Universidad
Rey Juan Carlos

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

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

II

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

Aquí viene un resumen del proyecto. Ha de constar de tres o cuatro párrafos, donde se presente de manera clara y concisa de qué va el proyecto. Han de quedar respondidas las siguientes preguntas:

- ¿De qué va este proyecto? ¿Cuál es su objetivo principal?
- ¿Cómo se ha realizado? ¿Qué tecnologías están involucradas?
- ¿En qué contexto se ha realizado el proyecto? ¿Es un proyecto dentro de un marco general?

Lo mejor es escribir el resumen al final.

Summary

Here comes a translation of the “Resumen” into English. Please, double check it for correct grammar and spelling. As it is the translation of the “Resumen”, which is supposed to be written at the end, this as well should be filled out just before submitting.

Índice general

Índice de figuras

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 está conformado por varias versiones desde 1993 las cuales explicaremos a continuación:

- **Versiones 0.x (1993 - 1995):** estas versiones fueron las primeras y mas 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 comenzo 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 peliculas 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.

1.2. 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 lineas 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, de cada lanzamiento (**release**), la evolución en el número de personas y equipos que lo mantienen, cuantos de ellos siguen en releases posteriores y que ocurre con los paquetes de un mantenedor si este abandona el proyecto. Se ha podido llevar acabo gracias al estudio de diferentes paquetes de datos aportados por Debian[1] 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:

- **Diseño del 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. .
- **Creación de 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

[1] `https://www.debian.org/releases/`

que se requiere.

- **Formulación de Queries.** Se analiza la información que se quiere extraer de la base de datos y se crean las llamadas necesarias para obtenerla.

- **Creación de 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.

Dr. Gregorio Robles 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.

Nombre de actividad	Fecha inicio	Duración en días	Fecha fin
Instalación y aprendizaje de las tecnologías a usar	23-nov	3	26-nov
Analisis de paquetes y creación de diagrama entidad-relación	27-nov	7	04-dic
Creación de las bases de datos junto a sus tablas sql	05-dic	2	07-dic
Parseo de la información de todos los paquetes	08-dic	35	12-ene
Inserción de los paquetes en las bases de datos	14-dic	62	14-feb
Formulación de querys sql	15-feb	413	03-abr
Creación de tablas y gráficas	07-mar	27	03-abr
Redacción de la memoria	04-abr		

Figura 2.1: Diagrama de Gantt

Capítulo 3

Estado del arte

3.1. Python 3.12.0

Python [2] 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 [9].

[1] `https://platform.openai.com/docs/libraries/python-library`

Figura 3.1: Lenguajes de programación más populares del mundo

3.2. My sql connector

Esta es la librería más importante de Python para poder desarrollar este proyecto. MySQL Connector/Python [10] 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 es [3] 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 lineas de código.

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

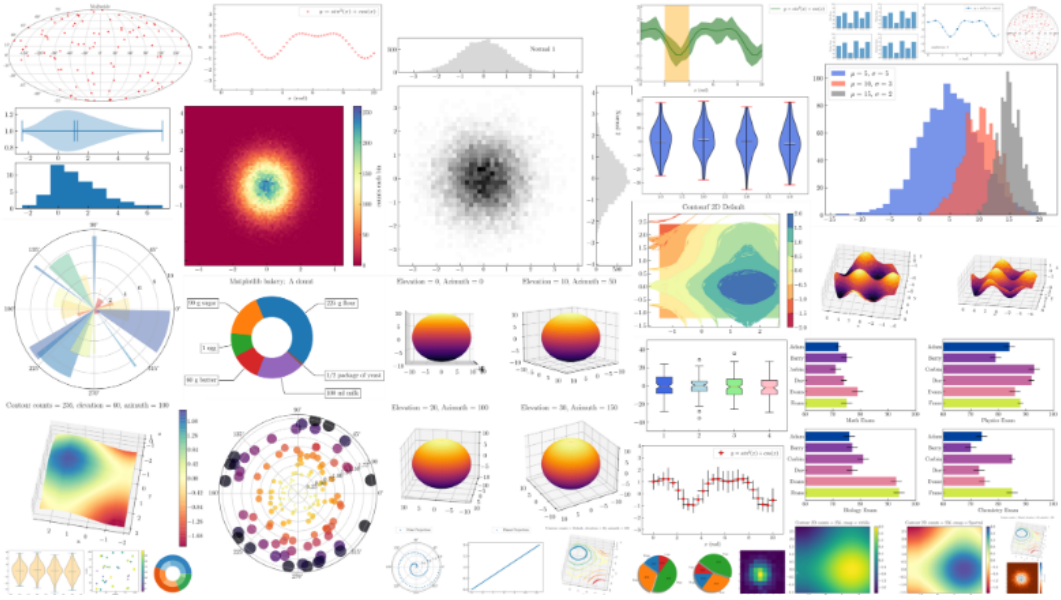

Figura 3.2: Ejemplos gráficas Matplotlib

3.4. Pycharm

PyCharm es el **IDE** [4] 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 programación 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 [1].

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.

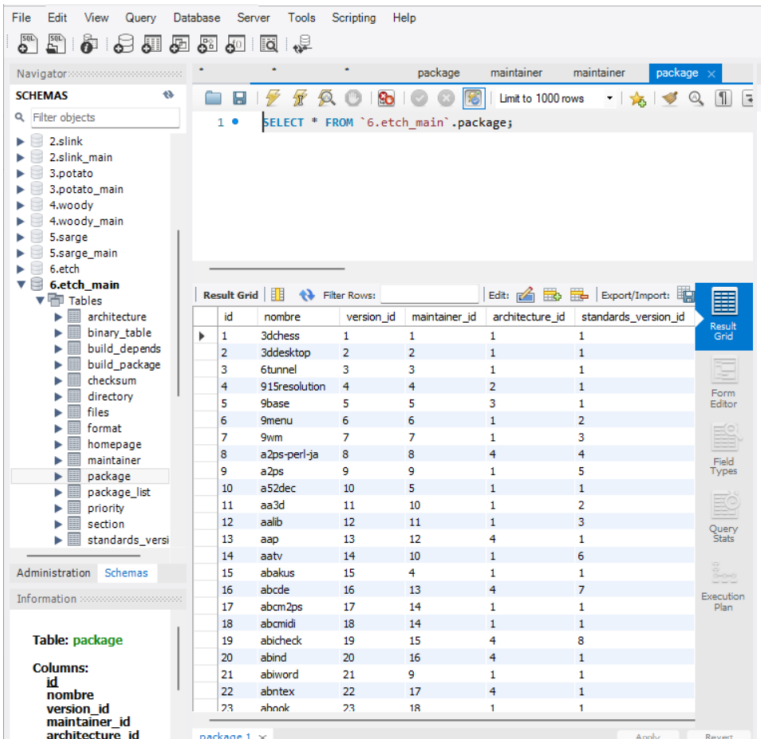

Figura 3.4: SQL Workbench

Capítulo 4

Diseño e implementación

4.1. Arquitectura general

El objetivo 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:

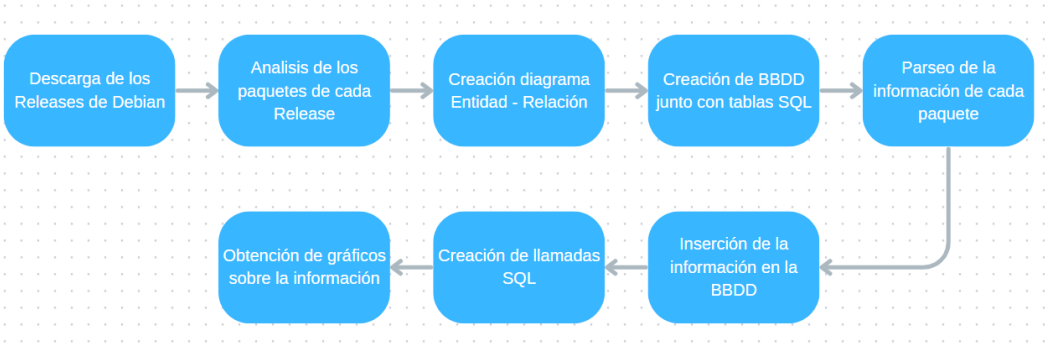

Figura 4.1: Fases del tratamiento de datos

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 [**?**]. 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. Conociendo las páginas donde encontrar los releases retirados [6] y los que siguen en uso [5] (aportados por mi tutor Gregorio), podemos descargar sus diferentes archivos **main** donde encontramos todos los paquetes de cada lanzamiento y con ello los datos que queremos tratar.

Figura 4.2: Descarga de los datos de releases

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 4.3**.

```
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
 9d93e6681a5c85923fafcecca1a8e63c1c4ab9bd74956e45b6b10e5d9e1018cd 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 4.3: 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 1 y es único.
- **Binary**: mínimo hay 1 o más.
- **Version**: siempre hay 1 y puede repetirse en varios paquetes.
- **Maintainer**: siempre hay 1 y puede repetirse en varios paquetes.
- **Uploaders**: mínimo hay 1 o más y puede repetirse en varios paquetes.
- **Build-Depends**: mínimo hay 1 o más y puede repetirse en varios paquetes.

- **Architecture**: siempre hay 1 y puede repetirse en varios paquetes.
- **Standards-Version**: siempre hay 1 y puede repetirse en varios paquetes.
- **Format**: siempre hay 1 y puede repetirse en varios paquetes.
- **Files**: mínimo hay 1 o más y son únicos.
- **Vcs-Browser**: (no está en todos los paquetes) siempre hay 1 y es único.
- **Vcs-Git**: (no está en todos los paquetes) siempre hay 1 y es único.
- **Checksums-Sha256**: mínimo hay 1 o más y son únicos.
- **Homepage**: (no está en todos los paquetes) siempre hay 1 y puede repetirse en varios paquetes.
- **Package-List**: mínimo hay 1 o más y son únicos.
- **Directory**: siempre hay 1 y es único.
- **Dgit**: (no está en todos los paquetes) siempre hay 1 y es único.
- **Testsuite**: (no está en todos los paquetes) siempre hay 1 y puede repetirse en varios paquetes.
- **Testsuite-Triggers**: (no está en todos los paquetes) mínimo hay 1 o más y puede repetirse en varios paquetes.
- **Priority**: siempre hay 1 y puede repetirse en varios paquetes.
- **Section**: siempre hay 1 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 [8]**, 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 4.4 se observa el diseño del diagrama junto con la Figura 4.5 que es su leyenda.

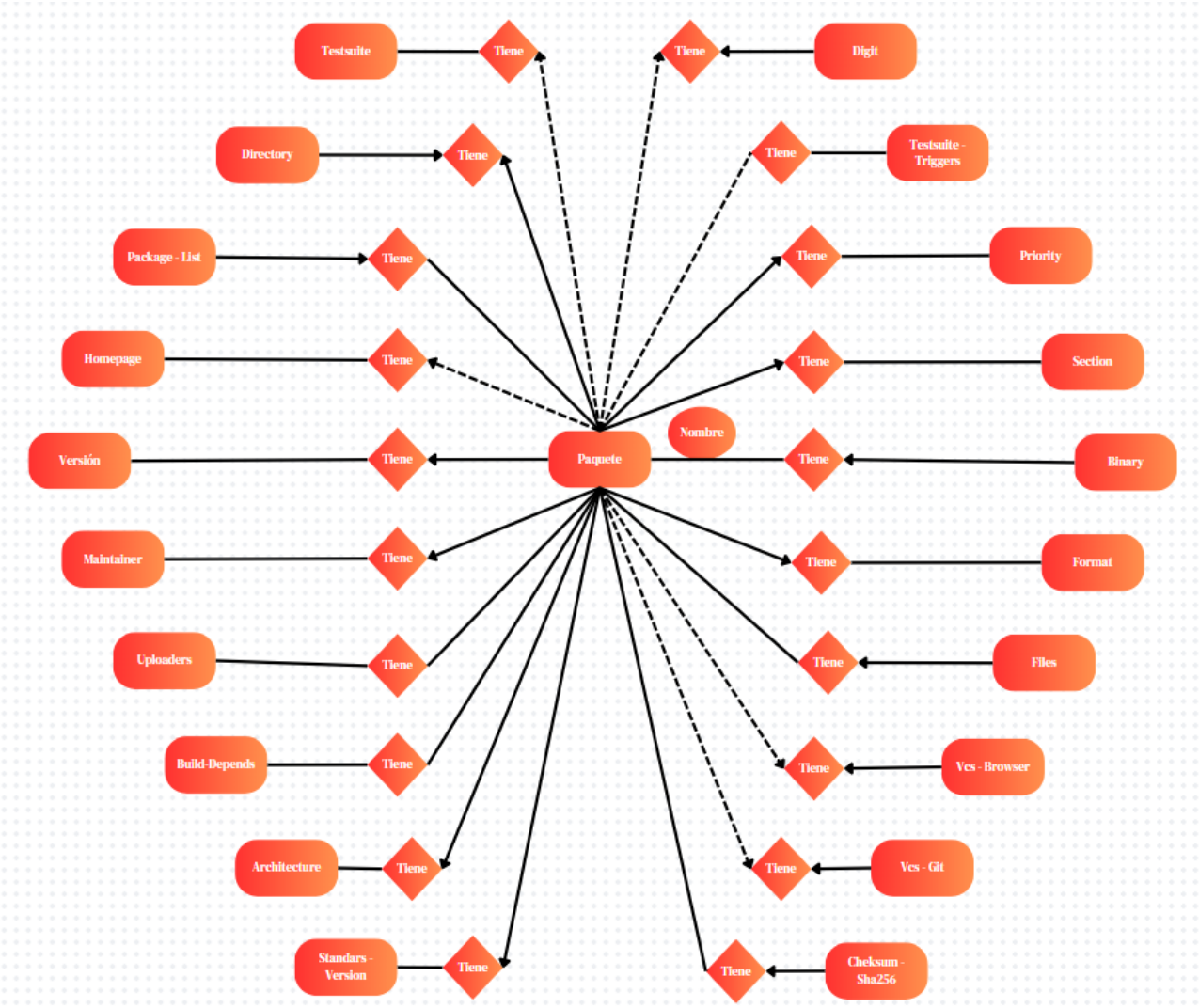

Figura 4.4: Diagrama ERD.

Cada rectángulo es una entidad que va unida a otras entidades mediante una serie de líneas y el rombo que las separa es el indicador 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.

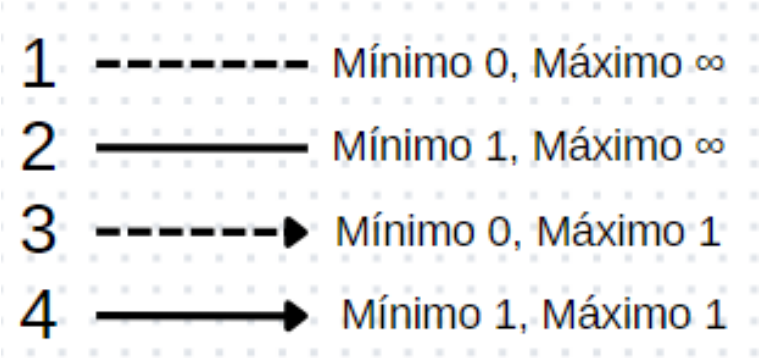

Figura 4.5: Leyenda de diagrama ERD.

En la **Figura 4.5** podemos ver el tipo de relación entre entidades:

- **1:** esta línea discontinua muestra una relación de **0 a infinito**, es decir, pueden existir ninguna, una o muchas entidades en el paquete.
- **2:** esta línea continua muestra una relación de **1 a infinito**, es decir, pueden existir una o muchas entidades en el paquete pero nunca puede no existir.
- **3:** esta línea discontinua terminada en flecha muestra una relación de **0 a 1**, es decir, pueden existir ninguna o una entidad en el paquete.
- **4:** esta línea continua terminada en flecha muestra una relación de **1 a 1**, es decir, solo puede existir una entidad en el paquete obligatoriamente.

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

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 como se muestra en la **Figura 4.6**.

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

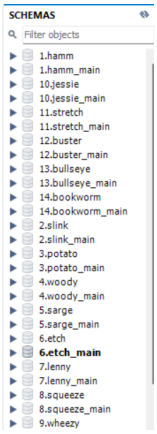

Figura 4.6: Bases de datos

- **Entidad 1 (flecha hacia) Entidad 2**: cuando esto ocurre necesitamos crear una **foreing key** para relacionar las tablas de cada entidad. **Entidad 1** debe tener una foreing 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)
)
"""
```

  Una **FOREIGN KEY** [11] 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** [**?**] 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 (linea) Entidad 2**: en este caso no tenemos claro que entidad debría acoger la foreing key por lo que se crea una tabla intermedia. Esta tabla constará de los identifica-

dores de cada tabla como sus primary key. En ella es en la que se definirán las foreing 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.

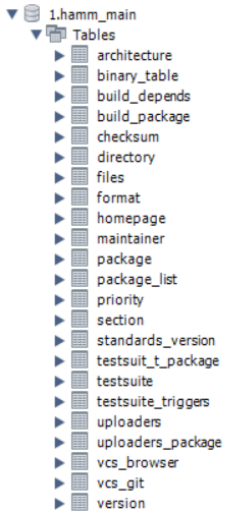

Figura 4.7: Tablas BBDD

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 linea por linea todo el fichero e ir guardando la información relevante de cada paquete.

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

2. **Especificar que 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', 'Uploa
'Standards-Version', 'Format', 'Files', 'Vcs-Browser', 'Vcs-Git', '
'Package-List', 'Directory', 'Priority', 'Section', 'Testsuite', 'T
```

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, bui
standards_version, format1, files_list, vcs_browser, vcs_git, chec
package_list, directory, priority, section, testsuite, testsuite_t
```

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.

```
      # Restablecer las listas y variables para el próximo paquete
files_list = []
package_list = []
check_list = {}
package = ""
binary = ""
version = ""
```

```
maintainer = ""
uploaders = ""
build_depends = ""
architecture = ""
standards_version = ""
format1 = ""
vcs_browser = ""
vcs_git = ""
homepage = ""
directory = ""
priority = ""
section = ""
testsuite = ""
testsuite_triggers = ""
```

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.

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,
 "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 4.8: 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 linea del paquete empieza por estos 2 casos debemos cambiar la lógica. Saltamos a las siguientes lineas guardando todos los datos en un array. Posteriormente se recorrerá y se insertarán en las diferentes tablas.

```
Files:
 266b8669362d98c0eef2a3fe83c1b29a 1924 1oom_1.0-2.dsc
 3001abc0dcd309815de06fd24277a325 533439 1oom_1.0.orig.tar.gz
 98832941cbf539d134971d55ca97a081 4028 1oom_1.0-2.debian.tar.xz
```

Figura 4.9: Files

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

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 especificas a la misma. De esta forma obtendremos la información que se necesita.

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 [7].** 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**.
- **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 mantainers 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", "Junio200
"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.

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:

```
Número total de maintainers en 1.hamm_main: 255

Número total de maintainers en 2.slink_main: 362
```

```
Número total de maintainers en 3.potato_main: 530

Número total de maintainers en 4.woody_main: 986

Número total de maintainers en 5.sarge_main: 1490

Número total de maintainers en 6.etch_main: 1760

Número total de maintainers en 7.lenny_main: 1971

Número total de maintainers en 8.squeeze_main: 2173

Número total de maintainers en 9.wheezy_main: 2286

Número total de maintainers en 10.jessie_main: 2468

Número total de maintainers en 11.stretch_main: 2412

Número total de maintainers en 12.buster_main: 2418

Número total de maintainers en 13.bullseye_main: 2298
```

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 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).

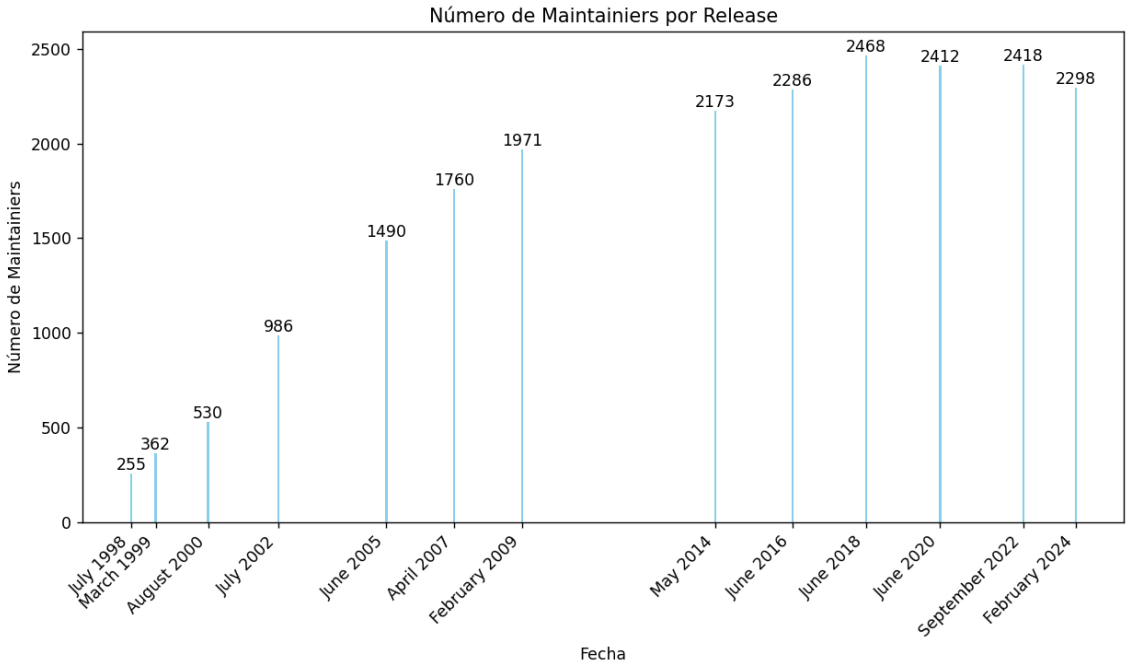

Figura 5.1: Número de maintainers de cada release

- 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.

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 optimo 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 5.2 y 5.1**

```
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
 69127335cecda88837fed160bb1b23a6 2505 bibindex_2.8-1.diff.gz
```

Figura 5.2: Ejemplo de nombre de equipo de maintainers

```
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 5.3: Ejemplo de nombre de equipo de maintainers

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

Una vez ejecutado estos son los resultados:

```
Número total de equipos en 1.hamm_main: 1

Número total de equipos en 2.slink_main: 2

Número total de equipos en 3.potato_main: 8

Número total de equipos en 4.woody_main: 8

Número total de equipos en 5.sarge_main: 31

Número total de equipos en 6.etch_main: 79

Número total de equipos en 7.lenny_main: 118

Número total de equipos en 8.squeeze_main: 147

Número total de equipos en 9.wheezy_main: 168

Número total de equipos en 10.jessie_main: 196

Número total de equipos en 11.stretch_main: 210

Número total de equipos en 12.buster_main: 289

Número total de equipos en 13.bullseye_main: 317
```

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

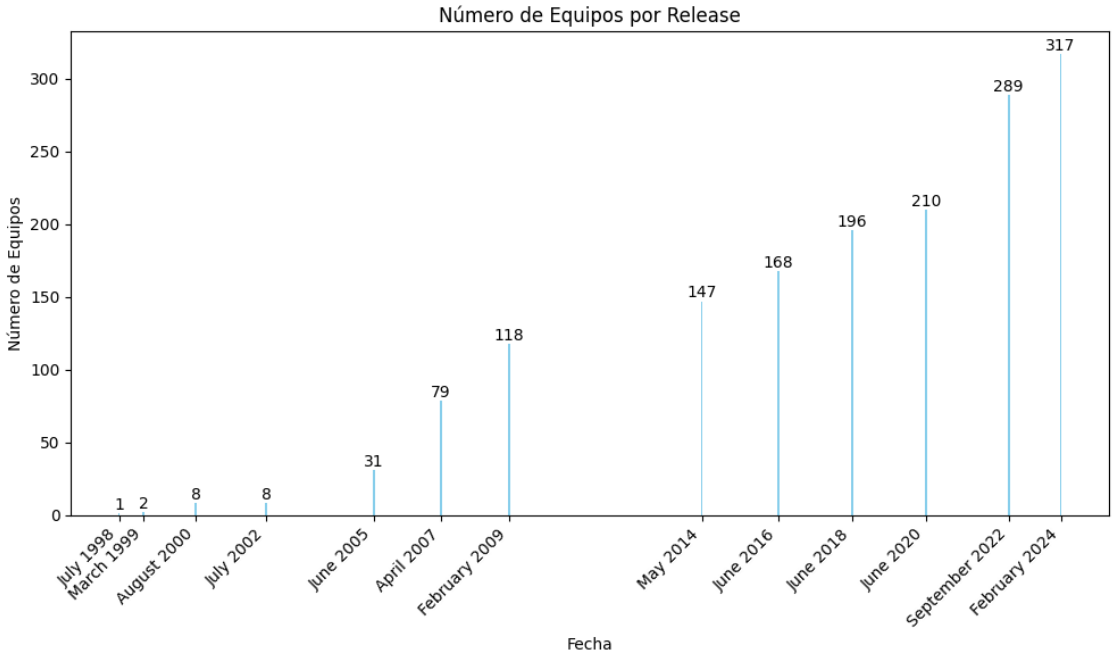

Figura 5.4: Equipos formados en cada release

Podemos apreciar observando la **Figura 5.4** 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.

- **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 las 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.

Primero lanzamos una query que nos muestra cuales son los mantenedores que se encuentran en todos los releases de Debian. Estos maintainers son los veteranos que llevan desde el comienzo.

Una vez lanzada este es el resultado:

```
Maintainers comunes en todas las bases de datos:

['Jaldhar H. Vyas', 'Hakan Ardo','Fredrik Hallenberg', 'Joel Rosdahl',
  'Dirk Eddelbuettel', 'Norbert Veber',
  'Bdale Garbee <bdale@gag.com>', 'Roberto Lumbreras <rover@debian.org>'
  'John Goerzen <jgoerzen@complete.org>', 'Paul Slootman <paul@debian.or
  'Craig Sanders <cas@taz.net.au>', 'Manoj Srivastava <srivasta@debian.o
  'Michael Meskes <meskes@debian.org>', 'Martin Schulze <joey@debian.org
  'Anselm Lingnau <lingnau@debian.org>', 'Martin Buck <mbuck@debian.org>
  'Anthony Fok <foka@debian.org>', 'Miquel van Smoorenburg <miquels@cist
  'Yann Dirson <dirson@debian.org>', 'Matthias Klose <doko@debian.org>',
  'Juan Cespedes <cespedes@debian.org>', 'Philip Hands <phil@hands.com>'
 'Davide G. M. Salvetti <salve@debian.org>', 'Craig Small <csmall@debian
 "Marco d'Itri <md@linux.it>", 'Riku Voipio <riku.voipio@iki.fi>',
 'Roderick Schertler <roderick@argon.org>']
```

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

```
Porcentajes de maintainers en la base de datos 1.hamm_main:
Con respecto a la base de datos 2.slink_main: 94.74%
Con respecto a la base de datos 3.potato_main: 82.46%
Con respecto a la base de datos 4.woody_main: 60.96%
Con respecto a la base de datos 5.sarge_main: 47.37%
Con respecto a la base de datos 6.etch_main: 38.60%
Con respecto a la base de datos 7.lenny_main: 32.46%
Con respecto a la base de datos 8.squeeze_main: 28.07%
Con respecto a la base de datos 9.wheezy_main: 27.19%
Con respecto a la base de datos 10.jessie_main: 25.44%
Con respecto a la base de datos 11.stretch_main: 21.05%
Con respecto a la base de datos 12.buster_main: 18.86%
Con respecto a la base de datos 13.bullseye_main: 17.98%

Porcentajes de maintainers en la base de datos 2.slink_main:
Con respecto a la base de datos 3.potato_main: 86.67%
Con respecto a la base de datos 4.woody_main: 66.67%
Con respecto a la base de datos 5.sarge_main: 51.43%
Con respecto a la base de datos 6.etch_main: 42.22%
Con respecto a la base de datos 7.lenny_main: 33.97%
Con respecto a la base de datos 8.squeeze_main: 29.21%
Con respecto a la base de datos 9.wheezy_main: 26.35%
Con respecto a la base de datos 10.jessie_main: 24.13%
Con respecto a la base de datos 11.stretch_main: 20.32%
Con respecto a la base de datos 12.buster_main: 18.41%
Con respecto a la base de datos 13.bullseye_main: 17.78%

Porcentajes de maintainers en la base de datos 3.potato_main:
Con respecto a la base de datos 4.woody_main: 77.68%
Con respecto a la base de datos 5.sarge_main: 56.87%
Con respecto a la base de datos 6.etch_main: 45.92%
```

```
Con respecto a la base de datos 7.lenny_main: 37.34%
Con respecto a la base de datos 8.squeeze_main: 31.97%
Con respecto a la base de datos 9.wheezy_main: 28.54%
Con respecto a la base de datos 10.jessie_main: 26.39%
Con respecto a la base de datos 11.stretch_main: 23.39%
Con respecto a la base de datos 12.buster_main: 20.60%
Con respecto a la base de datos 13.bullseye_main: 19.10%

Porcentajes de maintainers en la base de datos 4.woody_main:
Con respecto a la base de datos 5.sarge_main: 73.58%
Con respecto a la base de datos 6.etch_main: 58.93%
Con respecto a la base de datos 7.lenny_main: 49.39%
Con respecto a la base de datos 8.squeeze_main: 41.51%
Con respecto a la base de datos 9.wheezy_main: 36.40%
Con respecto a la base de datos 10.jessie_main: 33.85%
Con respecto a la base de datos 11.stretch_main: 29.63%
Con respecto a la base de datos 12.buster_main: 27.19%
Con respecto a la base de datos 13.bullseye_main: 23.97%

Porcentajes de maintainers en la base de datos 5.sarge_main:
Con respecto a la base de datos 6.etch_main: 82.92%
Con respecto a la base de datos 7.lenny_main: 69.69%
Con respecto a la base de datos 8.squeeze_main: 57.41%
Con respecto a la base de datos 9.wheezy_main: 48.91%
Con respecto a la base de datos 10.jessie_main: 44.84%
Con respecto a la base de datos 11.stretch_main: 38.44%
Con respecto a la base de datos 12.buster_main: 34.81%
Con respecto a la base de datos 13.bullseye_main: 30.67%

Porcentajes de maintainers en la base de datos 6.etch_main:
Con respecto a la base de datos 7.lenny_main: 84.26%
```

```
Con respecto a la base de datos 8.squeeze_main: 69.97%
Con respecto a la base de datos 9.wheezy_main: 59.94%
Con respecto a la base de datos 10.jessie_main: 53.92%
Con respecto a la base de datos 11.stretch_main: 45.77%
Con respecto a la base de datos 12.buster_main: 40.50%
Con respecto a la base de datos 13.bullseye_main: 35.67%

Porcentajes de maintainers en la base de datos 7.lenny_main:
Con respecto a la base de datos 8.squeeze_main: 83.57%
Con respecto a la base de datos 9.wheezy_main: 71.95%
Con respecto a la base de datos 10.jessie_main: 64.22%
Con respecto a la base de datos 11.stretch_main: 54.55%
Con respecto a la base de datos 12.buster_main: 47.80%
Con respecto a la base de datos 13.bullseye_main: 42.07%

Porcentajes de maintainers en la base de datos 8.squeeze_main:
Con respecto a la base de datos 9.wheezy_main: 85.59%
Con respecto a la base de datos 10.jessie_main: 76.48%
Con respecto a la base de datos 11.stretch_main: 64.88%
Con respecto a la base de datos 12.buster_main: 56.20%
Con respecto a la base de datos 13.bullseye_main: 48.78%

Porcentajes de maintainers en la base de datos 9.wheezy_main:
Con respecto a la base de datos 10.jessie_main: 88.84%
Con respecto a la base de datos 11.stretch_main: 74.89%
Con respecto a la base de datos 12.buster_main: 64.43%
Con respecto a la base de datos 13.bullseye_main: 54.88%

Porcentajes de maintainers en la base de datos 10.jessie_main:
Con respecto a la base de datos 11.stretch_main: 84.44%
Con respecto a la base de datos 12.buster_main: 72.15%
```

	1.hamm_main	2.slink_main	3.potato_main	4.woody_main	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
1.hamm_main	100	94.74	82.46	60.96	47.37	38.60	32.46	28.07	27.19	25.44	21.05	18.86	17.98
2.slink_main	-	100	86.67	66.67	51.43	42.22	33.97	29.21	26.35	24.13	20.32	18.41	17.78
3.potato_main	-	-	100	77.68	56.87	45.92	37.34	31.97	28.54	26.39	23.39	20.60	19.10
4.woody_main	-	-	-	100	73.58	58.93	49.39	41.51	36.40	33.85	29.63	27.19	23.97
5.sarge_main	-	-	-	-	100	82.92	69.69	57.41	48.91	44.84	38.44	34.81	30.67
6.etch_main	-	-	-	-	-	100	84.26	69.97	59.94	53.92	45.77	40.50	35.67
7.lenny_main	-	-	-	-	-	-	100	83.57	71.95	64.22	54.55	47.80	42.07
8.squeeze_main	-	-	-	-	-	-	-	100	85.59	76.48	64.88	56.20	48.78
9.wheezy_main	-	-	-	-	-	-	-	-	100	88.84	74.89	64.43	54.88
10.jessie_main	-	-	-	-	-	-	-	-	-	100	84.44	72.15	61.26
11.stretch_main	-	-	-	-	-	-	-	-	-	-	100	85.02	72.45
12.buster_main	-	-	-	-	-	-	-	-	-	-	-	100	85.41
13.bullseye_main	-	-	-	-	-	-	-	-	-	-	-	-	100

Figura 5.5: Porcentaje de maintainers que siguen en releases posteriores

```
Con respecto a la base de datos 13.bullseye_main: 61.26%


Porcentajes de maintainers en la base de datos 11.stretch_main:
Con respecto a la base de datos 12.buster_main: 85.02%
Con respecto a la base de datos 13.bullseye_main: 72.45%


Porcentajes de maintainers en la base de datos 12.buster_main:
Con respecto a la base de datos 13.bullseye_main: 85.41%
```

Una vez obtenida la información diseñamos una tabla para visualizar los datos.

En la Figura 5.5 podemos ver cual es el porcentaje 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 porcentaje de maintainers que aguantan en versiones posteriores.

Capítulo 6

Conclusiones

6.1. Consecución de objetivos

Esta sección es la sección espejo de las dos primeras del capítulo de objetivos, donde se planteaba el objetivo general y se elaboraban los específicos.

Es aquí donde hay que debatir qué se ha conseguido y qué no. Cuando algo no se ha conseguido, se ha de justificar, en términos de qué problemas se han encontrado y qué medidas se han tomado para mitigar esos problemas.

Y si has llegado hasta aquí, siempre es bueno pasarle el corrector ortográfico, que las erratas quedan fatal en la memoria final. Para eso, en Linux tenemos aspell, que se ejecuta de la siguiente manera desde la línea de *shell*:

```
aspell --lang=es_ES -c memoria.tex
```

6.2. Aplicación de lo aprendido

Aquí viene lo que has aprendido durante el Grado/Máster y que has aplicado en el TFG/TFM. Una buena idea es poner las asignaturas más relacionadas y comentar en un párrafo los conocimientos y habilidades puestos en práctica.

1. a

2. b

6.3. Lecciones aprendidas

Aquí viene lo que has aprendido en el Trabajo Fin de Grado/Máster.

1. Aquí viene uno.
2. Aquí viene otro.

6.4. Trabajos futuros

Ningún proyecto ni software se termina, así que aquí vienen ideas y funcionalidades que estaría bien tener implementadas en el futuro.

Es un apartado que sirve para dar ideas de cara a futuros TFGs/TFMs.

Apéndice A

Manual de usuario

Esto es un apéndice. Si has creado una aplicación, siempre viene bien tener un manual de usuario. Pues ponlo aquí.

Bibliografía

[1] G. Castillo. Mysql workbench: Qué es, descarga, instalación y uso, 2023.

[2] D. R. Center. ¿qué es python?, 2022.

[3] Datascientest. Matplotlib: todo lo que tienes que saber sobre la librería python de dataviz, 2022.

[4] Datascientest. Pycharm : Todo sobre el ide de python más popular, 2022.

[5] Debian. Versiones de debian, 2023.

[6] Debian. Versiones de debian archivadas, 2023.

[7] GeoTalleres. Conceptos básicos de sql, 2022.

[8] Lucidchart. ¿qué es un modelo entidad relación?, 2022.

[9] G. Moreno. Los lenguajes de programación más usados del mundo, 2019.

[10] M. SQL. Chapter 1 introduction to mysql connector/python, 2024.

[11] TheDataSchool. Qué es una foreign key en sql y para qué se utiliza, 2022.