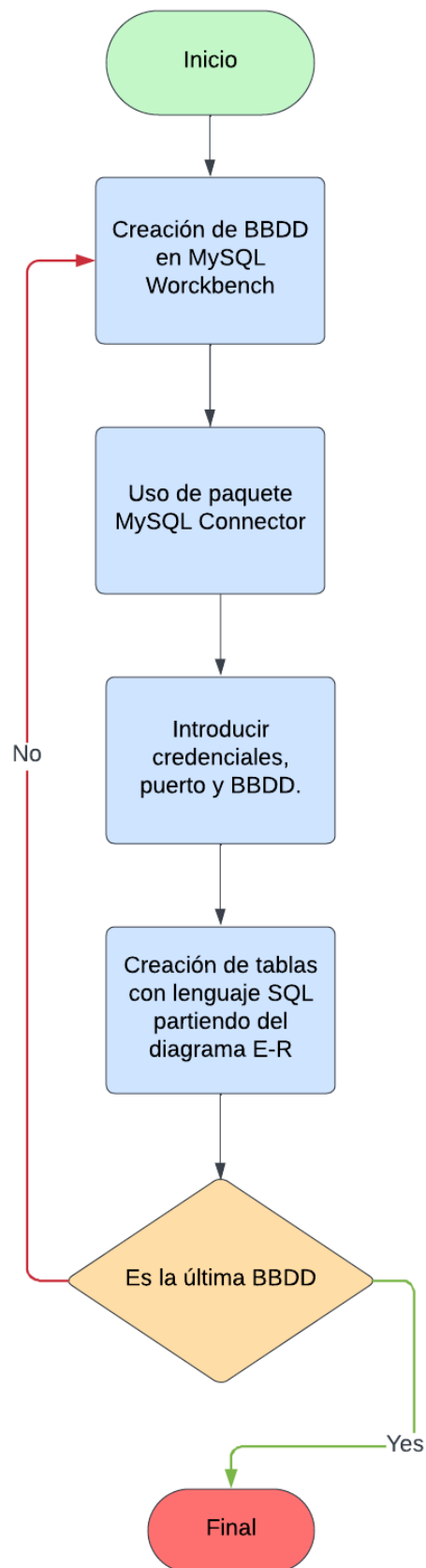


Figura 4.9: Flujo de creación de BBDD

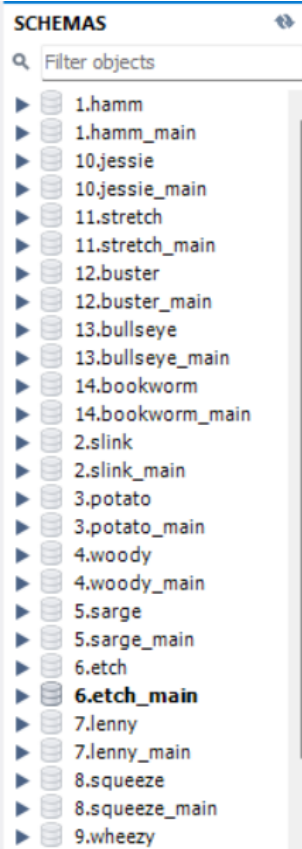


Figura 4.10: Bases de datos

```
)
"""
```

Una **FOREIGN KEY** [15] en SQL, es una clave (campo de una columna) que sirve para relacionar dos tablas. El campo FOREIGN KEY se relaciona o vincula con la PRIMARY KEY de otra tabla de la bbdd.

La restricción **PRIMARY KEY** [16] SQL identifica de forma única cada registro en una tabla. Deben contener valores únicos y no pueden contener valores NULL. Una tabla solo puede tener una clave principal, que puede consistir en campos simples o múltiples.

- **Entidad 1 (uno o muchos) Entidad 2:** en este caso no tenemos claro que entidad debería acoger la foreign key por lo que se crea una tabla intermedia. Esta tabla constará de los identificadores de cada tabla como sus primary key. En ella es en la que se definirán las foreign key hacia ambas tablas y así poder relacionarlas.

```
"""
CREATE TABLE IF NOT EXISTS uploaders_package (
package_id INT NOT NULL,
uploaders_id INT NOT NULL,
PRIMARY KEY (package_id, uploaders_id),
FOREIGN KEY (package_id) REFERENCES package(id),
FOREIGN KEY (uploaders_id) REFERENCES uploaders(id)
)
"""
```

Finalmente, se procede al desarrollo de un script que se conecte a la BBDD para crearlas y comenzar a usarlas.

Para ello, debemos establecer una conexión con la BBDD indicando el host, puerto, usuario, contraseña y nombre de la BBDD.

Todo se realiza a través de Python y con la librería `mysql.connector`.

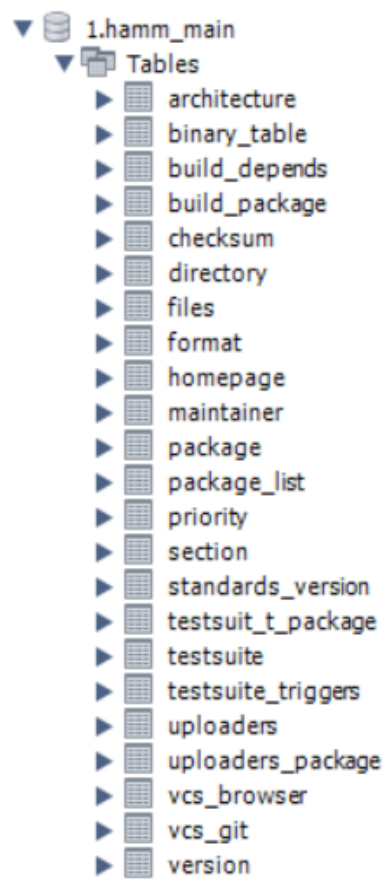


Figura 4.11: Tablas BBDD

4.6. Parseo de la información de los paquetes

Tras la creación de las diferentes BBDD con sus tablas correspondientes, ahora debemos enfocarnos en obtener la información de interés para introducirla en dichas tablas SQL.

Para ello se diseña un script en Pycharm siguiendo estos pasos:

1. **Lectura del archivo.** Las descargas de los releases realizadas en la sección 4.2 las tratamos como ficheros de texto. Por ello debemos leer línea por línea todo el fichero e ir guardando la información relevante de cada paquete.

```
with open('Sources/14.Sources - bookworm-main.txt', 'r',
         encoding='utf-8') as file:
    lines = iter(file.readlines())
```

2. **Especificar qué datos son relevantes.** Se crea un array de strings con los diferentes nombres de los campos que queremos sacar para verificar si alguno se encuentra en la línea en la que estamos leyendo.

```
array_items = ['Package', 'Binary', 'Version', 'Maintainer',
              'Uploaders', 'Build-Depends', 'Architecture',
              'Standards-Version', 'Format', 'Files', 'Vcs-Browser',
              'Vcs-Git', 'Checksums-Sha256', 'Homepage',
              'Package-List', 'Directory', 'Priority', 'Section',
              'Testsuite', 'Testsuite-Triggers']
```

3. **Buscar los datos relevantes en el archivo.** Con el diseño del array ya podemos encontrar la información de interés. Recorremos línea a línea el archivo, comparamos si esa línea empieza con alguno de los items definidos seguido de : y parseamos la información guardándola en unas variables definidas anteriormente.

```
for line in lines:
```

```

field, *value = map(str.strip, line.split(':', 1))
value = value[0] if value else ''

if field in array_items:

    if field == "Package":
        package = value

    elif field == "Binary":
        binary = value

    ...

```

4. **Identificar el final del paquete.** En un archivo hay miles de paquetes juntos separados por una línea en blanco. Este será nuestro identificador de que hemos recogido la información de un paquete. Si encuentra dicha línea, se llama a la función que nos permitirá insertar esta información en las diferentes tablas.

```

elif not line.strip():
# Esto indica el final de la información del paquete
insertar_info(package, binary, version, maintainer,
    uploaders, build_depends, architecture,
standards_version, format1, files_list, vcs_browser,
    vcs_git, check_list, homepage,
package_list, directory, priority, section, testsuite,
    testsuite_triggers)

```

5. **Restaurar las variables.** Este proceso es iterativo debido a que se realiza para cada paquete dentro de cada release. Necesitamos vaciar las variables para volver a obtener los datos de los próximos paquetes.

4.7. Inserción de la información en la BBDD

Una vez parseada la información de un paquete, debemos realizar la inserción en la BBDD. Esto se ejecuta mediante otro script en Pycharm y con la librería `mysql.connector`.

En la **Figura 4.12** se representa el flujo de trabajo realizado para insertar los datos necesarios en la BBDD. Definimos en un array los campos que queremos extraer de los paquetes, parseamos y guardamos los datos en variables. Si hemos obtenido todos los datos de un paquete, insertamos dichos datos en las tablas SQL respectivamente de cada BBDD.

1. **Conexión a la BBDD.** Primero debemos conectarnos a la BBDD para obtener la funcionalidad de insertar en ella. Debemos introducir los siguientes datos:

```
conexion = mysql.connector.connect (  
    host='localhost',  
    port=3306,  
    user='root',  
    password='xxxx',  
    db='14.bookworm_main'  
)
```

2. **Creación de diccionario.** Necesitamos asociar los diferentes valores, de las variables de la sección 4.6, a los nombres de las tablas para facilitar la lógica de dicho script.

```
datos = {  
    "version": version,  
    "maintainer": maintainer,  
    "uploaders": uploaders,  
    "build_depends": build_depends,  
    "architecture": architecture,  
    "standards_version": standards_version,  
    "format": format1,  
    "section": section,
```

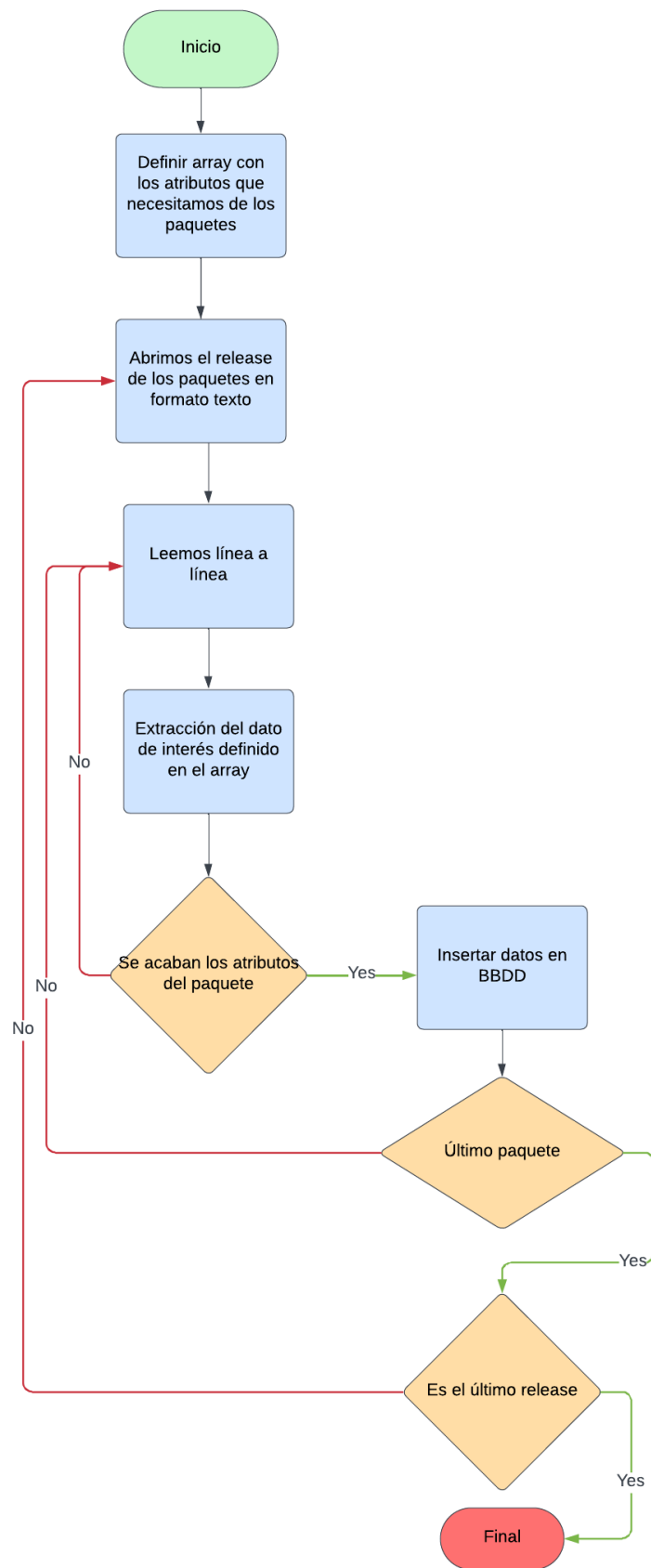


Figura 4.12: Flujo de inserción de los datos en BBDD

```
"priority": priority,
...
```

3. **Inserción de los datos.** Finalmente pasamos a el objetivo final de este punto, la inserción. Tenemos varios casos que deben ser contemplados para su optima realización:

- **Valores vacíos:** en este caso debemos comprobar si la variable en cuestión tiene un valor predeterminado o viene vacío. Hay paquetes en los que no existen ciertas entidades como vimos en la sección 4.3. Se actualiza el valor a **None** para evitar fallos en la BBDD al insertar.
- **Datos duplicados:** hay entidades que se repiten en múltiples paquetes como las versiones o los maintainer. Debemos tener en cuenta que, al crear las tablas, se ponen restricciones como los identificadores únicos. Esto significa que no se pueden meter duplicados y por ello debemos tenerlo en cuenta. Para ello, buscamos si el valor ya se encuentra en la tabla, si se confirma, se ignora para evitar errores en las tablas SQL.
- **Valores múltiples:** hay entidades que tienen varios datos separados por comas como Build-Depends. En estos casos debemos separarlo por comas creando un array. Después, se recorre el array y se va metiendo uno a uno verificando que no sean duplicados.

```
Build-Depends: debhelper-compat (= 13), libsamplerate0-dev, libSDL2-dev, libSDL2-mixer-dev
```

Figura 4.13: Build Depends

- **Files o Checksums-Sha256:** son 2 casos en los que no tenemos comas y vienen varios datos por entidad. Cuando verifiquemos que una línea del paquete empieza por estos 2 casos debemos cambiar la lógica. Saltamos a las siguientes líneas guardando todos los datos en un array. Posteriormente se recorrerá y se insertarán en las diferentes tablas.

Por último, hay que vincular diferentes tablas a través de sus foreign keys.


```
Files:
266b8669362d98c0eef2a3fe83c1b29a 1924 loom_1.0-2.dsc
3001abc0dcd309815de06fd24277a325 533439 loom_1.0.orig.tar.gz
98832941cbf539d134971d55ca97a081 4028 loom_1.0-2.debian.tar.xz
```

Figura 4.14: Atributo Files de cada paquete en el que se muestran diferentes archivos contenidos en él.

Al insertar un paquete, guardamos su id. De esta forma, al insertar otra entidad diferente (en otra tabla) sabemos que id de paquete tiene y con ello podemos referenciarlo sin problemas.

Esto es de suma importancia debido al diseño de las posteriores Queries.

4.8. Creación de Queries SQL

Para este proceso se necesita de nuevo un script en Pycharm que se conecte a la BBDD y, a través de lenguaje SQL, realice llamadas específicas a la misma. De esta forma obtendremos la información que se necesita.

En la **Figura 4.15** podemos visualizar el flujo de trabajo usado para crear las queries. Primero definimos un array con todas las BBDD, realizamos la conexión mediante las credenciales, diseñamos unas queries en lenguaje SQL y mostramos los resultados por pantalla.

1. **Conexión a la BBDD.** Primero debemos conectarnos a la BBDD para obtener la funcionalidad de realizar llamadas sobre ella como en la sección 4.7.
2. **Lenguaje de manipulación de datos [9].** Proporcionado por el sistema de gestión de base de datos, permite a los usuarios llevar a cabo las tareas de consulta o manipulación de los datos, organizados por el modelo de datos adecuado.
 - **Comandos:** para insertar como en la sección 4.7, actualizar o borrar pero en este caso se quiere consultar por lo que usaremos ‘SELECT’.
 - **Clausulas:** son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular. En nuestro caso usaremos ‘WHERE’.
 - **Operadores Lógicos:** como **AND**, **OR** y **NOT**.

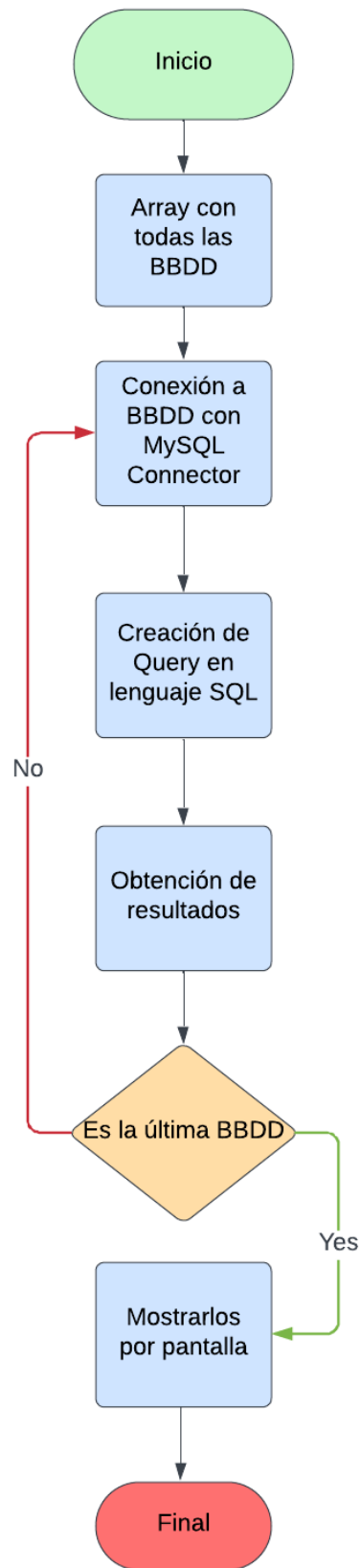


Figura 4.15: Flujo de creación de queries

- **Operadores de comparación:** como **LIKE, IN, BETWEEN, etc.**

3. **Queries.** Se crean diferentes funciones para separar las llamadas y facilitar la comprensión del código.

Este sería un ejemplo de una Query que devuelve el numero de maintainers total que hay en cada release o BBDD:

```
...
consulta = "SELECT COUNT(*) FROM maintainer"
cursor.execute(consulta)
...
```

Cuenta el número de mantenedores que hay en la tabla ‘maintainer’ de la base de datos actual. Esta consulta va dentro de un bucle que hace el cálculo para todas las BBDD.

Cada llamada contesta a las preguntas realizadas para este estudio obteniendo de forma eficiente la información de las BBDD.

4.9. Obtención de gráficas informativas

En este apartado sigue siendo necesario la creación de un script en Pycharm pero con la librería ‘Matplotlib’.

Con el diseño de las queries del apartado 4.8 obtenemos la información necesaria para responder a las diferentes cuestiones planteadas en este estudio. Para ayudar a la comprensión de estos datos debemos crear diferentes gráficas. Visualizar los resultados facilita la interpretación de los mismos para obtener unas conclusiones óptimas.

Para ello es necesario:

1. **Queries del script anterior:** como explicamos anteriormente, vamos a realizar una representación visual de la información obtenida por las queries por lo que son imprescindibles.

```
resultados_paquetes = obtener_numero_paquetes(array_database1)
```

2. **Definir los ejes:** estas gráficas aportarán información del número de paquetes existentes y el número de mantenedores que trabajan con cada versión. Debemos definir cada eje con sus correspondientes etiquetas para evitar confusiones.

```
x = ["Julio1998", "Marzo1999", "Agosto2000", "Julio2002",  
     "Junio2005", "Abril2007", "Febrero2009", "Mayo2014",  
     "Junio2016", "Junio2018", "Junio2020", "Septiembre2022",  
     "Febrero2024"]
```

- **Eje x:** fechas de los lanzamientos de los diferentes releases.
 - **Eje y:** número de paquetes o mantenedores (dependiendo de la gráfica) que hay en cada release.
3. **Conversión del eje x a objeto fecha:** este es uno de los detalles más importantes para presenciar y entender de forma correcta la información. Se trata de convertir cada etiqueta del eje x como 'Junio2022' a un objeto temporal. De esta forma se representará un espacio mayor o menor en el eje x dependiendo de la distancia entre fechas de lanzamiento de los releases.
 4. **Creación del gráfico:** en nuestro caso será un gráfico de barras. Para ello definimos el tipo de gráfica que queremos, agregamos los datos obtenidos por las queries, configuramos los diferentes ejes junto con el título de la gráfica y la mostramos.

En la **Figura 4.16** se representa el flujo de trabajo realizado para la creación de las gráficas. Comenzamos con los resultados de las queries anteriores. Luego definimos los ejes (el eje x en formato temporal), insertamos los resultados en la gráfica y lo mostramos por pantalla.

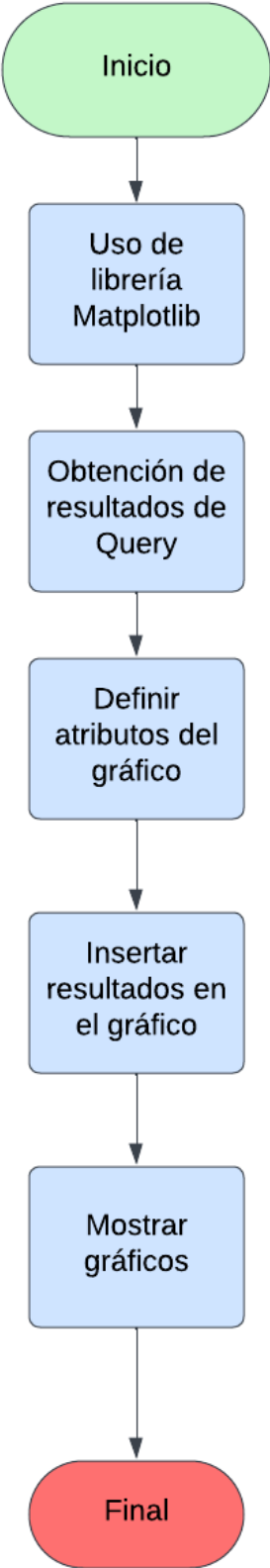


Figura 4.16: Flujo de creación de gráficas

Capítulo 5

Resultados

En este capítulo se muestran los diferentes resultados a las preguntas que nos realizamos en este proyecto. De esta forma podremos observar la información obtenida y sacar conclusiones a partir de ella.

5.1. ¿Cuántos mantenedores tiene Debian y cómo cambia este número con el tiempo?

Esta distribución lanza nuevas versiones cada cierto tiempo como mostramos en la sección 1.1.1. Cada versión está formada por múltiples paquetes y cada paquete lo mantiene una persona (aunque puede mantener varios paquetes a la vez). El número de mantenedores de dichos paquetes no es fijo ya que en cada versión pueden incorporarse nuevos o, por el contrario, abandonarlo.

Y por ello nos preguntamos, **¿cual es el número de mantenedores en cada versión? Y si no es fijo, ¿aumenta o disminuye en versiones posteriores?**

Para responderla se han aplicado unas queries con las que se obtiene el número total de maintainers únicos en cada versión/release.

Una vez ejecutadas las queries se obtienen estos resultados:

Una vez obtenidos los resultados podemos mostrar las gráficas para visualizarlos.

En la **Figura 5.1** se observa el número de mantenedores totales y únicos de por cada release. Gracias a que el eje x es un objeto temporal, podemos apreciar la diferencia entre tiempos de

Fecha	Release	Maintainers
Julio 1998	Hamm	255
Marzo 1999	Slink	362
Agosto 2000	Potato	530
Julio 2002	Woody	986
Junio 2005	Sarge	1,490
Abril 2007	Etch	1,760
Febrero 2009	Lenny	1,971
Mayo 2014	Squeeze	2,173
Junio 2016	Wheezy	2,286
Junio 2018	Jessie	2,468
Junio 2020	Stretch	2,412
Septiembre 2022	Buster	2,413
Febrero 2024	Bullseye	2,293

Cuadro 5.1: Tabla de versiones de Debian

lanzamiento sobre cada release.

- Se encuentran dos parones en los lanzamientos de los release entre julio de 2002 y junio de 2005, y entre febrero de 2009 y mayo de 2014 (alargandose más en el tiempo que el anterior).
- También podemos ver la tendencia que sigue la gráfica con respecto a los maintainers. Hasta Junio de 2018 hay una subida del número de maintainers pero a partir de este release se aprecia una pequeña bajada de los mismos.

También se ha calculado el número de paquetes que encontramos en cada release mediante una query y podemos ver el resultado en la **Figura 5.2**.

Como se puede apreciar, hay un aumento exponencial en la cantidad de paquetes llegando hasta los 30,795 en la última versión.

5.1. ¿CUÁNTOS MANTENEDORES TIENE DEBIAN Y CÓMO CAMBIA ESTE NÚMERO CON EL TIEMPO?

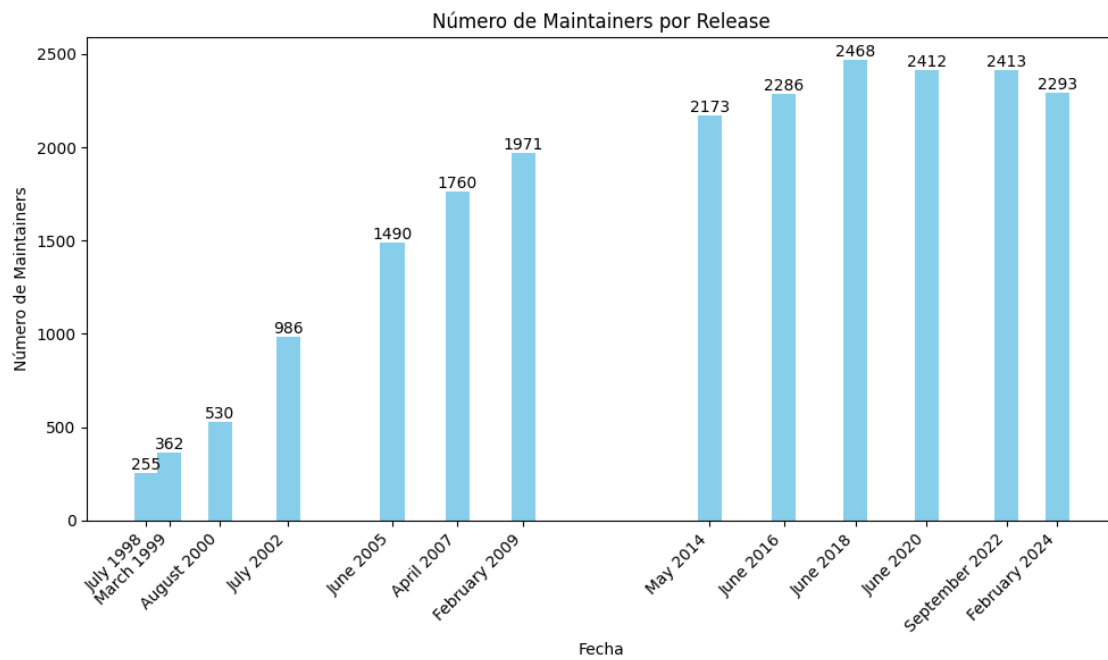


Figura 5.1: Número de maintainers de cada release

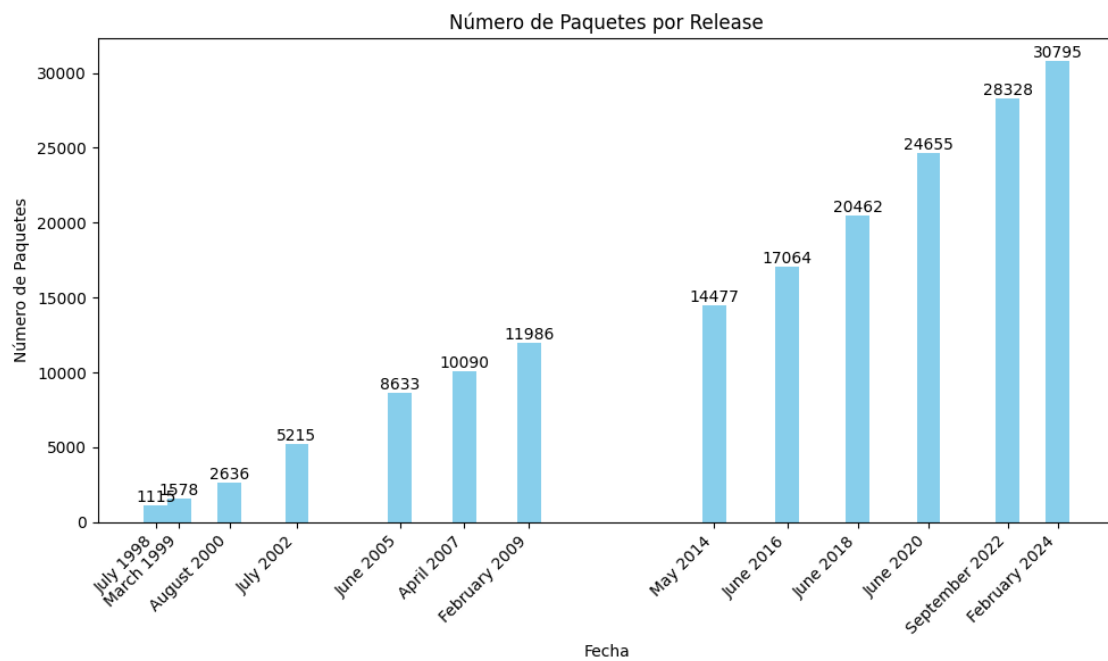


Figura 5.2: Número de paquetes de cada release

```

Package: bibindex
Priority: optional
Section: tex
Version: 2.8-1
Binary: bibindex
Maintainer: Debian QA Group <debian-qa@lists.debian.org>
Architecture: any|
Standards-Version: 2.4.0.0
Directory: dists/hamm/main/source/tex
Files:
 31fd38796715421fe08f5c88e39e6fc9 629 bibindex_2.8-1.dsc
 958806cd1cfdc0159e9da12205ebdd11 53372 bibindex_2.8.orig.tar.gz
 69127335cecd88837fed160bb1b23a6 2505 bibindex_2.8-1.diff.gz

```

Figura 5.3: Ejemplo de nombre de equipo de maintainers

5.2. ¿Existe una tendencia hacia la formación de equipos de mantenedores?

La distribución Debian estaba conformada en sus orígenes por una serie de personas que trabajaban para su desarrollo. Con el paso del tiempo comenzaron con la agrupación de estos para la creación de grupos Debian. Con ello se organizarían mejor y podrían desempeñar un óptimo trabajo especializándose en un área concreta.

Por ello nos preguntamos si existiera dicha tendencia a la creación de equipos y como varía con el tiempo.

Podemos diferenciar que maintainers son equipos o personas individuales debido al ‘nombre’ del campo ‘maintainer’ de cada paquete. Estos nombres suelen contener la palabra **Team** o **Group** como vemos en las **Figuras 5.3 y 5.1**

Para ello lanzamos la query para obtener el número de equipos en cada release.

Una vez ejecutado estos son los resultados:

Obteniendo los datos analíticamente, mostramos la gráfica para visualizar la información.

Podemos apreciar observando la **Figura 5.5** la tendencia que sigue la creación de equipos. Es claramente ascendente llegando hasta los 317 en su último release de 2024.

Se distinguen cuatro fases.

5.2. ¿EXISTE UNA TENDENCIA HACIA LA FORMACIÓN DE EQUIPOS DE MANTENEDORES?49

```
Package: apt
Priority: optional
Section: admin
Version: 0.3.10slink11
Binary: apt, libapt-pkg-dev, libapt-pkg-doc
Maintainer: APT Development Team <deity@lists.debian.org>
Architecture: any
Standards-Version: 2.4.1
Directory: dists/slink/main/source/admin
Files:
104ed7e94445e6225f53984636ea65a5 597 apt_0.3.10slink11.dsc
301967aa90a9a48cf8a84d92e3858ad1 396974 apt_0.3.10slink11.tar.gz
```

Figura 5.4: Ejemplo de nombre de equipo de maintainers

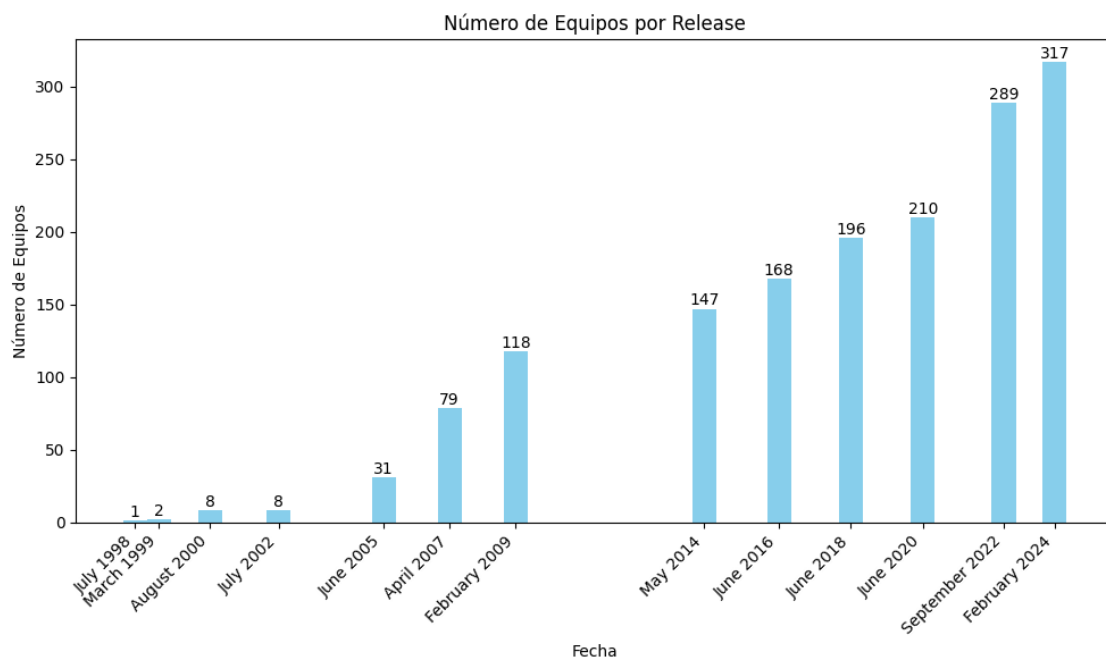


Figura 5.5: Equipos formados en cada release

Fecha	Release	Equipos
Julio 1998	Hamm	1
Marzo 1999	Slink	2
Agosto 2000	Potato	8
Julio 2002	Woody	8
Junio 2005	Sarge	31
Abril 2007	Etch	79
Febrero 2009	Lenny	118
Mayo 2014	Squeeze	147
Junio 2016	Wheezy	168
Junio 2018	Jessie	196
Junio 2020	Stretch	210
Septiembre 2022	Buster	289
Febrero 2024	Bullseye	317

Cuadro 5.2: Tabla de versiones de Debian con equipos

- **Fase 1:** los 4 primeros releases conformados por muy pocos equipos con un máximo de ocho de 1998 a 2002.
- **Fase 2:** los 3 siguientes a la ‘Fase 1’ cuentan con una subida sustancial de equipos llegando a un máximo de 118 de 2005 a 2009.
- **Fase 3:** los 4 siguientes a la ‘Fase 2’ cuentan con una subida en la conformación de equipos llegando a un máximo de 210 de 2014 a 2020.
- **Fase 4:** los 2 siguientes a la ‘Fase 3’ cuentan con una subida llegando a un máximo de 317 de 2005 a 2009.

5.3. ¿Cuántos mantenedores de versiones anteriores permanecen activos?

Como comentamos en la sección 5.1, con el paso del tiempo hay mantenedores nuevos que entran al proyecto y también existen mantenedores que lo abandonan. Por ello nos preguntamos cuantos de los mantenedores anteriores siguen en los releases posteriores y cual es el porcentaje de los mismos comparando los releases.

Como queremos comparar (en valor absoluto) cuantos maintainers se mantienen de un release a los demás, realizamos otra query para ello. Este es su resultado:

Release	1.Hamm	2. Slink	3.Potato	4. Woody	5. Sarge	6. Etch	7. Lenny	8. Squeeze	9. Wheezy	10. Jessie	11. Stretch	12. Buster	13. Bullseye
1. Hamm	255	242	210	155	121	98	83	71	69	64	54	48	46
2. Slink		362	314	241	186	153	132	106	95	88	73	67	64
3. Potato			530	412	300	243	198	169	151	140	125	109	101
4. Woody				986	726	576	487	410	359	333	292	268	236
5. Sarge					1490	1235	1039	855	728	668	573	519	457
6. Etch						1760	1483	1231	1055	949	805	713	628
7. Lenny							1971	1647	1418	1266	1076	941	829
8. Squeeze								2173	1859	1687	1409	1211	1046
9. Wheezy									2286	2029	1712	1470	1253
10. Jessie										2468	2082	1778	1511
11. Stretch											2412	2051	1742
12. Buster												2413	2059
13. Bullseye													2293

Cuadro 5.3: Valores absolutos de maintainers que permanecen activos en versiones posteriores

En la tabla anterior podemos ver cual es el valor absoluto de los maintainers de un release que se mantienen en el resto. Se lee de izquierda a derecha comparando el primero con el segundo, el primero con el tercero, el primero con el cuarto, etc.

Analizando esta tabla podemos observar que, según avanzan los releases, disminuye el número de maintainers que aguantan en versiones posteriores.

Ejecutamos una segunda query para observar los porcentajes de equipos de mantenedores:

En la tabla anterior se observa la evolución de los equipos o grupos que se mantienen de una versión en las posteriores en porcentajes.

Podemos ver como los grupos creados en los releases Hamm y Slink se mantienen en todos los releases pero en los demás va disminuyendo según se lanzan nuevas versiones.

Esto puede deberse a la creación de nuevos equipos que sustituyan a los anteriores para la

Release	1.Hamm	2. Slink	3.Potato	4. Woody	5. Sarge	6. Etch	7. Lenny	8. Squeeze	9. Wheezy	10. Jessie	11. Stretch	12. Buster	13. Bullseye
1. Hamm	1	1	1	1	1	1	1	1	1	1	1	1	1
2. Slink		2	2	2	2	2	2	2	2	2	2	2	2
3. Potato			8	4	4	4	4	4	4	4	4	4	4
4. Woody				8	7	4	4	4	4	4	4	4	4
5. Sarge					31	26	25	22	21	21	19	18	18
6. Etch						79	68	62	53	49	47	41	40
7. Lenny							118	104	93	90	78	66	63
8. Squeeze								147	130	124	109	83	78
9. Wheezy									168	153	131	99	91
10. Jessie										196	168	130	119
11. Stretch											210	166	150
12. Buster												289	263
13. Bullseye													317

Cuadro 5.4: Valores absolutos de equipos que permanecen activos en versiones

mejora de eficiencia al mantener paquetes.

También quisimos responder a la pregunta de cuál es la vida media de un maintainer, es decir, cuando se reduce a la mitad el número de maintainers de un release.

Ejecutamos un query en la que obtenemos dicho valor el cual tiene como resultado 3.5 releases.

Sabemos que no todos los maintainers de una versión se mantienen en las siguientes. Pero, ¿hay maintainers que comenzaron en la primera versión y hoy en día siguen activos?

Para resolverlo ejecutamos una query que, coge todos los maintainers únicos de cada versión y realiza un **set** [11].

Un 'set' es como una lista o un array pero con la peculiaridad de que sus items deben ser únicos, justo lo que necesitamos para resolver esta pregunta.

Una vez tengamos un 'set' con los maintainers de la primera versión, realizamos la **intersección** [2] con el 'set' de la segunda versión, y así sucesivamente.

Con una intersección entre las versiones conseguimos quedarnos únicamente con los maintainers que se repiten en todas las versiones. Estos son los que permanecerán activos.

Ejecutando la query obtenemos estos resultados:

- Manoj Srivastava
- Anthony Fok
- Riku Voipio
- Roderick Schertler
- Roberto Lumbreras
- Michael Meskes
- Davide G. M. Salvetti
- Juan Cespedes
- Craig Small

5.4. ¿CUÁL ES EL APORTE DE LOS MANTENEDORES QUE PERMANECEN EN VERSIONES POSTERIORES?

- Anselm Lingnau
- Marco d'Itri
- Fredrik Hallenberg
- Miquel van Smoorenburg
- Jaldhar H. Vyas
- Yann Dirson
- John Goerzen
- Bdale Garbee
- Hakan Ardo
- Dirk Eddelbuettel
- Matthias Klose
- Philip Hands
- Martin Buck
- Martin Schulze
- Craig Sanders
- Paul Sloodman
- Joel Rosdahl
- Norbert Veber

En esta lista/set encontramos los maintainers que llevan desde el inicio manteniendo paquetes de diferentes versiones en Debian.

5.4. ¿Cuál es el aporte de los mantenedores que permanecen en versiones posteriores?

Tras haber obtenido los maintainers que han estado activos en todas las versiones, queremos conocer que paquetes mantienen de cada versión.

Diseñamos una query para buscar en las BBDD que paquetes tienen, en el campo maintainer, el nombre de alguno de los mantenedores de la lista anterior. Con un bucle vamos recorriendo cada BBDD y almacenamos en arrays los resultados de los paquetes de cada maintainer.

```
"SELECT DISTINCT p.nombre FROM package p INNER JOIN maintainer m ON "  
"p.maintainer_id = m.id WHERE m.nombre = %s"
```

Una vez ejecutada dicha query, se muestran unos ejemplos de los resultados en la tabla.

Podemos observar que, en los tres ejemplos expuestos en la tabla, hay paquetes que se repiten varias veces en un mismo mantenedor. Esto se debe a que, como son maintainers que llevan desde el inicio de Debian, tienen asociados una serie de paquetes que mantendrán a lo largo de las próximas versiones a menos que abandonen o deje de interesar su mantenimiento.

Este resultado son unos ejemplos de lo que devuelve el script a la hora de ejecutarlo. Entendemos que carece de sentido mostrar perfiles individuales debido a que el estudio se basa en un proyecto grupal pero era la única forma de mostrar lo realizado.

5.5. ¿Qué sucede con los paquetes mantenidos por los mantenedores que abandonan el proyecto?

Hemos visto en la sección 5.4 cuantos mantenedores permanecen activos y cuales han sido sus aportaciones a lo largo de la evolución de Debian. Pero, ¿qué ocurre con los paquetes asociados a un maintainer que abandona el proyecto? Debe existir un método de control u organización para estos casos en los que no se puede dejar un paquete sin maintainer. Para ello, cogemos a uno de los maintainers que no aparezcan en la sección anterior y realizamos un seguimiento del paquete para ver que ocurre con él.

Cogemos como ejemplo a el maintainer **Martin Albisetti** y ejecutamos una query que devuelve los paquetes mantenidos por **Martin Albisetti** junto con la versión en la que se encuentra.

Releases	Packages	
7.lenny_main	2vcard	exifprobe
8.squeeze_main	2vcard	exifprobe
9.wheezy_main	2vcard	exifprobe
10.jessie_main	2vcard	

Como podemos ver en la tabla, este maintainer tiene asociados dos paquetes en los releases **Lenny**, **Squeeze** y **Wheezy**, en el release **Jessie** solo mantiene uno y abandona el proyecto para releases posteriores.

Ahora vamos a realizar un seguimiento a dichos paquetes en todos los releases obteniendo quien es la persona o equipo que los mantiene ejecutando otra query.

En esta tabla se puede observar que, a partir de la versión **Jessie**, el paquete **exifprobe** comienza a mantenerse por un grupo de Debian. En el release **Stretch** y **Buster** el paquete **2vcard** es mantenido por **Riley Baird** mientras que **exifprobe** sigue mantenido por el grupo anterior. En **Buster** cambia el paquete **exifprobe** de mantenedor a otro grupo de seguridad de Debian. Y finalmente se asignan los dos paquetes a grupos Debian.

Podemos ver los paquetes **huérfanos** se asocian a otros maintainers o a grupos para que trabajen en ellos.

También hay paquetes que dejan de usarse de una versión a otra aunque el maintainer no

Release	Package	Maintainer
7.lenny_main	2vcard	Martin Albisetti
7.lenny_main	exifprobe	Martin Albisetti
8.squeeze_main	2vcard	Martin Albisetti
8.squeeze_main	exifprobe	Martin Albisetti
9.wheezy_main	2vcard	Martin Albisetti
9.wheezy_main	exifprobe	Martin Albisetti
10.jessie_main	2vcard	Martin Albisetti
10.jessie_main	exifprobe	Debian Forensics
11.stretch_main	2vcard	Riley Baird
11.stretch_main	exifprobe	Debian Forensics
12.buster_main	2vcard	Riley Baird
12.buster_main	exifprobe	Debian Security Tools
13.bullseye_main	2vcard	Debian QA Group
13.bullseye_main	exifprobe	Debian Security Tools

haya abandonado el proyecto.

Observando otros ejemplos vemos una clara tendencia a la mantención de dichos paquetes por grupos Debian a partir de la versión Buster.

Como podemos ver en la **Figura 5.5**, a partir de la versión Buster hay un claro aumento en el número de equipos Debian, de 210 a 289, por lo que es lógico pensar que se han distribuido estos paquetes entre los diferentes grupos.

También se refleja en la **Figura 5.1** que en cada versión hay un aumento de los mantenedores hasta la versión Buster en la que observamos una tendencia de bajada en el número de los mismos.

Por ello podemos pensar que, hasta la versión Buster, se asignaban a maintainers pero a partir de dicha release comenzaron a organizarse en equipos y repartieron dichos paquetes entre los mismos.

5.6. ¿Los paquetes más importantes y de uso común son mantenidos por mantenedores más experimentados?

Para resolver esta pregunta debemos conocer cuales son los paquetes más importantes y usados de Debian. Entramos en la página de Debian Popcon [1], accedemos a las estadísticas de **la suma de paquetes ordenados**. Tras esto nos fijamos en el campo **vote** que indica el **número de personas que usan este paquete regularmente**. Estos paquetes son los que consideramos **importantes** como se muestra en la tabla.

#rank	name	inst	vote	old	recent	no-files
1	Not in sid	19992113	2857277	4663481	395618	12075737
2	util-linux	2890959	1641090	692861	339533	217115
3	systemd	1998462	1587252	149416	169750	92044
4	gnupg2	2248600	1580116	356679	178782	133023
5	libxcb	2578180	1237197	334256	201889	804838
6	dbus	1199936	953625	123986	120765	1560
7	krb5	1237409	858656	68086	69262	241405
8	gcc-14	1911702	788522	537396	164995	420789
9	libblockdev	904164	705667	263	104876	93358
10	libreoffice	2935606	670532	1480947	507072	277055
11	pam	955649	668089	207763	79222	575

Cuadro 5.6: Paquetes Popcon más usados ordenados por vote.

El paquete **Not in sid** no aparece en ninguna versión o release de Debian. Debian tiene varias ramas de desarrollo:

- **Unstable o Sid:** se desarrollan activamente los paquetes de Debian. Pueden tener errores y no se garantiza su estabilidad. Es usada principalmente por desarrolladores y usuarios avanzados para probar las últimas versiones.
- **Testing:** se prueban los paquetes que probablemente formarán parte de la próxima versión estable de Debian.

- **Stable:** contiene paquetes que han sido probados extensivamente y se consideran estables para el uso diario.

Este paquete **Not in sid** no está disponible en la rama unstable de Debian. Esto indica que el paquete no está en la fase de desarrollo activo o que no cumple con los estándares para ser incluido en la rama unstable.

Tras encontrar los nombres de los paquetes más importantes, diseñamos una query para obtener que maintainers son los que mantienen estos paquetes. Y al ejecutarla estos son los resultados:

Cuadro 5.7: Información de paquetes encontrados

Paquete	Base de datos	Maintainer
util-linux	1.hamm_main	Guy Maor
	2.slink_main	Vincent Renardias
	3.potato_main	Vincent Renardias
	4.woody_main	LaMont Jones
	5.sarge_main	LaMont Jones
	6.etch_main	LaMont Jones
	7.lenny_main	LaMont Jones
	8.squeeze_main	LaMont Jones
	9.wheezy_main	LaMont Jones
	10.jessie_main	Debian util-linux Maintainers
	11.stretch_main	Debian util-linux Maintainers
	12.buster_main	LaMont Jones
	13.bullseye_main	util-linux packagers
gnupg2	5.sarge_main	Matthias Urlichs
	6.etch_main	Eric Dorland
	7.lenny_main	Eric Dorland
	8.squeeze_main	Eric Dorland
	9.wheezy_main	Eric Dorland
	10.jessie_main	Debian GnuPG Maintainers

5.6. ¿LOS PAQUETES MÁS IMPORTANTES Y DE USO COMÚN SON MANTENIDOS POR MANTENEDORES?

Cuadro 5.7 – continuación de la página anterior

Paquete	Base de datos	Maintainer
	11.stretch_main 12.buster_main 13.bullseye_main	Debian GnuPG Maintainers Debian GnuPG Maintainers Debian GnuPG Maintainers
systemd	9.wheezy_main 10.jessie_main 11.stretch_main 12.buster_main 13.bullseye_main	Tollef Fog Heen Debian systemd Maintainers Debian systemd Maintainers Debian systemd Maintainers Debian systemd Maintainers
libxcb	7.lenny_main 8.squeeze_main 9.wheezy_main 10.jessie_main 11.stretch_main 12.buster_main 13.bullseye_main	XCB Developers XCB Developers XCB Developers XCB Developers Debian X Strike Force Debian X Strike Force Debian X Strike Force
dbus	5.sarge_main 6.etch_main 7.lenny_main 8.squeeze_main 9.wheezy_main 10.jessie_main 11.stretch_main 12.buster_main 13.bullseye_main	D-Bus Maintenance Team Utopia Maintenance Team Utopia Maintenance Team Utopia Maintenance Team Utopia Maintenance Team Utopia Maintenance Team Utopia Maintenance Team Utopia Maintenance Team Utopia Maintenance Team
krb5	4.woody_main 5.sarge_main 6.etch_main 7.lenny_main 8.squeeze_main	Sam Hartman Sam Hartman Sam Hartman Sam Hartman Sam Hartman

Cuadro 5.7 – continuación de la página anterior

Paquete	Base de datos	Maintainer
	9.wheezy_main	Sam Hartman
	10.jessie_main	Sam Hartman
	11.stretch_main	Sam Hartman
	12.buster_main	Sam Hartman
	13.bullseye_main	Sam Hartman
libblockdev	12.buster_main	Utopia Maintenance Team
	13.bullseye_main	Utopia Maintenance Team
libreoffice	9.wheezy_main	Debian LibreOffice Maintainers
	10.jessie_main	Debian LibreOffice Maintainers
	11.stretch_main	Debian LibreOffice Maintainers
	12.buster_main	Debian LibreOffice Maintainers
	13.bullseye_main	Debian LibreOffice Maintainers
avahi	6.etch_main	Utopia Maintenance Team
	7.lenny_main	Utopia Maintenance Team
	8.squeeze_main	Utopia Maintenance Team
	9.wheezy_main	Utopia Maintenance Team
	10.jessie_main	Utopia Maintenance Team
	11.stretch_main	Utopia Maintenance Team
	12.buster_main	Utopia Maintenance Team
	13.bullseye_main	Utopia Maintenance Team

Tras estos resultados, en los que se muestran los releases donde se encuentran estos paquetes y los maintainers asociados a los mismos, creamos una query que coja todos estos maintainers (de forma única) y compruebe si son expertos o no basándonos en la **vida media**.

La **vida media** de un maintainer es la suma de el número de releases en los que se encuentran cada maintainer dividido entre el número total de maintainers únicos.

Esto lo calculamos con otra query y nos devuelve como resultado que el número de apariciones de los maintainers es **21,409**, el número total de maintainers es **6,052** y la media de apariciones de un maintainer es **3.54**.

5.6. ¿LOS PAQUETES MÁS IMPORTANTES Y DE USO COMÚN SON MANTENIDOS POR MANTENEDORES EXPERTOS?

A partir de esta vida media podemos conocer si los maintainers que están asociados a los paquetes más importantes son **expertos** o no.

Comparamos las apariciones de cada uno de los maintainers anteriores con la vida media y diseñamos una query que devuelva si es experto. Estos son los resultados:

- El maintainer **Guy Maor** es un experto con presencia en 4 bases de datos (superando la media total).
- El maintainer **Vincent Renardias** es un experto con presencia en 4 bases de datos (superando la media total).
- El maintainer **LaMont Jones** es un experto con presencia en 11 bases de datos (superando la media total).
- El maintainer **Tollef Fog Heen** es un experto con presencia en 11 bases de datos (superando la media total).
- El maintainer **Eric Dorland** es un experto con presencia en 10 bases de datos (superando la media total).
- El maintainer **Sam Hartman** es un experto con presencia en 10 bases de datos (superando la media total).
- El maintainer **Matthias Urlich** es un experto con presencia en 9 bases de datos (superando la media total).
- El maintainer **Debian X Strike Force** es un experto con presencia en 9 bases de datos (superando la media total).
- El maintainer **Utopia Maintenance Team** es un experto con presencia en 8 bases de datos (superando la media total).
- El maintainer **XCB Developers** es un experto con presencia en 5 bases de datos (superando la media total).
- El maintainer **Debian LibreOffice Maintainers** es un experto con presencia en 5 bases de datos (superando la media total).

- El maintainer **Debian systemd Maintainers** es un experto con presencia en 4 bases de datos (superando la media total).
- El maintainer **Debian GnuPG Maintainers** es un experto con presencia en 4 bases de datos (superando la media total).
- El maintainer **D-Bus Maintenance Team** NO es un experto con presencia en 1 base de datos.
- El maintainer **util-linux packagers** NO es un experto con presencia en 1 base de datos.

Como se observa en los resultados, hay 13 maintainers expertos y 2 no expertos en un total de 15 maintainers. Los que definimos como no expertos son grupos recientes formados por Debian y esta puede ser la explicación de porque no llevan tantas versiones activos.

Capítulo 6

Conclusiones

En esta sección comentaremos cuales fueron los resultados del estudio realizado hace veinte años visto en la subsección 1.1.3 comparados con los obtenidos en este proyecto. De esta forma, podremos aportar las conclusiones adecuadas.

6.1. Conclusiones a las cuestiones planteadas

Como vimos en el capítulo de resultados 5 obtuvimos una serie de respuestas a las preguntas planteadas y, en consecuencia, aportaremos las conclusiones necesarias.

6.1.1. Conclusiones: ¿Cuántos mantenedores tiene Debian y cómo cambia este número con el tiempo?

En el **estudio anterior [10]** llegaron a la conclusión de que había un aumento en el número de maintainers de casi un **35 %** en cada release. También un aumento, a mayor escala, en el número de paquetes de cada release. Por ello pensaron que podía deberse a una mayor eficiencia de los maintainers por el desarrollo de nuevas tecnologías o en las prácticas de desarrollo.

Los resultados obtenidos en este proyecto difieren en ciertos aspectos del estudio anterior.

- Los porcentajes de aumento o disminución de maintainers entre los diferentes releases vistos en la **Figura 5.1** son:

1. Subida del 41 %.

2. Subida del 46 %.
3. Subida del 86 %.
4. Subida del 51 %.
5. Subida del 18 %.
6. Subida del 11 %.
7. Subida del 10 %.
8. Subida del 5 %.
9. Subida del 7 %.
10. Bajada del 2 %.
11. Se mantiene con un ligero aumento de 2 maintainers.
12. Bajada del 5 %.

La tendencia ha sido ascendente hasta la versión **10 Jessie** en la cual empezó a mantenerse e incluso a descender en número. Esto puede deberse a la creación de grupos los cuales se organizaban y distribuían el trabajo entre sí. Esta conclusión está respaldada por la **Figura 5.2** en la que se muestra un gran número de paquetes según evoluciona Debian. Por ello, no tendría sentido la disminución de los maintainer con el aumento de los paquetes para que siga siendo una distribución estable.

6.1.2. Conclusiones: ¿Existe una tendencia hacia la formación de equipos de mantenedores?

En el **estudio anterior [10]** se refleja que ha habido un claro aumento en el número de paquetes que mantiene un maintainer y que esto se debe a que se ha obtenido una mayor conciencia sobre la imprevisibilidad de los voluntarios formando equipos organizados los cuales mantengan dichos paquetes.

Los resultados de este proyecto cuadran con el estudio. Como se observa en la **Figura 5.5** hay un aumento exponencial en la creación de equipos. En el release 10 Jessie se observa un elevado aumento de 210 a 289 que coincide con la disminución de los maintainers y con el aumento del número de paquetes. Con ello concluimos que hay una clara tendencia a la formación de grupos y esto resuelve la pregunta anterior 6.1.1.

6.1.3. Conclusiones: ¿Cuántos mantenedores de versiones anteriores permanecen activos?

En el **estudio anterior [10]** se refleja que hay un claro descenso de los maintainers desde la primera versión que se mantienen en adelante. En el estudio de las primeras cuatro versiones se observa que ha bajado la aportación de los mismos y solo quedan (desde la primera versión) un 55 % de los mismos. Se intentó calcular la vida media de los maintainers, es decir, cuando su número baja hasta la mitad y se pronosticó que se daría al rededor de 7.5 años.

Las causas pueden ser por agotamiento pero estudios sobre recursos humanos dictan que se motivarían proyectos de software libre que ampliarían la vida media de los maintainers.

En este proyecto obtuvimos dos tablas en las que se observa la disminución de aportaciones de los maintainers y grupos a lo largo de las diferentes versiones.

Por ello se calculó la vida media de cada uno que consta de 3.5 releases. Calculando en cuantos años se lanza un release obtenemos que de media es cada 2.16 años. Si vemos el tiempo en años en el que se cumple la vida media de un maintainer se obtiene que son 7.5 años por lo que la conclusión que se obtuvo en el estudio [10] coincide veinte años después.

6.1.4. Conclusiones: ¿Cuál es el aporte de los mantenedores que permanecen en versiones posteriores?

En el **estudio anterior [10]** se refleja que, aunque el pensamiento inicial era que los mantenedores expertos tendrían más carga de trabajo, tienen números similares a maintainers más nuevos.

En este proyecto consideramos maintainers activos a los que llevan desde su primera versión hasta día de hoy en Debian. Como pudimos ver en los resultados, cada maintainer tiene asociado un gran conjunto de paquetes distribuidas entre todos los releases en los que se les asignaron.

Si nos fijamos en este dato podemos ver que estos maintainers mantienen muchos más paquetes que el resto pero hay que entender que esto es debido a que llevan desde el inicio.

Si los comparamos dentro de la misma versión, encontraremos resultados similares entre todos los maintainers por lo que la conclusión final coincide con la del estudio [10].

6.1.5. Conclusiones: ¿Cuál es el aporte de los mantenedores que permanecen en versiones posteriores?

En el **estudio anterior** [10] se refleja que cuando los mantenedores abandonan un proyecto, sus paquetes quedan huérfanos.

Más del 60 % de estos paquetes son adoptados en todas las versiones estudiadas, mostrando un proceso de regeneración natural en Debian. Sin embargo, el número de paquetes huérfanos crece más rápidamente que el de los adoptados, lo que sugiere que muchos paquetes no se mantienen en futuras versiones.

Además, es importante establecer mecanismos para evitar la introducción de paquetes insostenibles a largo plazo. Los proyectos con más desarrolladores tienen mayores probabilidades de mantenerse activos.

En este proyecto realizamos el seguimiento a un paquete de un maintainer específico que abandonó el proyecto. Pudimos observar que, los dos paquetes que tenía asociados, fueron adoptados por otros maintainers o grupos por lo que se mantienen en la actualidad. También entendimos que, al formarse una gran cantidad de grupos Debian, se están empezando a adoptar por dichos grupos los cuales se distribuyen organizadamente y con sus estándares los diferentes paquetes.

6.1.6. Conclusiones: ¿Los paquetes más importantes y de uso común son mantenidos por mantenedores más experimentados?

En el **estudio anterior** [10], se usaron datos del Concurso de Popularidad de Debian para investigar si los paquetes más importantes son mantenidos por voluntarios más experimentados.

Al analizar el número de instalaciones y uso regular por mantenedor, concluyen que los voluntarios más experimentados mantienen los paquetes más frecuentemente instalados y utilizados. Esto es evidente en todas las versiones de Debian, excepto en la 2.2, que tiene valores más altos que la 2.1, indicando una actividad inusual de los mantenedores de esa época.

Esto también sugiere que muchos componentes esenciales de Debian se introdujeron en las primeras versiones, mientras que los nuevos paquetes son mayoritariamente complementos y software menos utilizados.

En este proyecto volvimos a coger los datos de los paquetes del Concurso de Popularidad de

Debian. Diseñamos una query que devolviera que mantenedor o mantenedores tenían asignados los diez paquetes más usados y comparamos si eran expertos (comparando su vida con la vida media de 3.5 releases obtenida anteriormente).

Esto nos aportó un resultado de 13 expertos y 2 no expertos sobre un total de 15. Con esto podemos concluir que los paquetes más importantes de dicha distribución están asignados a los expertos.

6.2. Consecución de objetivos

En este proyecto se ha cumplido el objetivo general basado en poder responder a las preguntas planteadas en el estudio de Debian. Esto ha sido posible gracias a los diferentes objetivos específicos planteados:

- **Diseñar el diagrama entidad relación.** Este objetivo fue posible gracias al estudio de los diferentes paquetes y comprender las relaciones entre los atributos de los mismos. Este objetivo era el más importante ya que de él dependen los objetivos posteriores. Se encontraron ciertas dificultades para el diseño del diagrama debido a que esta formado por una entidad principal que acoge todas las demás. Hubo que adaptarlo para que en un folio se pudiera representar el mismo junto con sus tablas SQL. .
- **Crear la base de datos.** Este objetivo se separó en dos fases:
 1. **Crear las BBDD de cada release.** Se realizó de forma manual en el programa MySQL Work Bench.
 2. **Crear las tablas SQL** Se realizó en PyCharm mediante un script. Fue algo tedioso debido a tener que analizar las diferentes relaciones entre entidades y añadir de forma correcta sus Primary Key y Foreign Key. También hubo tablas que no se podían relacionar entre sí y se tuvieron que diseñar tablas intermedias.
- **Inserción de los datos.** Se realizó en Pycharm mediante un script. Este fue el objetivo con mayor dificultad debido a problemas al insertar dichos datos y al volumen de los mismos. Una vez realizado de forma correcta fue esencial para tener los datos estructurados y organizados de tal forma que se puedan consultar fácil y rápidamente.

- **Formular las Queries.** Se realizó en Pycharm mediante un script. Tuvimos que mezclar lenguaje Python con SQL para poder diseñar bien las queries y que se ejecutaran para todas las BBDD mediante bucles. El diseño fue lo más problemático debido a que hay que entender correctamente lo que se quiere resolver para poder aplicar un algoritmo que funcione.
- **Crear las tablas y gráficas.** Se realizó en Pycharm mediante un script. Una vez obtenidos los resultados con las queries, se usan los mismos para plasmarlos en gráficas o tablas. La mayor dificultad encontrada en este objetivo fue la inserción de un eje x temporal de forma que las gráficas quedaran espaciadas según el tiempo en el que se lanzó cada release.

6.3. Aplicación de lo aprendido

Este proyecto no hubiera sido posible sin los conocimientos obtenidos en diferentes asignaturas del Grado en Ingeniería de Tecnologías de Telecomunicación.

Las más relacionadas con este proyecto son:

- **Ingeniería de Sistemas de Información.** Esta asignatura me aportó los conocimientos necesarios sobre diagramas entidad - relación y lenguaje SQL para poder realizar las queries y como relacionar las tablas de las BBDD.
- **Sistemas Operativos.** Aunque el lenguaje usado fuera C y se basara en un lenguaje de programación de bajo nivel, esta asignatura me dio la base definitiva para entender la lógica de como programar y como pensar para realizar algoritmos correctamente.
- **Prácticas externas.** Esta asignatura la realicé en diferentes empresas y me ayudó a comprender lo que es trabajar de una forma más seria y ha sido fundamental a la hora de organizarme y saber priorizar el trabajo.
- **Servicios y Aplicaciones Telemáticas.** Esta asignatura me aportó el conocimiento en Python necesario para poder programar eficientemente en este lenguaje. También fue una de las piezas clave que me ayudó a comprender la programación y, con ello, a desarrollarme en la misma.

- **Fundamentos de la programación.** Esta asignatura me aportó el conocimiento base sobre qué era la programación y como había que pensar para obtener resultados en este ámbito.

6.4. Lecciones aprendidas

Lo que he podido aprender en este proyecto es:

1. Qué es Debian y cómo funciona internamente mediante estos releases.
2. El trabajo que hay detrás de una distribución de software libre.
3. Como diseñar de forma eficiente una BBDD, desde el parseo de los datos hasta el diseño de queries pasando por la inserción de los mismo en las BBDD.
4. La evolución de Debian a lo largo de sus trece versiones.
5. Como, hasta una de las distribuciones más importantes de Linux de software libre, acaba organizándose en grupos y siguiendo unos estándares para poder realizar el trabajo de forma eficiente.

6.5. Trabajos futuros

Para trabajos futuros se podrían:

1. Analizar los datos de este estudio y ver las tendencias que siguen pasados unos cuantos años y así entender en profundidad las fases que tiene Debian y su evolución con una mayor cantidad de datos.
2. Investigar otras distribuciones de Linux u otros sistemas operativos para comparar como funcionan, cuales son las más eficientes y por qué.
3. Estudiar datos como Binary, Files, Check-Sum256, etc, los cuales se encuentran en los paquetes y no hemos usado para este proyecto. Quizas se puedan sacar más conclusiones y entender mejor el funcionamiento y las tendencias de Debian.

Bibliografía

- [1] B. Allombert. Debian popularity contest, 2024.
- [2] L. M. Báez. Todo lo que debes saber de conjuntos en python — `set(...)`, 2019.
- [3] G. Castillo. Mysql workbench: Qué es, descarga, instalación y uso, 2023.
- [4] D. R. Center. ¿qué es python?, 2022.
- [5] Datascientest. Matplotlib: todo lo que tienes que saber sobre la librería python de dataviz, 2022.
- [6] Datascientest. Pycharm : Todo sobre el ide de python más popular, 2022.
- [7] Debian. Versiones de debian, 2023.
- [8] Debian. Versiones de debian archivadas, 2023.
- [9] GeoTalleres. Conceptos básicos de sql, 2022.
- [10] J. M. G.-B. Gregorio Robles and M. Michlmayr. Evolution of volunteer participation in libre software projects: Evidence from debian.
- [11] E. libro de python. 03. tipos y estructuras/set.
- [12] Lucidchart. ¿qué es un modelo entidad relación?, 2022.
- [13] G. Moreno. Los lenguajes de programación más usados del mundo, 2019.
- [14] M. SQL. Chapter 1 introduction to mysql connector/python, 2024.
- [15] TheDataSchool. Qué es una foreign key en sql y para qué se utiliza, 2022.
- [16] TheDataSchool. Sql primary key: Qué es y cómo utilizarla, 2022.