Proyecto de Grado

Presentado ante la ilustre Universidad de Los Andes como requisito parcial para obtener el Título de Ingeniero de Sistemas

Ciclos de vida del software libre. Caso de estudio Distribución Canaima GNU/Linux

Elaborado por

David A. Hernández Aponte

Tutor: Dr. Jacinto Dávila

Noviembre 2017

Ciclos de vida del software libre. Caso de estudio Distribución Canaima GNU/Linux

David A. Hernández Aponte

Proyecto de Grado — Sistemas Computacionales, 79 páginas

Resumen: Se realizó un estudio en el que se describe el ciclo de vida del desarrollo de software libre aplicado a una distribución Linux, usando como objeto de investigación la distribución Canaima GNU/Linux. Para ello se muestra la manera en que se ha llevado a cabo su desarrollo través de la historia de la distribución, y se hace un cotejo con otras distribuciones, como estrategia comparativa para establecer casos de éxito y/o fracaso en las diferentes fases de desarrollo de la misma.

Palabras clave: Linux, Software libre, Distribución Linux, Ciclo de vida, Canaima GNU/Linux, Desarrollo colaborativo, Debian, Ubuntu, COCOMO, SLOC.

Este trabajo fue procesado en LaTeX.

A mi madre.

Agradecimiento

A quienes corresponda (por definir).

Índice general

Índice de tablas

Índice de figuras

Quien hace, puede equivocarse. Quien nada hace, ya está equivocado.

Daniel Kon.

Capítulo 1

Introducción y Antecedentes

Introducción

Este proyecto tiene como objetivo evaluar la productividad asociada o asociable a una distribución de software libre o distro. Para ello, se revisa el concepto de ciclo de vida del software, y se usa como una aproximación a la correspondiente noción de ciclo de vida de toda una distribución, un concepto claramente más complejo y difícil de abordar. Algunos indicadores complementarios son convocados para esta tarea y se los justifica a partir de esfuerzos previos en el área como los que se describen a continuación.

Antecedentes

La Organización Internacional para la Normalización (*International Organization for Standardization*, ISO en inglés) presenta en 2008 la última modificación hasta la fecha de la norma para los procesos de vida del software ISO/IEC 12207, publicada por primera vez en 1995. Aquí se define *ciclo de vida* como la evolución de un sistema, producto, servicio, proyecto u otra entidad concebida por humanos, desde su concepción hasta su retiro (International Organization for Standardization, 2008).

También se explica el *modelo de ciclo de vida* como la descripción de procesos y actividades relacionadas con el ciclo de vida que pueden estar organizadas en etapas, las cuales también actúan como referencia común para su comunicación y entendimiento (International Organization for Standardization, 2008).

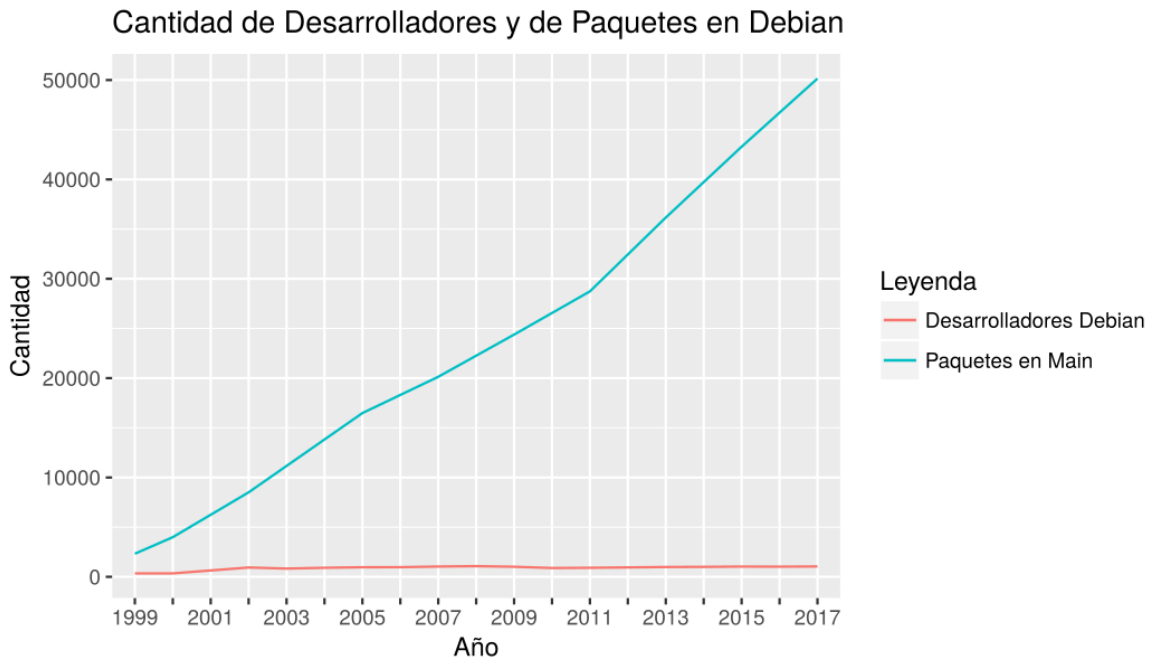
El Institute of Electrical and Electronics Engineers (1997), en su norma para la creación de procesos de software, define *ciclo de vida del software* como la secuencia de actividades específicas a un proyecto que son creadas asignando las actividades de esta norma a un modelo de ciclo de vida de software seleccionado. A su vez, un *modelo de ciclo de vida del software* es determinado como el marco de trabajo, seleccionado por cada organización, en el cuál se asignan actividades a esta norma para producir el ciclo de vida del software (Institute of Electrical and Electronics Engineers, 1997). Finalmente se define *procesos de ciclo de vida del software* como la descripción específica de los procesos de proyectos basados en el ciclo de vida del software y en los mecanismos que definen el entorno del proyecto dentro de la organización. (Institute of Electrical and Electronics Engineers, 1997).

Capiluppi y Michlmayr (2007) realizaron un estudio empírico del ciclo de vida en dos proyectos de software libre basados en comunidades de voluntarios para estudiar las fases catedral y bazar. Los autores parten de la hipótesis que estas dos fases no son mutuamente excluyentes, y afirman. “Los proyectos de software libre comienzan en una fase de catedral, y de manera potencial migran a una fase de bazar” (Capiluppi y Michlmayr, 2007, p.15). También identifican una fase de transición a la que se refieren como “fundamental para lograr un verdadero éxito y que el proyecto resulte popular” (Capiluppi y Michlmayr, 2007, p.16). Dice Raymond (1999) que, el enfoque estilo catedral en la programación, se distingue en proyectos de pocos desarrolladores y con largos periodos entre la liberación de las versiones, estos largos periodos se deben al tiempo que les toma corregir los errores. En el enfoque estilo bazar, al contrario, se liberan versiones con más frecuencia buscando llegar al mayor número de desarrolladores que puedan ayudar a ubicar errores y además colaborar en su corrección.

Por su parte, Amor et al. (2005a) realizaron una investigación en la que utilizan la versión 3.1 de la distribución Debian como caso de estudio de medición de software

libre. Allí se concluye que el crecimiento de Debian se incrementa año tras año, al igual que el número de voluntarios y el número de paquetes. Así, señalan a Debian como uno de los mayores sistemas de software del mundo, sino el más grande. Sin embargo, la sostenibilidad del proyecto es cuestionada al apuntar un comportamiento inestable en el tamaño medio del proyecto "probablemente debido a un crecimiento de los paquetes más rápido que el número de mantenedores de Debian" (Amor et al., 2005a, p.14). En la Figura 1.1 se advierte un crecimiento sostenido de forma lineal en la cantidad de paquetes dentro del repositorio *main* de la distribución Debian, mientras que la cantidad de Desarrolladores presenta un comportamiento con poca variabilidad a lo largo del tiempo.

Figura 1.1: Crecimiento en el número de Desarrolladores Debian y paquetes del repositorio principal entre los años 1999 y 2017



Fuentes: Wikipedia[1], Archivo Debian[2], Repositorio Debian[3] y Listado de Desarrolladores Debian[4]

[1] https://en.wikipedia.org/wiki/Debian
[2] http://archive.debian.org/
[3] http://ftp.debian.org/debian/
[4] https://nm.debian.org/public/people/dd_all

La técnica de medición de líneas de código fuente físicas (*Source Lines of Code* o SLOC en inglés) permite realizar una serie de cálculos para estimar el tiempo de desarrollo y esfuerzo de un proyecto aplicando el Modelo Constructivo de Costos (*Constructive Cost Model* – COCOMO) (Boehm, 1981).

Esta técnica "es una de las más simples y de las más ampliamente usadas para la comparación de piezas de software." (González et al., 2003, p.6). Ya en el pasado se ha utilizado este método por medio de la utilización de paquetes de software que facilitan el cálculo de las líneas de código como muestran los trabajos de Wheeler, González, Amor y otros autores (Wheeler, 2001; González et al., 2001, 2003; Amor et al., 2004, 2005a,b, 2007, 2009; Herraiz et al., 2006).

Planteamiento del Problema

En el software libre el modelo participativo es vital para el éxito y superviviencia de los proyectos. Para ello, cada proyecto define los protocolos para aceptar y facilitar la colaboración de terceros. Las contribuciones se pueden presentar, por ejemplo, proponiendo modificaciones de código en un repositorio con control de versiones, reportando un mal funcionamiento del software, proponiendo nuevas ideas para ser implementadas en un futuro, colaborando con la traducción a diferentes idiomas y otros.

La distribución Canaima GNU/Linux nace desde la necesidad de centrar esfuerzos para el desarrollo, mantenimiento y soporte de un sistema operativo libre, con estándares abiertos y para ser usado como una plataforma tecnológica común en las instituciones de la Administración Pública Nacional (APN). Adicionalmente, se busca disminuir el gasto público destinado a la compra de licencias de software privativo que poco aportan al plan nacional de desarrollo el cual, entre otras aspectos, impulsa el carácter libre del conocimiento y fortalece el modelo de trabajo colaborativo.

Sin embargo en la actualidad, es posible afirmar que la distribución Canaima GNU/Linux está siendo poco usada dentro de la APN y que no se están cumpliendo las expectativas que dieron origen a su creación. Se cree que la debilidad en la conducción del Proyecto Canaima sea el asunto medular que ofrece razón a

esta situación, tales como: planificación equívoca en la aplicación del proyecto; insuficiencia de personal el cual además no se encuentra a dedicación exclusiva del mismo y otros. Esto quiere decir que, existen pocas personas dedicadas a este proyecto cuyas dimensiones socio-técnicas requieren de un equipo amplio y multidisciplinario que garantice la concreción de los objetivos para los cuales fue concebida esta distribución.

La presente investigación está orientada a mostrar por una parte, la productividad del desarrollo de Canaima GNU/Linux comparándola con otras distribuciones de referencia; y por la otra, presentar la viabilidad que tiene la misma para recibir contribuciones por parte de la comunidad de desarrolladores; y con ello formular algunas recomendaciones para mejorar la situación descrita en torno a la distribución Canaima GNU/Linux.

Objetivos

Evaluar la productividad de una distro es, como se ha sugerido, un desafío complejo. En este proyecto nos concentramos en evaluar la productividad asociada al proyecto Canaima, un proyecto nacional que ha tenido el apoyo financiero estatal.

Objetivo General

- Evaluar la productividad de la metadistribución de Software Libre Canaima GNU/Linux.

Objetivos Específicos

1. Caracterizar los ciclos de vida del software libre, identificando indicadores cuantivativos y cualitativos de cada etapa.
2. Recolectar datos relevantes a esos indicadores para un conjunto de distribuciones Linux referenciales.
3. Comparar la distro Canaima GNU/Linux con esas distribuciones referenciales.

4. Analizar el estado actual del proyecto Canaima y explicar las razones para sus niveles actuales de productividad.

Para alcanzar los objetivos descritos, se han realizado una serie de actividades que incluyen la recolección de datos para cada distro objeto de referencia, la estimación de esfuerzo y costos empleados en el desarrollo de cada proyecto, y la caracterización del modelo de desarrollo del proyecto Canaima.

El documento está organizado en 5 capítulos:

- Capítulo 1: correspondiente a la introducción, antecedentes, planteamiento del problema y objetivos.
- Capítulo 2: se definen los conceptos en torno al ciclo de vida del software libre y la estimación de esfuerzo y costos así como la descripción de las técnicas utilizadas para determinar estas estimaciones.
- Capítulo 3: en este capítulo se muestran los datos referenciales comunes que sirven como puntos de comparación entre las distribuciones evaluadas.
- Capítulo 4: se realiza un recuento histórico de la distribución Canaima GNU/Linux y se hace la caracterización del modelo de desarrollo que permite describir su ciclo de vida.
- Capítulo 5: se plasman las conclusiones, observaciones y recomendaciones de la investigación.

Capítulo 2

Sobre el ciclo de vida del software libre y estimación de costos

---

La naturaleza de las distribuciones de software libre basadas en linux permite que se realicen diversos estudios sobre esos desarrollos. A partir de repositorios públicos, por cada versión liberada se pueden extraer una serie datos que sirven como insumo para alimentar bases de conocimientos que caracterizan estas distros; por ejemplo, determinar los lenguajes utilizados, cantidad de líneas de código, número de paquetes de software en una distribución, frecuencia de lanzamiento de nuevas versiones, etc., datos que a su vez permiten mostrar la evolución de las distribuciones en el tiempo.

Para poder describir el ciclo de vida de la distribución Canaima GNU/Linux se hace un recuento histórico del proyecto desde sus primeras versiones liberadas en 2007 hasta la fecha actual. Se determinan en este estudio los factores claves que forman parte de los procesos mas relevantes de un ciclo de vida para ubicar el modelo de desarrollo que hoy en día es aplicado en el proyecto Canaima.

Tener otras figuras con las cuales comparar la distribución nacional ayuda establecer algunas ideas acerca de la magnitud que conlleva un proyecto de software de tal envergadura. Por ello se toma como referentes dos de las distribuciones más

populares en el ambiente Linux: Debian y Ubuntu.

Las consideraciones tomadas para la selección de estas distribuciones radica principalmente en que parten de la misma rama. Debian es la distribución madre de Ubuntu y de Canaima, es decir, estas dos distribuciones son derivaciones directas de Debian, se basan en ella para su construcción y por ello comparten algunas características básicas: son distribuciones gratuitas, tienen soporte para los mismos sistemas de archivos[1], tienen un modelo de lanzamiento fijo, comparten el mismo sistema de gestión de paquetes (DEB) y tienen un sistema de repositorios equivalente.

Las diferencias principales se pueden apreciar en los modelos de gobernanza y en la gestión de los diferentes procesos del desarrollo de las distribuciones, y son estas diferencias las que servirán para descubrir el ciclo de vida de la distribución Canaima/GNU Linux.

Ciclo de Vida

"Un ciclo de vida para un proyecto se compone de fases sucesivas compuestas por tareas que se pueden planificar" (Instituto Nacional de Tecnologías de la Comunicación, 2009, p.24).

Como se dijo en el capítulo anterior, la vida de un sistema o de un producto de software puede ser modelado a través de un modelo de ciclo de vida basado en fases. Los modelos pueden ser usados para representar toda su vida, desde la concepción hasta su retiro, o la representación de una porción de vida correspondiente al proyecto actual. El modelo de ciclo de vida está comprendido por una secuencia de fases que se pueden solapar y/o iterar, según sea apropiado para el alcance, magnitud, complejidad, necesidades y oportunidades cambiantes del proyecto. Cada etapa es descrita como una declaración de propósitos y resultados. Los procesos y actividades del ciclo de vida son seleccionados y empleados en una etapa para cumplir el propósito y los resultados de esa fase (International Organization for Standardization, 2008).

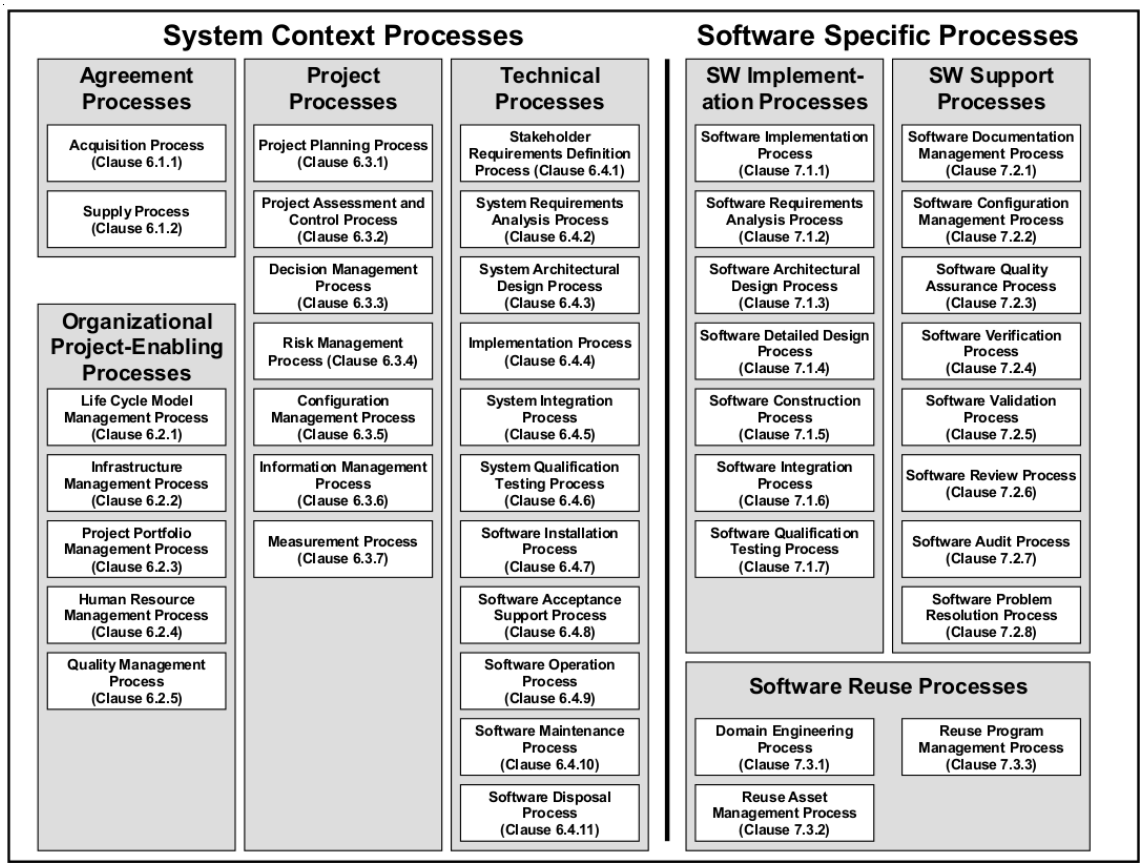
"El modelo ISO/IEC 12207 divide a sus procesos en dos grupos: procesos del

[1] Según la versión de Debian en las que estén basadas.

ciclo de vida de los sistemas en general y procesos específicos del ciclo de vida del sistema de software" (Rentería, 2012, p.86). Dentro de estos dos procesos principales las actividades que pueden ser ejecutadas durante el ciclo de vida de un sistema de software se agrupan en siete procesos. Cada uno de los procesos del ciclo de vida dentro de esos grupos es descrito en términos de su propósito y resultados deseados y lista las actividades y tareas que deben realizarse para alcanzar esos resultados (International Organization for Standardization, 2008).

Los siete procesos descritos en el estándar son: procesos de acuerdo; procesos de habilitación de proyectos organizacionales; procesos de proyecto; procesos técnicos; procesos de implementación de software; procesos de soporte de software y procesos de reutilización de software. La figura 2.1 muestra la categorización de los grupos y subgrupos de los procesos del ciclo de vida según el estándar de la ISO.

Figura 2.1: Grupos de los procesos del ciclo de vida



Fuente: International Organization for Standardization (2008)

El modelo descrito por la ISO no está vinculado con alguna metodología o modelo de ciclo de vida de software en particular. Por ejemplo, modelos de desarrollo tipo cascada, incremental, prototipo, espiral, etc. Pretende ser una referencia general que las organizaciones adaptarían a sus necesidades. En cambio, el modelo de referencia está destinado a ser adoptado por una organización basada en sus necesidades de negocio y dominio de aplicación. El proceso definido por la organización es adoptado a su vez por los proyectos de la organización en el contexto de los requisitos del cliente (International Organization for Standardization, 2008).

Procesos del software

Según Somerville (2005), "un proceso del software es un conjunto de actividades y resultados asociados que producen un producto de software" (p.8). A su vez, Pressman (2010) dice:

> El proceso de software forma la base para el control de la administración de proyectos de software, y establece el contexto en el que se aplican métodos técnicos, se generan productos del trabajo (modelos, documentos, datos, reportes, formatos, etc.), se establecen puntos de referencia, se asegura la calidad y se administra el cambio de manera apropiada. (p.12)

Por su parte Mardones (2008) puntualiza:

> Toda organización que lleve a cabo un proceso de software, debe trabajar con un modelo que se adapte a su marco de trabajo y ajustarlo a sus actividades específicas, a las personas que realizarán el proceso y al ambiente en el que se ejecutará el trabajo. (p.11)

También, Somerville (2005), hace una definición de un modelo de proceso del software, queda así:

> Un modelo de procesos del software es una descripción simplificada de un proceso del software que presenta una visión de ese proceso.

> Estos modelos pueden incluir actividades que son parte de los procesos y productos de software y el papel de las personas involucradas en la ingeniería del software. (p.8)

Canaima es descrito como un proyecto socio-tecnológico-productivo abierto[2]. Por ello, la tarea de describir su ciclo de vida va más allá del mero enfoque técnico. En otras palabras Somerville (2005) dice que, los sistemas socio-técnicos no sólo incluyen componentes de hardware y software sino que también incluyen personas, políticas y reglas organizacionales.

Estos factores serán explicados en el capítulo siguiente, como parte de la caracterización de los procesos que componen el desarrollo de la distribución Canaima y así dibujar el ciclo de vida, y el modelo de ciclo de vida, que mejor corresponda con las prácticas empleadas en este proyecto.

Estimación de costos

El Modelo Constructivo de Costos – COCOMO

> El modelo COCOMO es un modelo empírico que se obtuvo recopilando datos de varios proyectos grandes. Estos datos fueron analizados para descubrir las fórmulas que mejor se ajustaban a las observaciones. Estas fórmulas vinculan el tamaño del sistema y del producto, factores del proyecto y del equipo con el esfuerzo necesario para desarrollar el sistema.
> (Somerville, 2005, p.572)

En el modelo COCOMO, desarrollado por Boehm a inicios de la década de 1980, no sólo se consideró las características del producto sino también aquellas concernientes al equipo y ambiente de desarrollo. De este modo, Boehm caracteriza los desarrollos de software de acuerdo a su complejidad de desarrollo (Indian Institutes of Technology Kharagpur, 2006). Estas tres categorías son:

[2] ¿Qué es Canaima? http://canaima.softwarelibre.gob.ve/canaima/que-es-canaima Recuperado en marzo de 2017.

Orgánico: Proyectos pequeños de menos de 50 KSLOC, compuesto por equipos pequeños con experiencia en el desarrollo de proyectos semejantes, en ambientes estables y sin fuertes restricciones de tiempo.

Semiempotrado: Proyectos de mediana escala, entre 50 y 300 KSLOC, desarrollado por equipos con experiencia limitada y restricciones de tiempo moderadas.

Empotrado: Proyectos de gran escala, con mas de 300 KSLOC, desarrollados por equipos compuesto de personal que pudieran no tener mucha experiencia, posee fuertes restricciones de tiempo y procedimientos complejos.

Adicionalmente, sobre los modelos que operan dentro de COCOMO, Gómez et al. (2000) dicen:

> COCOMO' 81 está compuesto por tres modelos que corresponden a distintos niveles de detalle y precisión. Mencionados en orden creciente son: Modelo Básico, Intermedio y Detallado. La estimación es más precisa a medida que se toman en cuenta mayor cantidad de factores que influyen en el desarrollo de un producto de software. (p.6)

El modelo COCOMO básico utiliza el número de líneas de código fuente de los paquetes para estimar los recursos mínimos que se necesitan para construir el sistema (González et al., 2003). Pero debe tomarse en cuenta que, el modelo utilizado para esta estimación asume en cualquier caso un entorno de desarrollo privativo "clásico", por lo que debe considerarse con cierto cuidado (González et al., 2001). En cualquier caso, las estimaciones obtenidas de este modelo tienen la intención de darnos una idea de los costos de tiempo, esfuerzo y personal que tomarían los proyectos evaluados aplicando modelos de desarrollo privativo.

Para los cálculos efectuados en el estudio se asume un modelo orgánico. Se selecciona este modelo fundamentado que una distribución linux, es una colección de paquetes de software diseñados para interactuar en ambientes basados en el núcleo linux, organizados y configurados de tal forma que permiten hacer funcionar una computadora para poder efectuar una gran variedad de tareas. Cada paquete admitido en una distro es un proyecto, generalmente pequeño, compuesto por un

grupo de programadores con pericia en el desarrollo de aplicaciones similares. Cada proyecto tiene sus modos de disponer de los tiempos de entrega y procedimientos. Cada vez que se hace un lanzamiento de una nueva versión de una distribución, se selecciona la última versión estable de cada paquete, los mantenedores se encargan de velar porque los paquetes cumplan con todos los requisitos de calidad de cada distribución antes de ser admitidos en los repositorios.

Líneas de código fuente (SLOC)

Una línea física de código fuente (*Source Line of Code (SLOC)* en inglés) es aquella que termina en una línea nueva o con un demarcador de fin de archivo, y además contiene, por lo menos, un carácter diferente a un espacio en blanco o comentario (Wheeler, 2001). En otras palabras, toda línea que represente un comentario, esté vacía o esté compuesta íntegramente de espacios en blanco o tabulaciones no se consideran como líneas de código.

Determinar las estimaciones de costo de desarrollo siguiendo el modelo COCOMO supone conocer la cantidad de líneas de código fuente físicas. El proceso aplicado para obtener estos números implica hacerse del código fuente de los paquetes a evaluar.

Como se desea realizar comparaciones con términos comunes entre las tres distribuciones, se seleccionan los repositorios principales de cada distro, los repositorios "*main*" en Debian y en Ubuntu, y el repositorio "*usuarios*" de Canaima. Estos repositorios representan paquetes de software libre y abierto mantenidos enteramente por la comunidad de desarrolladores y mantenedores de cada distribución. Esto quiere decir que, aunque puede que haya paquetes "iguales" en los repositorios, los mantenedores se encargan de adaptar los programas para que cumplan con los estándares de publicación y calidad de cada distro.

Ya contando con los paquetes fuentes descomprimidos a evaluar se debe recorrer cada archivo en busca de las líneas de código, comentarios y líneas en blanco.

Para poder calcular el número de líneas de código fuente en el presente estudio se ha optado por utilizar la herramienta `cloc`[3]. `Cloc` es un programa con licencia

[3] https://github.com/AlDanial/cloc

libre, multiplataforma, hecho en Perl, que analiza los paquetes de software dados y procede a contar las líneas en blanco, líneas con comentarios y las líneas de código fuente físicas, reconociendo más de 200 lenguajes de programación.

Una vez obtenido el número de líneas de código fuente se efectúan los cálculos respectivos de las diferentes estimaciones, tales como: esfuerzo estimado de desarrollo, productividad, tiempo de desarrollo, cantidad de personas requeridas y costo total del proyecto.

En este trabajo se asume cada distribución como la suma de cada uno de los proyectos (paquetes) de software que lo componen. Dicho esto, se toma como la cantidad total de líneas de código fuente a la suma de las líneas de código fuente de cada paquete de software contenido en el repositorio principal de cada distribución.

Procedimientos para el cálculo de estimaciones

En resumen, para poder determinar las estimaciones de costos aplicando el modelo COCOMO básico en cada distribución, se realizan los siguientes operaciones:

1. Descargar el archivo `Sources` que contiene la lista de paquetes fuentes de la rama principal de la versión de la distro a evaluar.
2. Ejecutar el script de descarga de paquetes fuentes.
3. Ejectuar el script de descompresión y ordenamiento de paquetes fuente.
4. Correr el programa `cloc` para estimar el número de líneas de código fuente físicas.
5. Realizar las estimaciones de esfuerzo, tiempo de desarrollo, personal, productividad y costo total.

Scripts para descargar paquetes fuente

Para descargar el código fuente de cada paquete dentro del repositorio principal de una distribución basada en Debian se ha utilizado el comando `apt-get -download-only source nombredelpaquete`. Esta acción descarga un conjunto

de archivos que permiten compilar y construir nuevas versiones del paquete desde las fuentes.

Las extensiones asociadas a estos archivos posibilita escoger el archivo que interesa para hacer las mediciones. Así, el programa utilizado para automatizar la descarga de todos los archivos fuente se enfoca en conservar el archivo con el código fuente creado por el autor, con extensión `.orig.tar.gz`, y elimina el resto de archivos descargados que no son necesarios para el proceso de cálculo de líneas de código.

El listado de los paquetes de un repositorio se encuentra en un archivo llamado "*Sources*". Este archivo se ubica dentro del mismo repositorio público, en un directorio de nombre `source`, para cada rama del repositorio y para cada una de las versiones de la distribución.

El código para automatización de descarga de paquetes fuentes se aprecia en el Código 1 del Apéndice. Básicamente el programa lee cada línea del archivo *Sources*, el cual es el índice de archivos fuente que aloja el repositorio, y se encarga de identificar y extraer el nombre de cada paquete que se encuentra dentro del índice para luego proceder a la descarga en el disco duro del archivo comprimido que contiene el código fuente del nombre del paquete procesado.

Scripts para ordenar y descomprimir paquetes fuentes

Los archivos fuentes se descargan en un formato empaquetado (o comprimido). A modo de mantener un orden y para efectos de contabilizar cada directorio como un proyecto diferente, el script crea un directorio para cada uno de los paquetes, esto ayuda entre otras cosas a evitar sobreescritura de archivos en un directorio de descompresión común debido a nombres de archivos y directorios iguales, y organiza cada subproyecto (paquete de código) en un espacio propio.

Los directorios creados inician con el símbolo guión bajo (_). Esto se debe a que la orden utilizada para descomprimir los paquetes no permite la creación de un directorio con el mismo nombre que el archivo a desempaquetar. El conjunto de instrucciones utilizado para ejecutar todos estos pasos se aprecia en el Código 2 del Apéndice.

Cantidad de líneas de Código Fuente

Previo al uso de `cloc` se prepara el ambiente de trabajo donde será instalada una nueva instancia de la distribución a evaluar. Es decir, se instalará un nuevo sistema operativo dentro de un directorio del sistema operativo actual. De este modo, se dispone de un entorno operativo limpio, recién instalado, y sin riesgo de afectar el sistema operativo anfitrión. Los pasos para ello se describen a continuación tomando como ejemplo la distribución Debian:

Instalación de los paquetes necesarios para crear una jaula.

```
1 apt install debootstrap
```

Instalación de una jaula Debian Jessie (como root)

```
1 debootstrap jessie jaula http://localhost/debian/
```

Donde:

`deboostrap`: es el comando para construir la jaula.

`jessie`: es la distribución a instalar en la jaula.

`jaula`: es el nombre del directorio donde se creará la jaula.

`http://localhost/debian/`: es la dirección del repositorio local. Al utilizar un repositorio local se disminuye considerablemente el tiempo de descarga de los paquetes.

Se copia el script que se encargará de descargar los paquetes de la rama principal de la distribución hacia la jaula recién creada.

```
1 sudo cp paquetes.py jaula/home/
```

Se repite la operación con el programa que se encargará de crear directorios y descomprimir los archivos en estos directorios.

```
1 sudo cp mkdirmv.sh jaula/home/
```

Se entra a la jaula

```
1 sudo chroot jaula
```

En caso de ser necesario se modifica el `sources.list` para usar sólo el repositorio local en su rama principal y se actualiza.

```
nano /etc/apt/sources.list
apt update
```

Se instalan los paquetes necesarios para poder operar dentro de la jaula.

```
apt install xz-utils tar bzip2 python locales sloccount cloc
```

Se configura el lenguaje del entorno.

```
locale-gen es_VE es_VE.UTF-8
dpkg-reconfigure locales
```

Se descarga el paquete de fuentes desde el repositorio local y se descomprime:

```
wget http://localhost/debian/dists/jessie/main/source/Sources.gz
gzip -d Sources.gz
```

Se procede a correr el script de descarga

```
python paquetes.py
```

Luego se corre el programa que se encarga de ordenar cada archivo en un directorio propio, descomprimirlo y borrar el archivo comprimido.

```
./mkdirmv.sh
```

Se corre el programa de medidas de métricas[4]

```
cloc --use-sloccount --exclude-ext=po --csv --report-file=cloc.csv _*
```

Donde:

`cloc`: es el programa para determinar la cantidad de líneas de código.

`-use-sloccount`: es la orden que le indica a `cloc` usar los contadores compilados de `SLOCCount`[5] para los lenguajes C, java, pascal, php y xml que mejoran el desempeño en tiempo del programa.

`-exclude-ext=po`: argumento acompañado de una lista de extensiones que le indica al programa excluir todos los archivos de ese tipo en los cálculos.

`-csv`: es el argumento que indica al programa que el resultado se debe presentar con

[4] Asegúrese de haber borrado los programas `paquetes.py`, `mkdirmv.sh` y el archivo `Source` para no incluirlos en las métricas.

[5] https://www.dwheeler.com/sloccount/

un formato de valores separados por coma (csv).
`-report-file=cloc.csv`: es el argumento que ordena al programa almacenar los resultados en el archivo `cloc.csv`.
`_*`: ruta de los archivos o directorios que se analizarán. En este caso se analizan todos los archivos y directorios en el directorio actual que inician con guión bajo (_)

Caso Debian

Para la estimación de los valores en el caso del repositorio Debian se ha debido recurrir a una variación en el método de recolección de datos. Debido a que el programa `cloc` se quedaba sin responder en un punto del proceso, se procedió a evaluar por separado cada paquete, para luego compilar la información de los 20.498[6] reportes generados por `cloc`.

Para correr `cloc` en cada directorio por separado se utilizó un script que genera un archivo de reporte csv por cada paquete procesado satisfactoriamente. El código se muestra en el Código 3 del Apéndice.

Una vez con todos los reportes generados se procedió a evaluar esta información compilando los datos para generar un reporte global. Para ello también se ha creado un programa en python que hace uso de la biblioteca `pandas`[7] para el análisis de los datos. El código fuente de este programa se aprecia en el Código 4 del Apéndice.

Datos recolectados

Un ejemplo de los resultados arrojados por `cloc` para los paquetes `zygrib`, `zyn`, `zynaddsubfx` y `zypper` de la distribución Debian 8.0 se muestran en la tabla 2.1.

[6] De los 20.981 paquetes analizados, 7 paquetes causaban que el programa se quedara sin respuesta y otros 476 no generaron reporte.
[7] http://pandas.pydata.org/

Tabla 2.1: SLOC agrupados por lenguaje

Lenguaje	Cantidad de archivos	Líneas en blanco	Líneas con comentarios	Líneas de código fuente
C++	280	16.125	18,203	86.929
C/C++ Header	309	7.546	9,825	20.167
Python	67	2.758	1,799	10.622
C	14	708	424	3.888
make	7	402	32	3.399
CMake	29	221	208	1.016
IDL	3	47	0	439
Bourne Shell	5	27	27	219
Bourne Again Shell	2	29	8	202
XML	5	0	0	88
Perl	1	5	4	36

Estimación de valores

Tomando los valores bajo el modelo de estimación COCOMO Básico y aplicando los valores establecidos para el modo orgánico presentados en la Tabla 2.2, se realizan los cálculos de esfuerzo y tiempo de desarrollo mediante un script en lenguaje R (ver Código 5 del Apéndice) usando como entrada de datos el mismo archivo csv generado por el programa de medidas de métricas.

Ecuaciones

$$Epm = a \times (KSLOC)^b \tag{2.1}$$

$$Tdev = c \times (Epm)^d \tag{2.2}$$

$$Per = \frac{Epm}{Tdev} \tag{2.3}$$

$$Ctd = Per \times Spa \tag{2.4}$$

Donde:

Epm: es la estimación del esfuerzo de desarrollo, en persona-mes.

$KSLOC$: es el número de líneas de código fuente físicas, en miles.

$Tdev$: es la estimación del tiempo de desarrollo del proyecto, en meses.

Per: es el número de personas requeridas, en personas.

Ctd: es el costo total estimado de desarrollo del proyecto, en US$.

Spa: es el salario promedio anual estimado de programadores y analistas[8].

a, b, c y d: son los coeficientes según el tipo de proyecto, ver tabla 2.2.

Tabla 2.2: Constantes para el cálculo de distintos aspectos de costes para el modelo COCOMO básico

Tipo de proyecto	a	b	c	d
Orgánico	**2,40**	**1,05**	**2,50**	**0,38**
Medio	3,00	1,12	2,50	0,35
Embebido	3,60	1,20	2,50	0,32

Continuando con el ejemplo usando los paquetes `zygrib`, `zyn`, `zynaddsubfx` y `zypper` de la distribución Debian 8.0, la Tabla 2.3 presenta los cálculos de estimación de esfuerzo para estos paquetes.

[8]Se toma como referencia el valor US$60.445,00 para el mes de enero de 2017, dato obtenido desde http://www.payscale.com/research/US/Job=Computer_Programmer/Salary

Tabla 2.3: Estimaciones de esfuerzo y costos para los paquetes `zygrib`, `zyn`, `zynaddsubfx` y `zypper` de la distribución Debian 8.0 aplicando COCOMO Básico

Líneas de código fuente:	127.005
Esfuerzo estimado de desarrollo (persona–mes):	388,35
Tiempo de desarrollo estimado (meses):	24,09
Personas requeridas estimadas (personas):	16,12
Costo total estimado del proyecto (US$):	1.190.288,83

Para el Costo Total estimado se toma el valor US$ 73.837,00 anual como salario de referencia para ingenieros de software, desarrolladores y programadores en enero de 2017[9].

La calidad de estas estimaciones de productividad depende crucialmente de la obtención de datos de procesos o ciclos de vida en alguna etapa estable y reconocida. Por esta razón, en este trabajo se realizó un esfuerzo especial por identificar y levantar la información de proyectos de distribución de software de amplia trayectoria global. Esos proyectos se explican y justifican en el capítulo siguiente.

[9] http://www.payscale.com/research/US/Job=Software_Engineer_%2f_Developer_%2f_Programmer/Salary#CareerPaths

Capítulo 3

Datos referenciales de las más importantes distribuciones de software

---

Debian

Definición

Debian es una de las primeras distribuciones basadas en el núcleo Linux, también es una de las más prolíficas derivándose de ella otras cientos de distribuciones (Commons, 2017). Lowe et al. (2015) establecen que el proyecto Debian fue creado por Ian Murdock en agosto de 1993. En la página web del proyecto[1] definen a Debian como un sistema operativo libre, dispone de versiones para el núcleo Linux y el núcleo FreeBSD. Además, Debian se presenta con mas de 43000 paquetes de software. Debian está sostenida por una comunidad de voluntarios, organizados bajo directrices y estructuras que se explican a continuación.

[1] https://www.debian.org/intro/about

Gobernanza

Política

Debian ha creado un documento denominado Contrato social de Debian[2] que enmarca los principios filosóficos para el trabajo en torno al proyecto. Allí se especifican los puntos que fundamentan la distribución.

Se divide en dos secciones: el "contrato social" con la comunidad de software libre, y luego, las directrices de software libre de Debian (*Debian Free Software Guidelines – DFSG*). La primera parte establece los compromisos con la comunidad de software Libre y la segunda parte dispone las normas que deben cumplirse para que las contribuciones sean aceptadas y admitidas en el proyecto. Las directrices, de una manera más extendida, describen que el software debe ser compatible con las 4 libertades del Software Libre. De hecho, la Definición de Código Abierto[3] (*Open Source Definition – OSD*) está basada directamente en las DFSG.

Organización

La estructura organizativa de Debian expone una organización jerárquica. Está explicada en la "Constitución de Debian" cuya primera versión (versión 1.0) fue ratificada el 2 de diciembre de 1998 y su última versión (versión 1.7) ratificada el 14 de agosto de 2016[4]. El documento describe la estructura organizativa para la toma de decisiones y resolución de conflictos dentro del Proyecto Debian. Esta estructura está compuesto por: el Líder del Proyecto (*Debian Leader Project – DLP*), el Comité Técnico, el Secretario del Proyecto, los Delegados del Líder del Proyecto, los Desarrolladores Individuales y los nombrados mediante una Resolución General o elección. Ver Figura 3.1.

El Líder del Proyecto es el representante oficial del Proyecto Debian. El cargo tiene validez por un año. Cualquier Desarrollador Debian tiene derecho a postularse

[2] https://www.debian.org/social_contract
[3] https://opensource.org/osd
[4] https://www.debian.org/devel/constitution

como candidato y su designación se hace a través de una elección donde todos los Desarrolladores Debian tienen derecho a voto.

Entre sus atribuciones se tienen las siguientes tareas: designar delegados, asignar autoridad a otros desarrolladores, tomar decisiones urgentes, nombrar nuevos miembros del Comité Técnico (junto con el Comité).

El Comité Técnico es el organismo facultado para tomar decisiones finales sobre las disputas técnicas en el proyecto Debian. Está compuesto por un máximo de ocho Desarrolladores y debe tener un mínimo de cuatro miembros. Cuando el puesto queda vacante, se debe elegir a un presidente de entre y por los miembros del comité quienes son nominados automáticamente para este cargo. Si para el 1 de enero de cada año existiera algún miembro que ha servido por mas de 42 meses y además es uno de los dos miembros mas antiguos, entonces su nombramiento expira el 31 de diciembre de ese año.

Algunas de las atribuciones del Comité son: decidir sobre cualquier norma en materia técnica, tomar una decisión cuando se le solicite, nombrar nuevos miembros para sí mismo (junto con el Líder del Proyecto) o eliminar miembros existentes, nombrar al Presidente del Consejo Técnico, ejercer las funciones de Líder del Proyecto (junto con el Secretario).

El Secretario del Proyecto es nombrado por el Líder del Proyecto y el saliente Secretario del Proyecto. Cualquier Desarrollador Debian puede ser candidato. El cargo tiene una duración de un año con opción a ratificación.

Entre sus funciones se lista: el llevar a cabo votaciones, ejercer las funciones de Líder del Proyecto (junto con el Presidente del Comité Técnico), delegar parte de sus funciones a otro miembro.

Los Delegados nombrados por el Líder del Proyecto para tareas específicas, son nombrados por el Líder del Proyecto y pueden ser reemplazados a discreción del mismo.

Cumplen dos tareas principales: disponer de las atribuciones que el Líder del Proyecto haya delegado en ellos y tomar ciertas decisiones que el Líder no

puede tomar directamente.

Los Desarrolladores individuales son voluntarios que comparten los objetivos del proyecto y trabajan en consecuencia para lograrlos mientras participan en el proyecto.

Se cuenta entre sus concesiones: postularse como candidato a Líder del Proyecto, votar en las Resoluciones Generales y en las elecciones a Líder.

Los Desarrolladores mediante una Resolución General o elección. Tienen el derecho de: elegir o cesar al Líder del Proyecto, enmendar la constitución previa mayoría de las tres cuartas partes de los miembros, tomar o anular algunas decisiones.

Figura 3.1: Integrantes de la estructura organizativa de Debian

A su vez, dentro de la comunidad de desarrolladores también se presentan diferentes roles, y en Debian reciben distintas denominaciones según su actividad y permisos para la publicación de paquetes en los repositorios de Debian.

De este modo, dentro del entorno Debian los desarrolladores se distinguen en:

Autor original (*upstream author*): es la persona que escribió el programa original.

Mantenedor actual (*upstream maintainer*): es la persona que actualmente se encarga de mantener el programa.

Mantenedor (*maintainer*): es la persona que se encarga de empaquetar el programa para Debian.

Patrocinador (*sponsor*): es la persona que ayuda a los mantenedores a subir los paquetes al repositorio oficial de paquetes Debian, luego de que estos sean revisados.

Mentor: es la persona que ayuda a los mantenedores noveles con el empaquetado y otras actividades de desarrollo.

Mantenedor Debian (DM) (*Debian Maintainer*): es una persona con permisos limitados de subida de paquetes al repositorio oficial de paquetes Debian.

Desarrollador Debian (DD) (*Debian Developer*): es un miembro del proyecto Debian con permisos plenos de subida de paquetes al repositorio oficial de paquetes Debian.

Colaboradores

Para finales del año 2016 existían poco más de 1300 colaboradores del proyecto Debian entre Mantenedores Debian y Desarrolladores Debian.

- 271 Mantenedores Debian[5]
- 1043 Desarrolladores Debian[6]

Histórico de distribuciones

El hecho de que una distribución mantenga un archivo donde se almacenan las versiones publicadas durante su vida añade la posibilidad de explorar aspectos históricos y técnicos de la distribución.

[5] https://nm.debian.org/public/people/dm_all [Consultado 14 de noviembre de 2016]
[6] https://nm.debian.org/public/people/dd_all [Consultado 14 de noviembre de 2016]

Debian mantiene un registro de la mayoría de las versiones liberadas hasta la fecha, específicamente desde la versión 3.0 (Woody). Este directorio puede encontrarse en la dirección http://cdimage.debian.org/mirror/cdimage/archive/.

Lanzamientos

La Tabla 3.1 muestra la fecha de lanzamiento de las diferentes versiones de la distribución Debian.

Tabla 3.1: Lanzamientos de versiones de Debian

Versión	**Nombre Clave**	**Fecha de Lanzamiento**
0.1–0.90	Debian	agosto–diciembre 1993
0.91	Debian	enero 1994
0.93R5	Debian	marzo 1995
0.93R6	Debian	26/10/1995
1.1	Buzz	17/06/1996
1.2	Rex	12/12/1996
1.3	Bo	02/07/1997
2.0	Hamm	24/07/1998
2.1	Slink	09/03/1999
2.2	Potato	15/08/2000
3.0	Woody	19/07/2002
3.1	Sarge	06/06/2005
4.0	Etch	08/04/2007
5.0	Lenny	14/02/2009
6.0	Squeeze	06/02/2011
7.0	Wheezy	04/05/2013
8.0	Jessie	25/04/2015
9.0	Stretch	17/06/2017

Fuentes: Lowe et al. (2015) y *DebianReleases*[7]

[7] https://wiki.debian.org/DebianReleases

En la tabla se distinguen dos etapas, la primera se da durante la primera década de vida, con lanzamientos constantes en cortos períodos de tiempo para luego entrar a la segunda etapa más estable con lanzamientos bienales.

Arquitecturas

Hasta la versión 8.0 (Jessie) Debian soporta las arquitecturas mostradas en la tabla 3.2.

Tabla 3.2: Arquitecturas soportadas oficialmente por Debian

Adaptación	**Arquitectura**
amd64	PC de 64 bits (amd64)
arm64	ARM de 64 bits (AArch64)
armel	EABI ARM
armhf	ABI ARM de punto flotante
i386	PC de 32 bits (i386)
mips	MIPS (modo big-endian)
mipsel	MIPS (modo little-endian)
powerpc	Motorola/IBM PowerPC
ppc64el	POWER7+, POWER8
s390x	System z

Fuentes: Debian[8], DistroWatch[9]

De las tres distribuciones en este estudio, Debian es la que más soporte ofrece a distintas arquitecturas y plataformas. La tabla 3.2 sólo muestra los trabajos de soporte oficiales, pero también existen adaptaciones en desarrollo[10] para el funcionamiento de la distribución en otros sistemas como GNU/Hurd y FreeBSD.

[8] https://www.debian.org/ports/index.es.html#portlist-released
[9] https://distrowatch.com/table.php?distribution=debian

[10] https://www.debian.org/ports/index.es.html#portlist-other

Repositorios Debian

El proyecto Debian puede prestar soporte a los paquetes en 4 repositorios de aplicaciones de forma simultánea.

Antigua estable (*oldstable*): Esta versión contiene la versión inmediatamente anterior a la versión estable.

Estable (*stable*): La versión estable contiene la última versión oficial de la distribución Debian.

En pruebas (*testing*): La rama de pruebas contiene paquetes que aún no han sido aceptados dentro de la versión estable, pero se encuentra en cola para ello.

Inestable (*unstable*): La versión inestable es la version que se encuentra en desarrollo constante. Esta versión siempre es llamada por el nombre clave “Sid”.

Debian también alberga un directorio ***Experimental***. Esta rama aloja algunos paquetes en fases muy tempranas de desarrollo. Es más un repositorio de trabajo aislado pensado para el uso de los desarrolladores y empaquetadores a modo de laboratorio para ir probando sus paquetes y recibir comentarios de otros desarrolladores mientras alcanzan etapas estables de desarrollo.

La rama estable tiene soporte completo de seguridad. Los lanzamientos de nuevas versiones suceden aproximadamente cada dos años. Cuando una versión estable pasa a la fase antigua estable recibe soporte del equipo de seguridad por aproximadamente un año mas.

Existe un proyecto que inició con la versión 6 (Squeeze) de Debian para dar soporte a largo plazo (*Long Term Support* – LTS, en inglés) a todas las versiones estables, el objetivo es extender por dos años adicionales el soporte a las ramas. El equipo está integrado por voluntarios que toman la responsabilidad del mantenimiento de la seguridad una vez que el equipo de seguridad de Debian haya culminado el plazo establecido de soporte. De este modo se tiene, dos años aproximadamente de soporte mientras la rama está en su fase estable, un año mientras es antigua estable, y dos años de soporte adicionales como LTS para un aproximado de 5 años en total para cada versión publicada.

Componentes

Según su compatibilidad con las directrices de software libre de Debian (DFSG)[11] Debian categoriza los paquetes incluidos en su sistema dentro de tres componentes, estos son:

main: Paquetes compatibles con las DFSG, que además no dependen de paquetes que se encuentren fuera de este componente para su operación. Estos paquetes son los únicos que se consideran parte de la distribución Debian.

contrib: Paquetes compatibles con las DFSG pero que tienen dependencias fuera de *main*, incluida la posibilidad de tener dependencias en el componente *non-free*.

non-free: Paquetes no compatibles con las DFSG.

Listas de discusión

La cantidad de personas involucradas en los proyectos colaborativos así como la distribución geográfica desde donde participan hacen necesaria la búsqueda de herramientas que posibiliten una adecuada interacción entre los actores.

La lista de correo es quizás la herramienta que mejor se adapta a los fines comunicacionales en las comunidades de desarrollo distribuido. Permite el envío y recepción de mensajes a múltiples usuarios de manera inmediata o compiladas en tiempos preestablecidos y facilita la persistencia de un registro de mensajes ordenados cronológicamente.

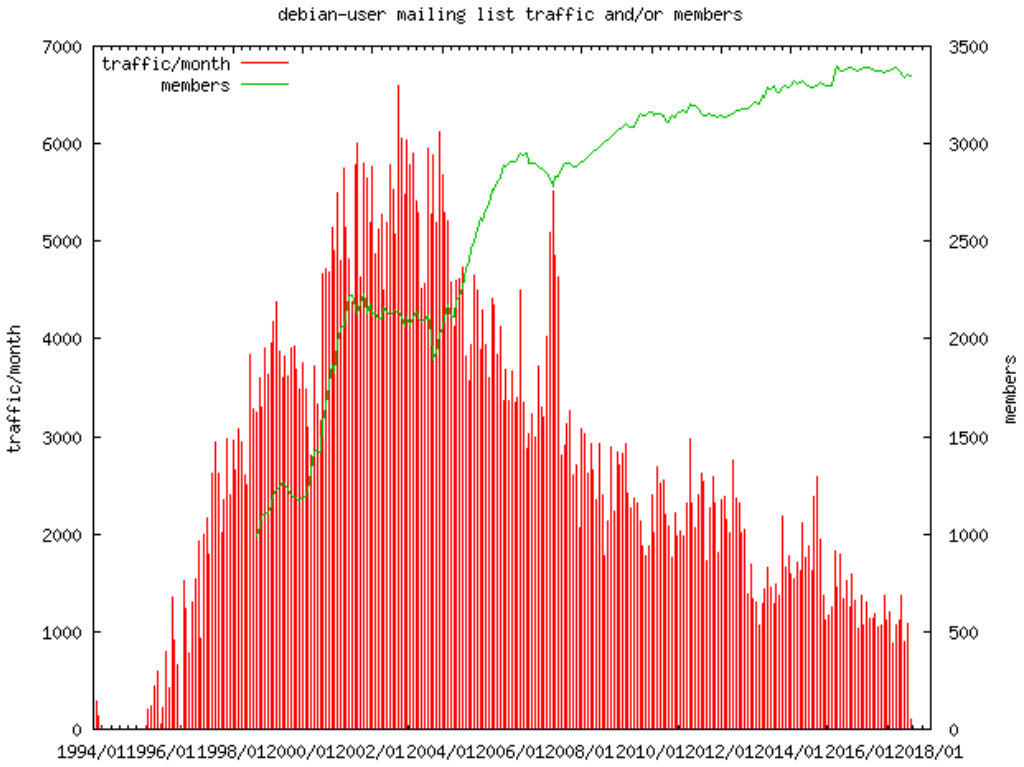
En este trabajo se cuantifica la actividad en dos listas públicas de correos específicas para cada distribución, la lista de discusión de usuarios y la lista de desarrolladores. Esto, con el objetivo de mostrar el movimiento y la interacción de la comunidad que integra dichas listas. Las listas de usuarios están entendidas para el soporte y asistencia de los usuarios de las distribuciones y la listas de desarrolladores para temas más técnicos.

[11] https://www.debian.org/social_contract.es.html#guidelines

Lista de usuarios

La Figura 3.2 muestra un crecimiento constante. El pico de actividad más alto en la lista es alcanzado en octubre del año 2002 superando la barrera de los 6500 mensajes en ese mes, centrándose en temas de soporte y discusión. Luego comienza un paulatino descenso hasta marzo de 2007 donde se aprecia un repunte de 5499 mensajes en donde se presentan discusiones interesantes sobre derechos de autor, soporte y opiniones devenidas en una declaración de Ian Murdock sobre el impacto de Ubuntu en los usuarios de Debian[12]. Luego continua su descenso en la actividad, presentando breves momentos de actividad intensificada, con temas relacionados a soporte.

Figura 3.2: Actividad de la lista de correos *Users* de Debian



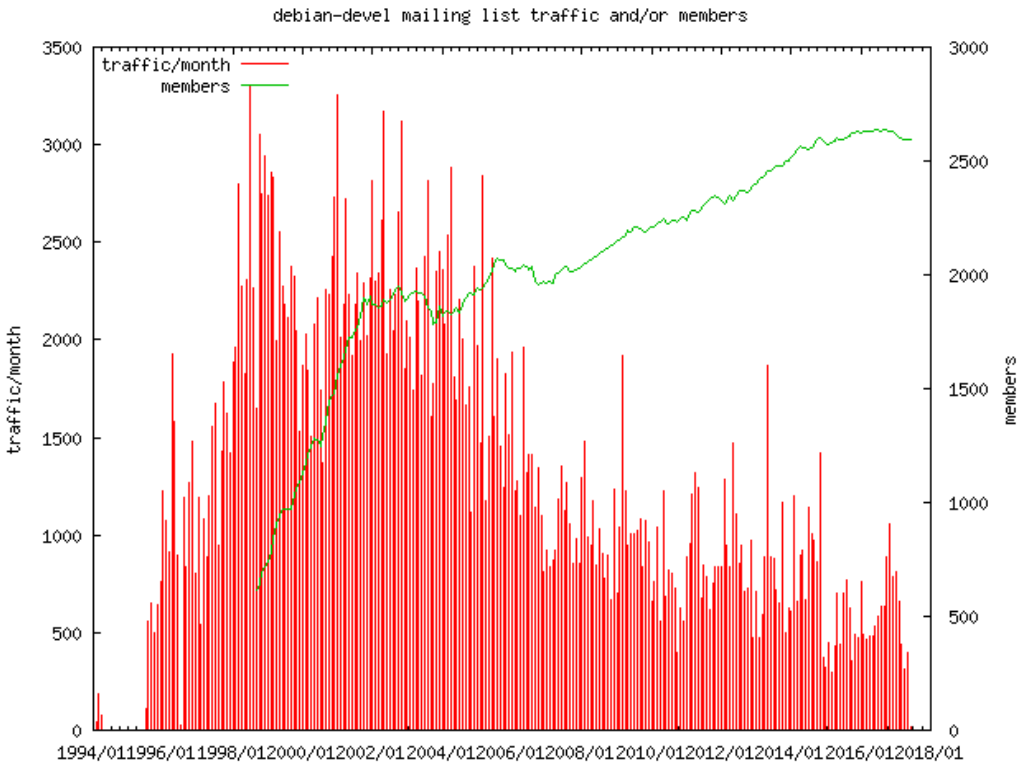
Fuente: https://lists.debian.org/debian-user/ (Recuperado 06/06/2017)

[12] https://lists.debian.org/debian-user/2007/03/msg03347.html

Lista de desarrolladores

Para el caso de la actividad en la lista de desarrolladores de Debian, presentada en la Figura 3.3, se percibe un comportamiento ligeramente similar, pero con picos de actividad más intensificados y más frecuentes. Los periodos de mayor actividad se encuentra entre los años 1998 y 2006, no llegando a superar los 3500 mensajes al mes.

Figura 3.3: Actividad de la lista de correos *Developers* de Debian



Fuente: https://lists.debian.org/debian-devel/ (Recuperado 07/06/2017)

Ciclo de vida de un paquete en Debian

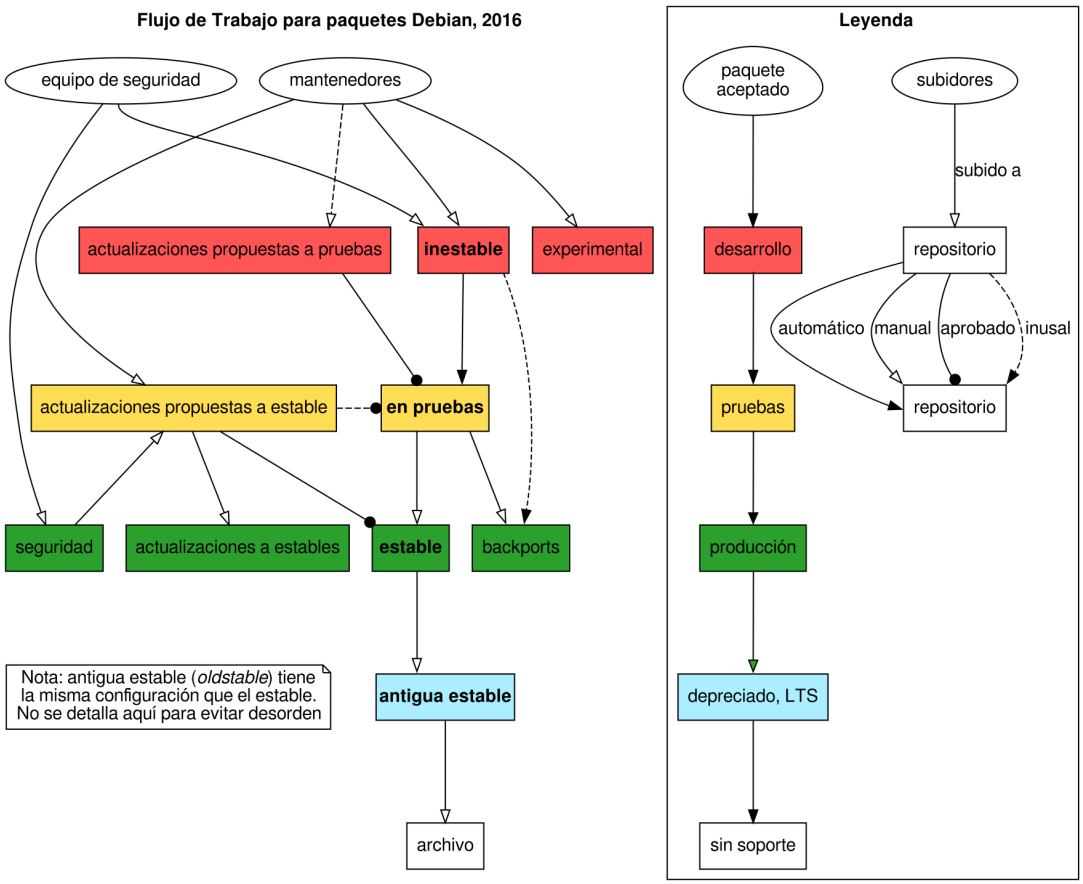
En la figura 3.4 se presenta el diagrama de flujo de paquetes entre las diferentes versiones de Debian.

Allí se describe las posibles rutas que atraviesa un paquete aceptado a través las diferentes ramas y repositorios del sistema Debian. Los repositorios en negrita

representan aquellos que contienen el sistema base, es decir, ese repositorio contiene los paquetes requeridos para poder tener un sistema operativo funcional, mientras que el resto de repositorios solamente contienen algunos paquetes y no podrían funcionar por sí solos.

El diagrama se explica de la siguiente forma. Un paquete puede ser subido por el equipo de seguridad a la rama de desarrollo (inestable) o directamente al repositorio de seguridad para corregir errores urgentes que puedan vulnerar el sistema.

Figura 3.4: Flujo de trabajo de paquetes Debian para 2016



Fuente: Imagen basada en el código de Antoine Beaupré[13]

[13] https://anonscm.debian.org/git/collab-maint/package-cycle.git/tree/package-cycle.dot (Recuperado Junio 2017)

La otra forma es transitar por el proceso regular. El paquete sube a los repositorios de desarrollo, al repositorio inestable, donde será compilado para las diferentes arquitecturas y será probado hasta que éste se encuentre en un estado sin fallos críticos y sin modificaciones recientes.

Para pasar de la fase en pruebas o *testing* a la versión estable, primero se debe anunciar una etapa de congelamiento conocida como "*freeze*". En esta etapa se impide la subida de nuevas versiones para evitar agregar paquetes con nuevos errores. Cada paquete se prueba para su aprobación final y aquellos paquetes en los que no se hayan corregidos sus errores serán eliminados del repositorio.

Culminada la etapa de congelamiento y comprobado todo el repositorio, este pasa a ser el nuevo repositorio estable, y el que era estable, pasa a antiguo estable. A su vez, el que era antigua estable deja de tener soporte del proyecto Debian y pasa a manos de voluntarios que prestarán soporte de seguridad como versión LTS por dos años más para luego finalizar en los archivos del proyecto.

La figura 3.4 también muestra repositorios especiales como los repositorios de actualizaciones propuestas a las diferentes versiones, el repositorio de seguridad, y el de retroimportaciones (*backports* en inglés).

Los repositorios "Actualizaciones Propuestas para pruebas", "Actualizaciones Propuestas para estable" y "Actualizaciones de estable" son repositorios donde se van colocando los paquetes listos para ser incluidos en la próxima "actualización puntual", cada dos meses se hace una nueva actualización de la versión estable.

El repositorio de retroimportaciones toma los paquetes de la versión en pruebas y se ajustan y recompilan para su uso en la versión estable. De esta forma, se tiene un repositorio en la versión estable con paquetes que ya están actualizados y están siendo utilizados en la versión de pruebas.

Estimación de costos

La Tabla 3.3 expone un resumen con los datos obtenidos del análisis realizado sobre el componente *main* de Debian. Se estudiaron 20.981 paquetes de software y se detectaron 179 lenguajes.

Tabla 3.3: SLOC agrupados por lenguaje para Debian

Lenguaje	Cantidad de archivos	Líneas en blanco	Líneas con comentarios	Líneas de código fuente
C	534.083	**40.610.907**	**45.978.259**	**228.239.179**
C++	484.030	26.725.540	24.715.793	141.455.570
Bourne Shell	95.241	14.824.473	15.983.187	99.179.801
Cabecera C/C++	**809.250**	19.839.105	36.676.482	88.063.518
HTML	365.730	6.967.723	1.841.372	65.097.778
m4	49.654	3.183.855	698.068	56.911.113
Java	398.850	9.858.765	23.133.295	47.973.926
XML	179.091	2.218.577	1.456.931	45.182.189
Python	196.352	7.544.353	9.611.562	32.015.199
Qt Linguist	7.535	111.793	18	21.013.940
JavaScript	148.352	3.461.765	4.497.530	18.030.906
Perl	100.525	4.420.066	5.141.400	17.125.106
TEX	23.472	1.252.733	3.265.439	10.365.660
C#	67.943	1.803.520	2.796.671	10.173.671
Fortran 77	42.097	484.591	4.295.158	9.407.107
Fortran 90	12.819	312.233	1.108.344	7.448.896
Lisp	18.520	1.092.827	1.537.110	7.339.162
Module Definition	8.624	96.572	48.120	7.095.719
PHP	52.913	1.161.255	3.146.443	6.469.612
Otros	731.630	13.414.341	14.506.986	96.629.959
Total	**4.326.711**	**159.384.994**	**200.438.168**	**1.015.218.011**

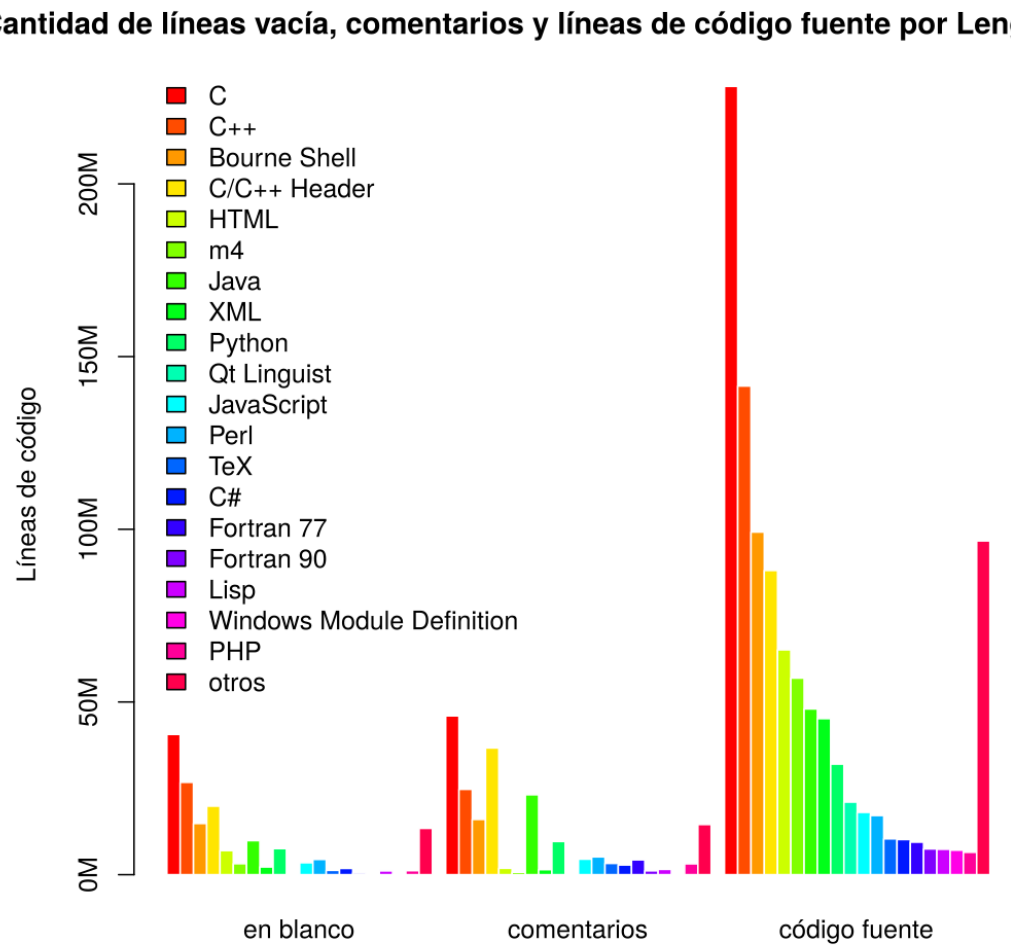
Fecha del archivo Source: 06/05/2017

Número de paquetes descargados: 20.981

Cantidad de lenguajes reconocidos: 179

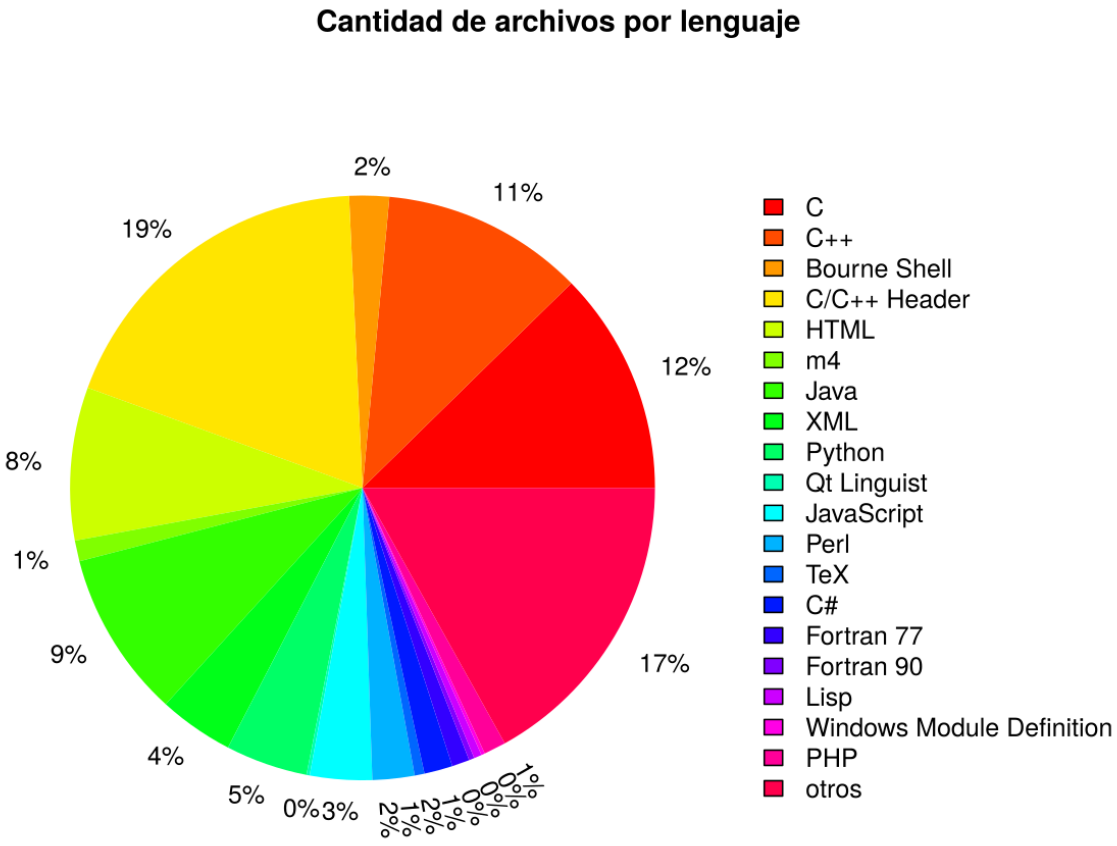
Se aprecia como el lenguaje C domina en cantidad de líneas en blanco, líneas con comentarios y líneas de código fuente. La Figura 3.5 visualiza los resultados de este análisis.

Figura 3.5: Cantidad de líneas vacía, comentarios y líneas de código fuente en los lenguajes del componente *main* de Debian



Los archivos de cabecera C/C++ lideran como el lenguaje con mayor cantidad de archivos presente en el repositorio, con un 19% del total analizado, seguido de los archivos codificados en lenguaje C y C++ con 12% y 11% respectivamente, tal como se muestra en la Figura 3.6.

Figura 3.6: Porcentaje de archivos por lenguaje dentro del componente *main* de Debian



COCOMO

La cantidad de líneas de código fuente es el insumo principal para la estimación de esfuerzo y costos utilizando COCOMO básico. El total de código fuente de todos los paquetes analizados en el componente *main* de Debian en su versión 8.0 supera las 1.015 millones de líneas. La Tabla 3.4 presenta los valores obtenidos aplicando COCOMO Básico.

Tabla 3.4: Estimaciones de esfuerzo y costos para el componente *main* de Debian aplicando COCOMO Básico

Líneas de código fuente:	1.015.218.011
Esfuerzo estimado de desarrollo (persona–mes):	4.865.175,62
Tiempo de desarrollo estimado (meses):	869,04
Personas requeridas estimadas (personas):	5.598,36
Costo total estimado del proyecto (US$):	413.366.347,61

Para el Costo Total estimado se toma el valor US$ 73.837,00 anual como salario de referencia para ingenieros de software, desarrolladores y programadores en enero de 2017[14].

Así, para enero de 2017, el desarrollo de Debian costaría aproximadamente US$ 413.366.347,61, tomando como referencia el salario calculado a US$ 73.837 y cerca de 5600 personas. El tiempo requerido para completar el proyecto sería de 72 años aproximadamente.

Ubuntu

Definición

Ubuntu es una distribución GNU/Linux basada en Debian, concebida por Mark Shuttleworth en 2004. Según el equipo que trabaja en el Manual de Ubuntu, se estima que Ubuntu está instalado en el 2% de las computadoras a nivel mundial, esto equivale a diez millones de usuarios en todo el mundo (The Ubuntu Manual Team, 2016). Sin embargo, la página "Ubuntu Insights"[15] de Ubuntu afirma que, para mayo de 2017, existían más de 40 millones de usuarios de escritorio.

[14] http://www.payscale.com/research/US/Job=Software_Engineer_%2f_Developer_%2f_Programmer/Salary#CareerPaths

[15] https://insights.ubuntu.com/about

El soporte comercial de Ubuntu proviene de la empresa Canonical[16], propiedad también de Mark Shuttleworth. Luego, la distribución Ubuntu es un proyecto que mezcla un proyecto colaborativo entre empresa privada y comunidad de desarrolladores. Esta aleación empresa–comunidad se ve esquematizada en la estructura organizativa del proyecto, la cual se describe en seguida.

Gobernanza

Política

Ubuntu explica sus políticas a través de varias páginas en su web y al igual que Debian, Ubuntu comparte los principios del movimiento de Software Libre y de Código Abierto.

La filosofía de la distribución traza los objetivos principales que conducen el trabajo dentro del proyecto: El software debe ser libre y accesible a todo el mundo[17].

De este modo, Ubuntu cree que cada usuario de computadora debería:

- Poseer la libertad de descarga, ejecución, copia, distribución, estudio, compartición, modificación y mejoramiento del software para cualquier propósito, sin tener que pagar por licencias.
- Ser capaz de usar su software en la lengua de su elección.
- Poder usar todo el software independientemente de su discapacidad.

Asimismo, Ubuntu estableció un Código de Conducta[18] que sirve como carta de compromiso entre los miembros de Ubuntu para la interacción de los mismos. El Código de Conducta es la pauta que indica como se deben dirimir los conflictos dentro del proyecto. Al mismo tiempo, es una guía que incentiva la colaboración y el liderazgo.

[16] https://www.canonical.com
[17] https://www.ubuntu.com/about/about-ubuntu/our-philosophy
[18] https://www.ubuntu.com/about/about-ubuntu/conduct

Roles

Patrocinador (*sponsors*): Es la persona que puede revisar el paquete y subirlo a los repositorios.

Desarrollador prospecto de Ubuntu (*Ubuntu Prospective Developers*): Es aquella persona que recién comienza a contribuir con Ubuntu.

Desarrollador contribuidor de Ubuntu (*Ubuntu Contributing Developers*): Es aquella persona reconocida con una membresía de Ubuntu.

Desarrollador Ubuntu de equipos delegados (*Ubuntu Developers from delegated teams*): Es la persona que puede subir paquetes a determinados grupos de trabajo.

MOTU (*Master of the Universe*): Es aquella persona que puede subir paquetes a los repositorios *Universe* and *Multiverse*.

Desarrollador del núcleo de Ubuntu (core-dev) (*Ubuntu Core Developers*): Es aquella persona que puede subir paquetes a todas las áreas de Ubuntu.

Subidor por paquete (Per-package Uploaders): Es aquella persona que puede subir paquetes específicos.

Colaboradores

- 87 Ubuntu Core Developers[19]
- 159 Ubuntu Contributing Developers[20]
- 148 MOTU[21]
- 734 Ubuntu Members[22]

[19] `https://launchpad.net/~ubuntu-core-dev` [Consultado 14 de noviembre de 2016]
[20] `https://launchpad.net/~ubuntu-developer-members` [Consultado 14 de noviembre de 2016]
[21] `https://launchpad.net/~motu` [Consultado 14 de noviembre de 2016]
[22] `https://launchpad.net/~ubuntumembers` [Consultado 14 de noviembre de 2016]

Componentes

Ubuntu categoriza los paquetes incluidos en su sistema dentro de cuatro repositorios, de acuerdo a si son libres o no, según la Filosofía de Software Libre de Ubuntu[23], estos repositorios son:

Main: Software libre y abierto mantenido por Canonical.

Universe: Software libre y abierto mantenido por la comunidad.

Multiverse: Software con restricciones de licencia.

Restricted: Controladores propietarios.

Arquitecturas

Ubuntu está oficialmente soportada y portada para 7 arquitecturas, mostradas en la tabla 3.5.

Tabla 3.5: Arquitecturas soportadas oficialmente por Ubuntu

Adaptación	Arquitectura
i386	Intel x86
amd64	AMD64, Intel 64 (x86_64) y EM64T
arm64	ARM SoC (sistema en chip) de 64 bit
armhf	ARM con hardware FPU
ppc64el	POWER8 y variantes OpenPOWER
S390X	System z y LinuxONE
powerpc	IBM/Motorola PowerPC

Fuentes: Ubuntu Documentation[24]

[23] https://www.ubuntu.com/about/about-ubuntu/our-philosophy
[24] https://help.ubuntu.com/community/SupportedArchitectures

Tiempos de soporte

Ubuntu hace lanzamientos con diferentes tiempos de soporte, cada seis meses se publica una versión estable, y las versiones estable con soporte de larga duración se publican cada dos años.

Estable: Soporte por nueve meses.

LTS (*Long Term Support*): Soporte por cinco años.

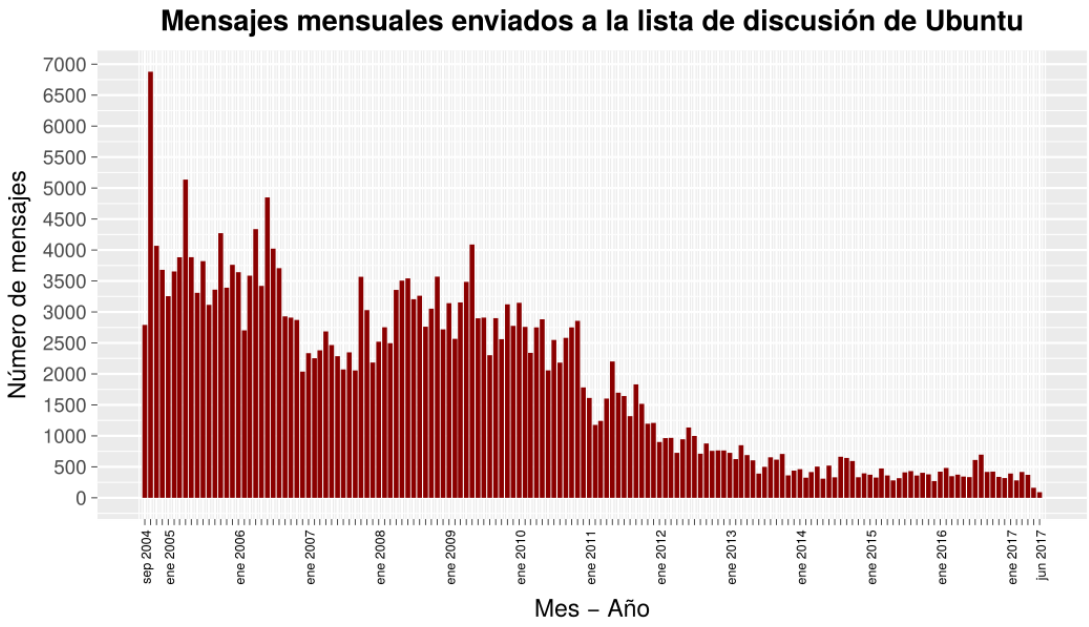
Histórico de distribuciones

- Ubuntu `http://old-releases.ubuntu.com/releases/`

Listas de discusión

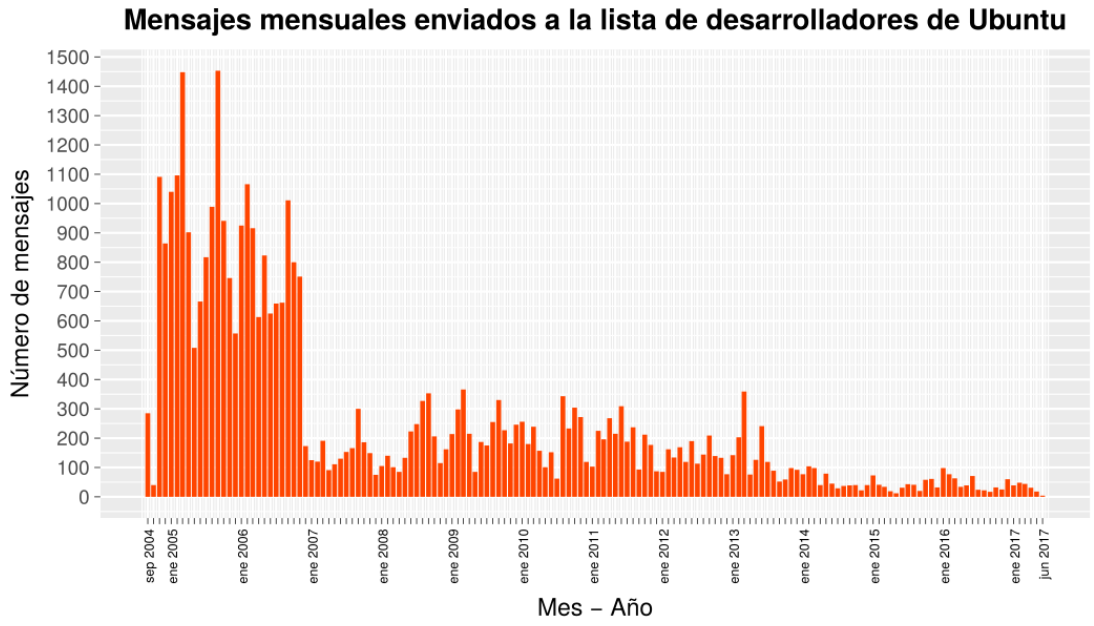
Lista de usuarios

Figura 3.7: Actividad de la lista de correos *Users* de Ubuntu



Lista de desarrolladores

Figura 3.8: Actividad de la lista de correos *Developers* de Ubuntu



Lanzamientos

La Tabla 3.6 muestra la fecha de lanzamiento de las diferentes versiones de la distribución Ubuntu.

Tabla 3.6: Lanzamientos de versiones de Ubuntu

Versión	**Nombre Clave**	**Fecha de Lanzamiento**
4.10	Warty Warthog	20/10/2004
5.04	Hoary Hedgehog	08/04/2005
5.10	Breezy Badger	13/10/2005
6.06 LTS	Dapper Drake	01/06/2006
6.10	Edgy Eft	26/10/2006
7.04	Feisty Fawn	19/04/2007
7.10	Gutsy Gibbon	18/10/2007
8.04 LTS	Hardy Heron	24/04/2008
8.10	Intrepid Ibex	30/10/2008
9.04	Jaunty Jackalope	23/04/2009
9.10	Karmic Koala	29/10/2009
10.04 LTS	Lucid Lynx	29/04/2010
10.10	Maverick Meerkat	10/10/2010
11.04	Natty Narwhal	28/04/2011
11.10	Oneiric Ocelot	13/10/2011
12.04 LTS	Precise Pangolin	26/04/2012
12.10	Quantal Quetzal	18/10/2012
13.04	Raring Ringtail	25/04/2013
13.10	Saucy Salamander	17/10/2013
14.04 LTS	Saucy Salamander	17/04/2014
14.10	Utopic Unicorn	23/10/2014
15.04	Vivid Vervet	23/04/2015
15.10	Wily Werewolf	21/10/2015
16.04 LTS	Xenial Xerus	21/04/2016
16.10	Yakkety Yak	13/10/2016

Fuentes: Wikipedia [25]

Estimación de costos

Componente *main*

- Fecha del archivo Source: 02/04/2017
- Número de paquetes descargados: 2.495
- Cantidad de lenguajes reconocidos: 150

[25] https://es.wikipedia.org/wiki/Ubuntu#Lanzamientos_y_soporte

Tabla 3.7: SLOC agrupados por lenguaje para Ubuntu

Lenguaje	**Cantidad de archivos**	**Líneas en blanco**	**Líneas con comentarios**	**Líneas de código fuente**
C	157.152	11.690.328	12.772.978	66.131.833
C++	125.344	6.748.270	6.123.082	37.054.512
Cabecera C/C++	208.053	5.092.406	8.860.093	25.843.040
HTML	105.562	2.635.762	466.304	19.884.751
Bourne Shell	19.586	2.907.499	2.291.133	15.932.894
XML	43.247	423.980	93.078	10.597.667
Python	50.832	2.072.539	2.532.364	8.568.869
JavaScript	47.643	1.222.966	1.748.238	7.324.667
C#	41.546	1.116.899	1.324.878	6.064.615
TEX	13.314	499.529	2.314.342	5.766.279
m4	7.007	391.713	109.621	3.590.276
Perl	13.508	592.890	749.975	3.309.042
Ensamblador	15.860	396.138	649.093	2.882.766
Java	18.395	546.084	1.194.939	2.540.210
Go	11.674	334.856	409.455	2.457.237
JSON	6.015	3.834	0	1.819.038
Erlang	4.541	254.875	347.135	1.521.217
Lisp	2.247	175.152	246.317	1.248.017
YAML	1.579	5.852	2.771	1.234.255
Otros	161.869	2.155.228	2.753.629	16.526.958
Total	**1.054.974**	**39.266.834**	**44.989.783**	**240.297.751**

Figura 3.9: Porcentaje de archivos por lenguaje dentro del componente *main* de Ubuntu

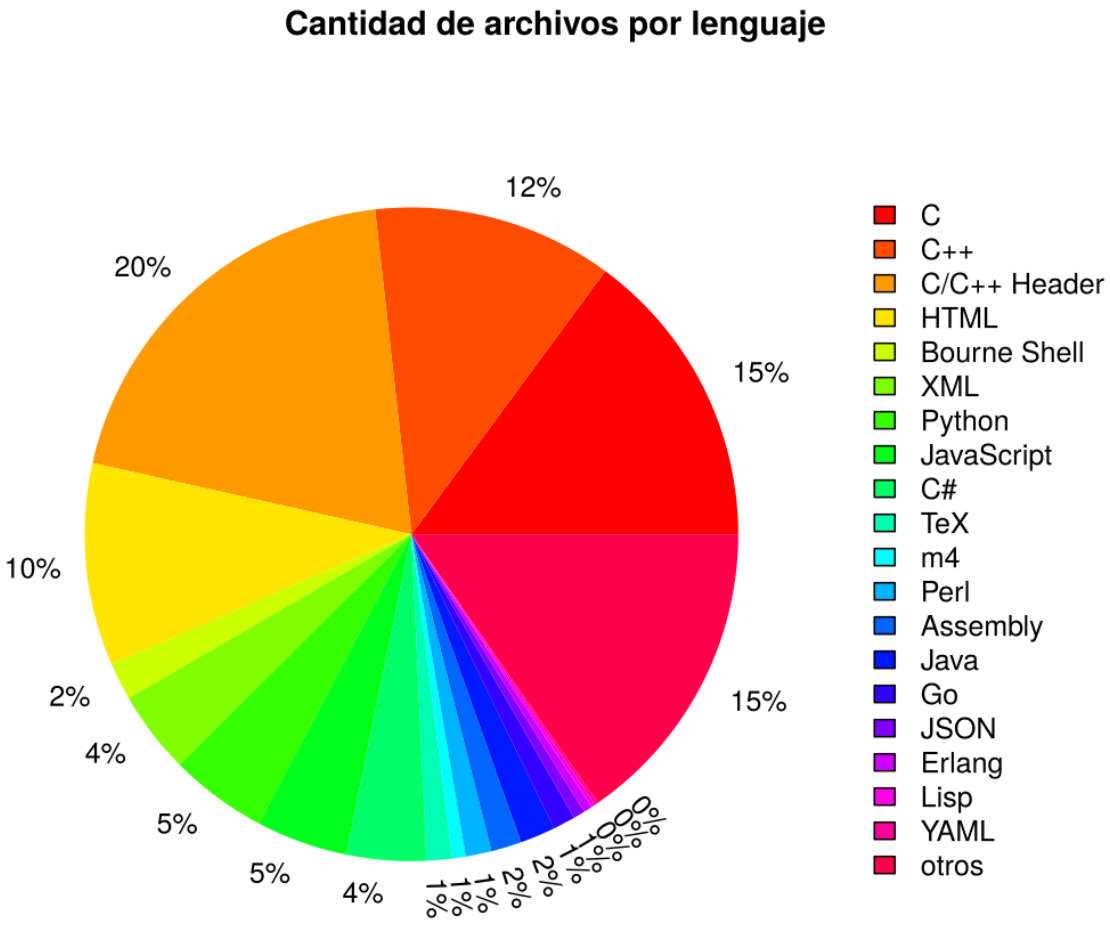
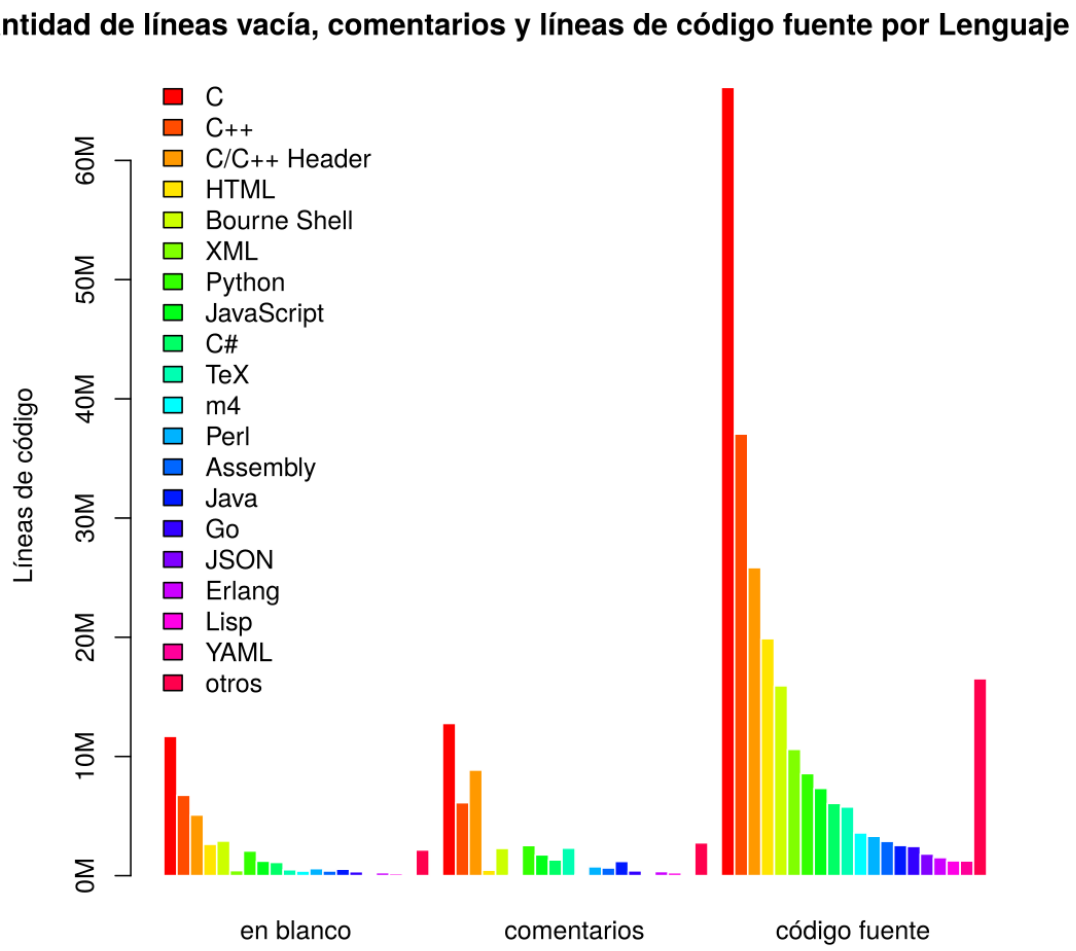


Figura 3.10: Cantidad de líneas vacía, comentarios y líneas de código fuente en los lenguajes del componente *main* de Ubuntu



COCOMO

Tabla 3.8: Estimaciones de esfuerzo y costos para el componente *main* de Ubuntu aplicando COCOMO Básico

Líneas de código fuente:	240.297.751
Esfuerzo estimado de desarrollo (persona–mes):	1.071.515,44
Productividad estimada:	0,22
Tiempo de desarrollo estimado (meses):	489,03
Personas requeridas estimadas (personas):	2.191,08
Costo total estimado del proyecto (US$):	161.783.066,47

Para el Costo Total estimado se toma el valor US$ 73.837,00 anual como salario de referencia para ingenieros de software, desarrolladores y programadores en enero de 2017[26].

Canaima GNU/Linux

Definición

Canaima GNU/Linux es una metadistribución Linux basada en Debian, forma parte del Proyecto Canaima, originado a raíz de la promulgación del decreto 3.390 el cual dispone sobre el uso prioritario de Software Libre en los entes pertenecientes a la Administración Pública Nacional (APN). En el artículo número 7 se designa al entonces Ministerio de Ciencia y Tecnología la responsabilidad de "proveer la Distribución Software Libre desarrollado con Estándares Abiertos para el Estado Venezolano" (República Bolivariana de Venezuela, 2004, p.9).

Canaima se crea "con el propósito de lograr inter-operabilidad, homogeneidad y sustentabilidad de la plataforma informática que utilizan los servidores públicos en

[26] http://www.payscale.com/research/US/Job=Software_Engineer_%2f_Developer_%2f_Programmer/Salary#CareerPaths

sus procesos de producción intelectual y gestión" (Equipo de desarrollo de Canaima, 2009, p.7). Además, "su objetivo estratégico es apalancar el proceso de migración a Software Libre en las instituciones de la Administración Pública Nacional, para avanzar hacia la independencia tecnológica, con pleno ejercicio de la soberanía nacional" (Equipo de desarrollo de Canaima, 2009, p.7).

Así pues, 3 años luego del Decreto 3390, en 2007, nace la primera versión de la Distribución GNU/Linux del Estado Venezolano desarrollada en primera instancia para el Ministerio de Ciencia y Tecnología Martínez (2012), y que poco después adoptaría el nombre de Canaima.

Sobre la distribución y el proyecto Canaima Figuera (2013) nos dice:

> Para su generación y mantenimiento, se creó el Proyecto Canaima, que ha evolucionado desde un producto de una institución particular, hasta un complejo y rico proyecto socio-tecnológico con diversas ramificaciones, donde participan decenas de personas distribuidas en toda la geografía nacional, a título individual o provenientes de instituciones, colectivos sociales y organizaciones diversas. (p.198)

El desarrollo de Canaima inicia con un estilo del tipo catedral, que poco a poco muestra una migración a un estilo del tipo bazar.

Componentes

Canaima, siendo una distribución derivada de Debian, comparte la misma estructura de repositorios:

usuarios: Paquetes compatibles con las DFSG, que además no dependen de paquetes que se encuentren fuera de este componente para su operación. Estos paquetes son los únicos que se consideran parte de la distribución Debian.

aportes: Paquetes compatibles con las DFSG pero que tienen dependencias fuera de *main*, incluida la posibilidad de tener dependencias en el componente *non-free*.

no-libres: Paquetes no compatibles con las DFSG.

Arquitecturas

Canaima construye su distro bajo 2 arquitecturas, mostradas en la tabla 3.9.

Tabla 3.9: Arquitecturas soportadas oficialmente por Canaima GNU/Linux

Adaptación	Arquitectura
i386	PC de 32 bits (i386)
amd64	PC de 64 bits (amd64)

Fuentes: Repositorio Canaima[27]

Equipo Canaima GNU/Linux (septiembre de 2016)

- Un Jefe de Oficina Canaima GNU/Linux.
  - Coordinar el equipo de trabajo del Proyecto Canaima.
  - Promover el cambio del modelo productivo del Proyecto desde el año 2014.
  - Producción de software centrado en el usuario final.
  - Diseño y construcción de una fábrica de ensamblaje de software en Venezuela.
- Dos Articulación Sociotecnológica.
  - Impulso de procesos de sistematización de experiencias, organización y métodos, desarrollo de laboratorios de usabilidad para la evaluación del entorno de escritorio de Canaima GNU/Linux.
  - Apoyo a procesos de migración a tecnologías libres.
  - Redacción, edición y publicación de notas de prensa para el Portal Web del Proyecto Canaima GNU/Linux.

[27] http://repositorio.canaima.softwarelibre.gob.ve/

    - Levantamiento y análisis de requerimientos técnicos
    - Formación tecnológica
    - Sistematización de experiencias, planificación y gestión.

- Cuatro desarrolladores.
    - Gestión de la plataforma de versionamiento de Canaima GNU/Linux
    - Adaptación de Paquetes.
    - Mantenimiento de Paquetes.
    - Pruebas de correcto funcionamiento de los paquetes.
    - Adaptación de Canaima GNU/Linux a Canaima Educativo.
- Dos Plataforma Tecnológica.
    - Mantenimiento y Actualización de la Plataforma Tecnológica.
    - Monitoreo de la Plataforma Tecnológica.
    - Creación de servicios para el uso del área de desarrollo y comunidad.
- Un Laboratorio y Plataforma Tecnológica de Canaima Educativo.
    - Probar y adaptar las versiones de Canaima GNU/Linux en las portátiles Canaima.
    - Administrar parte de la plataforma de Canaima Educativo.
- Un Soporte Técnico.
    - Solventar casos registrados en los medios de soporte técnico de Canaima.
    - Realizar pruebas de las versiones de Canaima.
    - Actualizar constantemente los espacios de Soporte Técnico de Canaima.
- Un Gestión Administrativa.
    - Procesos administrativos del proyecto Canaima GNU/Linux.

Política

- Contrato social de Canaima http://wiki.canaima.softwarelibre.gob.ve/index.php/Contrato_Social

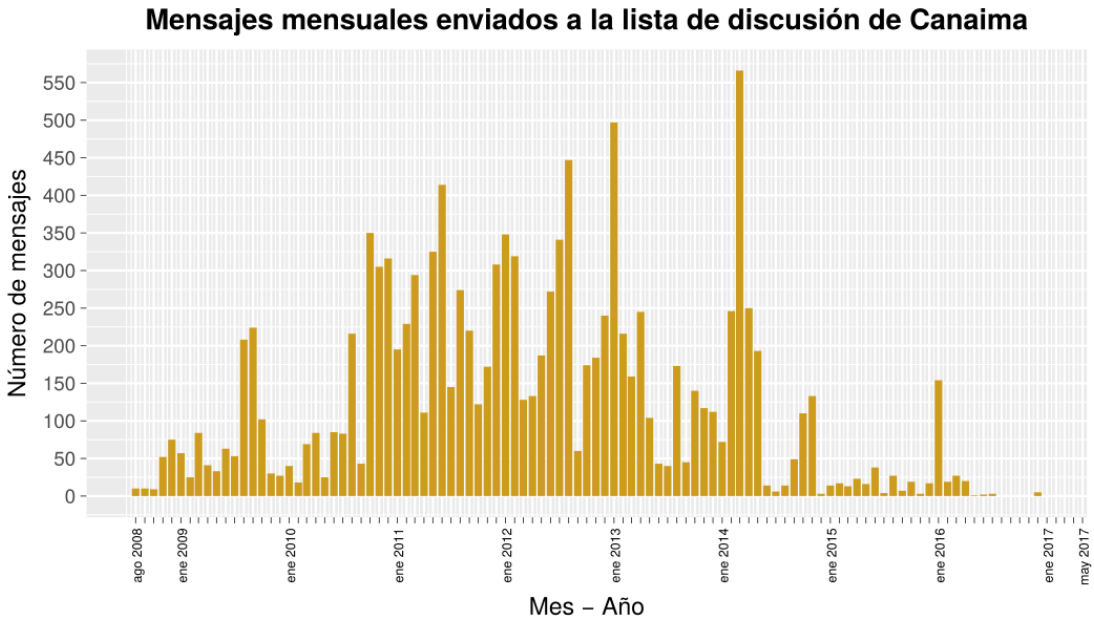
Histórico de distribuciones

- Canaima http://canaima.softwarelibre.gob.ve/descargas/canaima-gnu-linux/repositorio-de-distribuciones

Listas de discusión

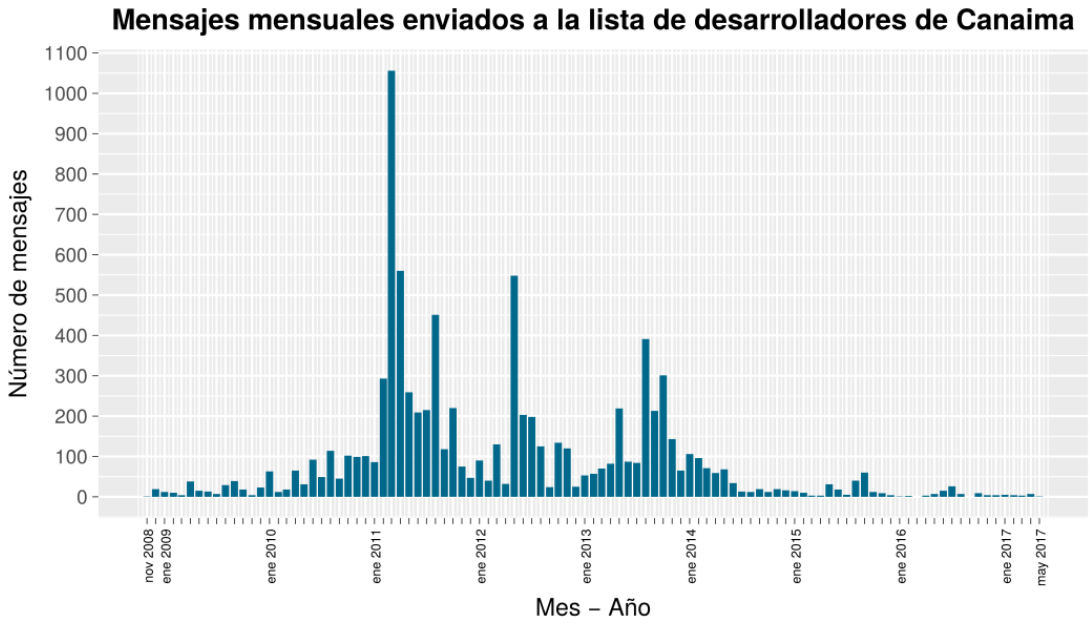
Lista de usuarios

Figura 3.11: Actividad de la lista de correos *Discusión* de Canaima



Lista de desarrolladores

Figura 3.12: Actividad de la lista de correos *Desarrolladores* de Canaima



Lanzamientos

La Tabla 3.10 muestra la fecha de lanzamiento de las diferentes versiones de la distribución Canaima GNU/Linux.

Tabla 3.10: Lanzamientos de versiones de Canaima

Versión	Nombre Clave	Fecha de Lanzamiento
1.0	Canaima	18/10/2007
2.0	Canaima	05/02/2009
2.0.1 RC1	Canaima	16/04/2009
2.0.1	Canaima	15/05/2009
2.0.2	Canaima	22/05/2009
2.0.3	Canaima	03/07/2009
2.0.4	Canaima	17/10/2009
2.1 RC	Canaima	21/05/2010
3.0 RC	Roraima	10/02/2011
3.0 RC2	Roraima	22/02/2011
3.0	Roraima	05/05/2011
3.1 VC1	Auyantepui	29/12/2011
3.1 VC2	Auyantepui	06/07/2012
3.1 VC3	Auyantepui	18/07/2012
3.1	Auyantepui	14/11/2012
4.0	Kerepakupai	04/12/2013
4.1	Kukenán	04/09/2014
5.0 VC1	Chimantá	23/12/2015
5.0	Chimantá	20/12/2016

Fuentes: Canaima GNU/Linux[28] y Wikipedia [29]

[28] http://canaima.softwarelibre.gob.ve
[29] http://es.wikipedia.org/wiki/Canaima_(distribuci%C3%B3n_Linux)

Estimación de costos

Componente *main*

- Fecha del archivo Source: 01/06/2016
- Número de paquetes descargados: 35
- Cantidad de lenguajes reconocidos: 19

Tabla 3.11: SLOC agrupados por lenguaje para Canaima

Lenguaje	**Cantidad de archivos**	**Líneas en blanco**	**Líneas con comentarios**	**Líneas de código fuente**
JavaScript	211	10.421	21.619	65.567
C#	341	9.921	10.647	44.215
CSS	40	4.709	4.055	44.060
Bourne Shell	40	4.420	5.700	31.785
Python	155	4.246	5.776	13.948
JSON	2	0	0	11.560
XML	21	265	1.659	11.082
m4	5	1.020	106	8.793
Glade	8	0	15	6.985
HTML	52	376	79	3.136
MSBuild script	6	0	14	1.515
make	33	350	233	1.303
Windows Resource File	11	212	0	1.108
Bourne Again Shell	58	760	633	1.101
Markdown	5	121	0	251
YAML	20	7	0	191
C	2	9	5	25
INI	3	0	0	22
D	1	0	0	1
Total	**1.014**	**36.837**	**50.541**	**246.648**

Figura 3.13: Porcentaje de archivos por lenguaje dentro del componente *usuarios* de Canaima

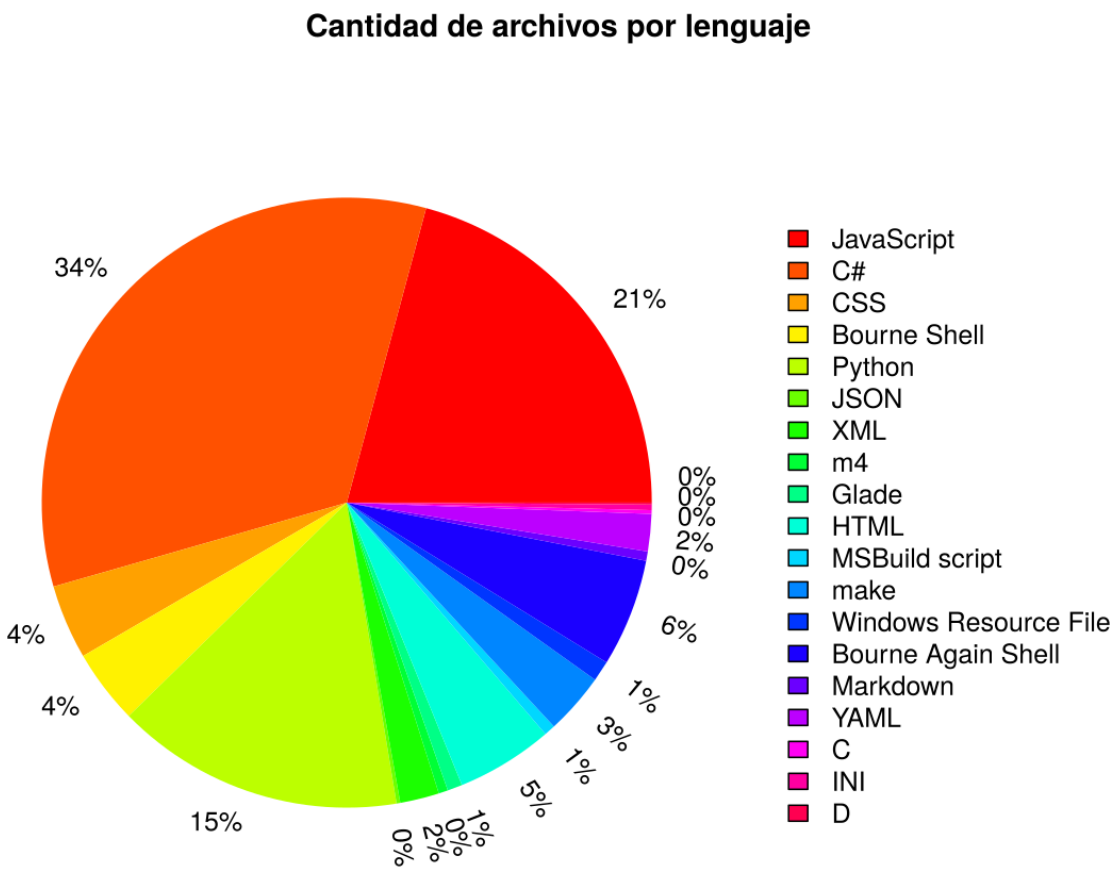
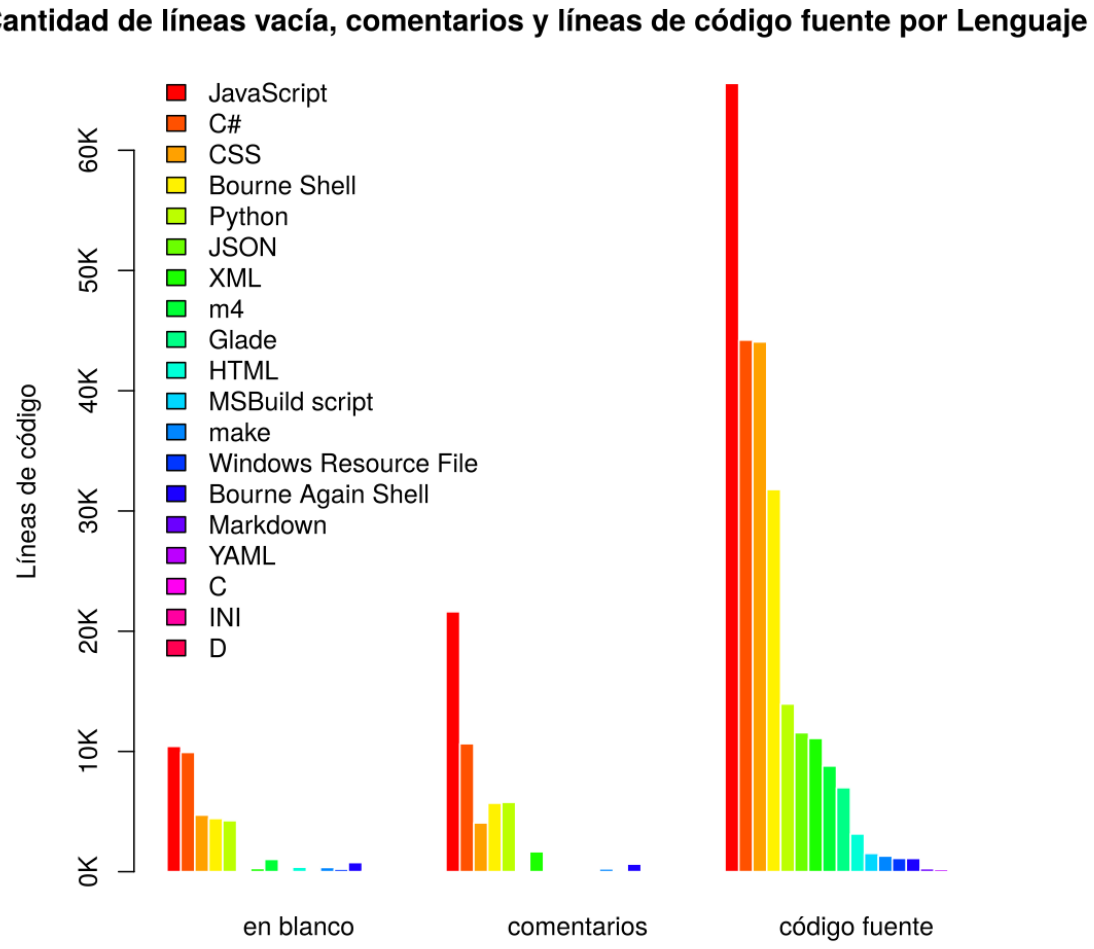


Figura 3.14: Cantidad de líneas vacía, comentarios y líneas de código fuente en los lenguajes del componente *usuarios* de Canaima



COCOMO

Tabla 3.12: Estimaciones de esfuerzo y costos para el componente *usuarios* de Canaima aplicando COCOMO Básico

Líneas de código fuente:	246.648
Esfuerzo estimado de desarrollo (persona–mes):	779,64
Productividad estimada:	0,32
Tiempo de desarrollo estimado (meses):	31,39
Personas requeridas estimadas (personas):	24,83
Costo total estimado del proyecto (US$):	1.833.612,33

Para el Costo Total estimado se toma el valor US$ 73.837,00 anual como salario de referencia para ingenieros de software, desarrolladores y programadores en enero de 2017[30].

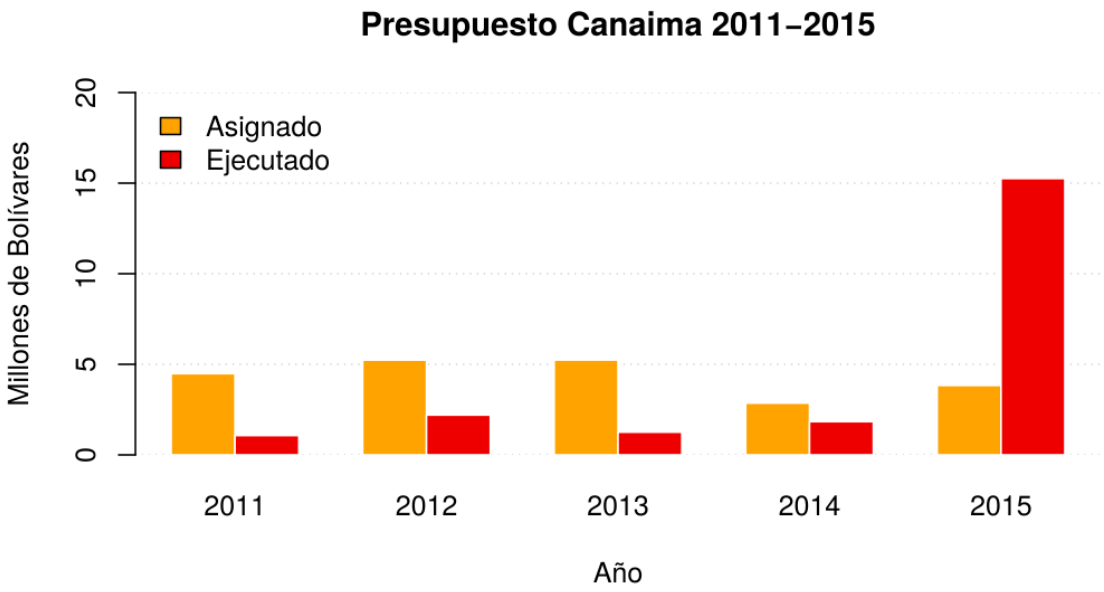
Presupuesto Canaima

Tomando datos de la página del Centro Nacional de Tecnologías de Información (CNTI)[31], se muestra en la figura 3.15 la relación del presupuesto asignado y el presupuesto ejecutado entre los años 2011 y 2015, ambos inclusive. Un dato interesante es que en el año 2015, según la información publicada por el CNTI, el total ejecutado es aproximadamente 4 veces el monto asignado para esta actividad, e históricamente, en promedio, el total ejecutado no llegaba a sobrepasar los 2 millones de bolívares.

[30] http://www.payscale.com/research/US/Job=Software_Engineer_%2f_Developer_%2f_Programmer/Salary#CareerPaths

[31] http://www.cnti.gob.ve/institucion/responsabilidad-social.html

Figura 3.15: Presupuesto asignado al desarrollo de la distribución Canaima GNU/Linux



El proyecto Gaudalinex, una distribución GNU/Linux con apoyo financiero de la Junta de Andalucía, España; en el año 2016[32] adjudicó y formalizó el contrato para el desarrollo de la distribución Guadalinex y su par empresarial GECOS (Guadalinex Escritorio COrporativo eStándar) por un monto total de €84.700,00[33].

[32] http://www.juntadeandalucia.es/contratacion/ContractFormalizationDetail.action?code=2016-0000008954

[33] Aquí se establece que la hora de un programador es de €21,78 (con IVA).

Capítulo 4

Estado de desarrollo y nivel de productividad de la Distribución Canaima GNU/Linux

---

En principio, para las 2 primeras versiones de Canaima, el modelo de desarrollo se ajustaba a los tiempos de desarrollo de Debian, pero no se ajustaban a los tiempos de la Administración Pública Nacional.

[Continua...]

A partir de la versión 3 se da inicio a una nueva etapa dentro del proyecto Canaima, esta versión marca el inicio de un modelo algo más endógeno, aquí el Estado pasa de ser un simple cliente que contrata un servicio para la construcción de una herramienta de software a un actor directo en los procesos, toma de decisiones y construcción de la distribución.

[Continua...]

“Canaima GNU/Linux posee un modelo de desarrollo basado en Debian, con algunas modificaciones para adaptarla a las necesidades propias de Venezuela.”

Figura 4.1: Ciclo de desarrollo de Canaima GNU/Linux

Fuente: http://canaima.softwarelibre.gob.ve/canaima/soporte

[Continua...]

2012 fue el último año donde se hicieron públicos los datos sobre el proceso de migración de software libre en la APN[1], los datos de ese año se alcanzaron a través de un censo realizado entre noviembre de 2012 y febrero de 2013, 126 instituciones de las 580 que conformaban para ese entonces la APN suministraron la información requerida, de allí se obtiene que el 51,14%, el equivalente a 70.871 de las máquinas

[1] http://www.cnti.gob.ve/til-venezuela/estadisticas-de-migracion/estadisticas-de-migracion.html

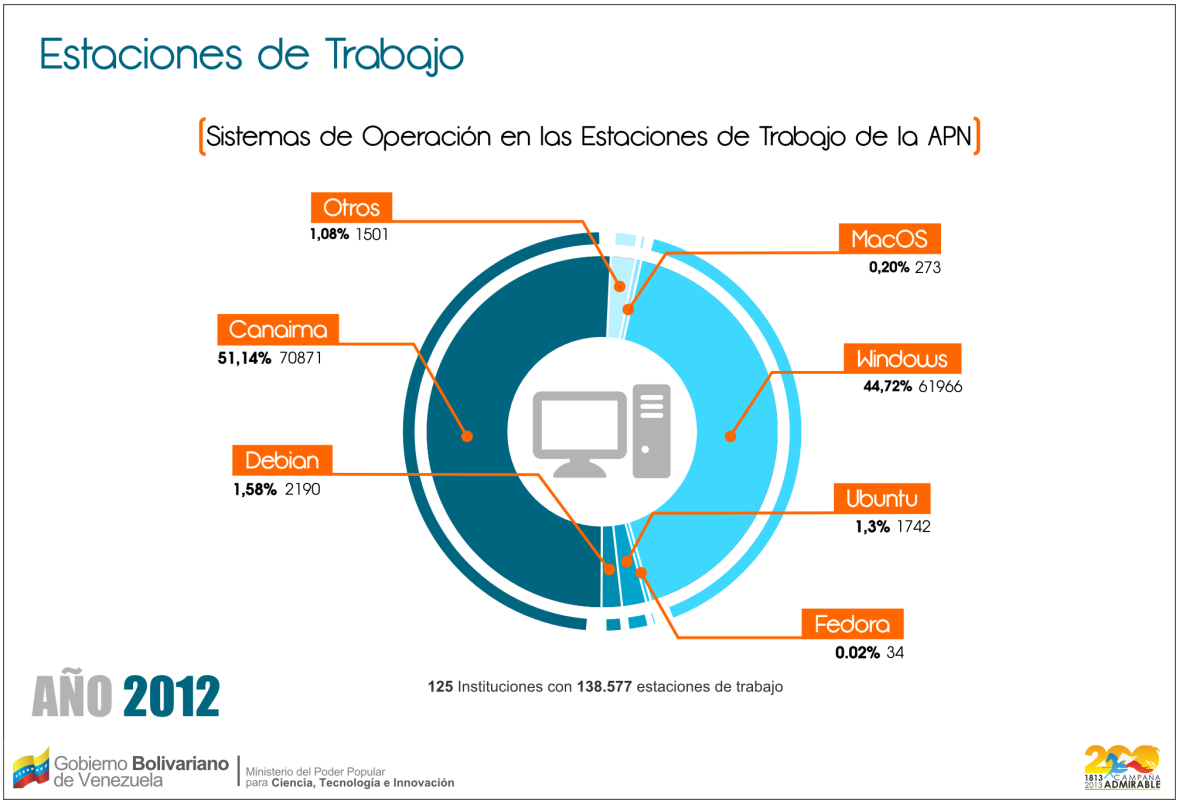
evaluadas, tenían instalado Canaima GNU/Linux, le seguía en cantidad el sistema operativo privativo Microsoft Windows con 44,72% equivalente a 61.966 máquinas (ver figura 4.2). Se debe acotar que el estudio no hace mención sobre estaciones de trabajo con múltiples sistemas instalados, por lo que cabe la posibilidad que varias de las máquinas con Linux instalado también dispusieran Windows y aquí cabría la pregunta, de haber maquinas con múltiples sistemas ¿cuál de los sistemas instalados era el sistema operativo de uso regular?

Para sacar estimaciones de número de máquinas que cuentan con la distribución Canaima GNU/Linux, se suele tomar como referencia el número de computadoras entregadas bajo el programa Canaima Educativo, éste consiste en dotar de una computadora portátil con contenido educativo a niños y maestros de educación primaria de escuelas públicas. Para diciembre de 2016 se habían entregado 5.177.000 computadoras Canaima en todo el país (Oria, 2016).

El problema en mantener esta afirmación es que el proyecto inició la entrega de computadoras en 2009, el Estado no añadió un sistema de seguimiento de hardware o software, por lo que no es posible en este momento saber algunos datos de interés estadístico para el Estado como el número exacto de computadoras operativas, tampoco se puede saber, de este número de máquinas operativas, cuántas aún conservan el software entregado originalmente, cuántas de estas le han actualizado el sistema o le han instalado otro sistema operativo (libre o privativo), cuántas aún están en manos de sus dueños originales, etc.

Por todo esto no es correcto alegar que Canaima “tiene la comunidad de Software Libre más grande del mundo con los más de 5 millones de usuarias y usuarias de todo el territorio nacional que tienen su canaimita.” (Hernández, 2016). A lo sumo se podría afirmar que se han entregado más de 5 millones de computadoras portátiles a estudiantes y docentes desde que inició el proyecto Canaima Educativo, pero no hay manera de asegurar que esa sea la cantidad de usuarios actuales.

Figura 4.2: Sistemas de operación en las estaciones de trabajo de la APN



Fuente: http://www.cnti.gob.ve/til-venezuela/estadisticas-de-migracion/estadisticas-de-migracion.html (Recuperado 28/03/2017)

Capítulo 5

Conclusiones y recomendaciones

Calcular la cantidad de usuarios a nivel mundial de un sistema operativo como Linux no es tarea fácil, y conocer cuál es la distribución Linux que usan es aún mas cuesta arriba, solamente podemos llegar a algunas estimaciones a través de ciertas técnicas.

Existen empresas que se dedican a obtener estadísticas detalladas de los visitantes de las páginas web de sus clientes, para ello les proveen de un pequeño script que debe ser insertado en la página web y este se encarga de obtener la información de los visitantes; cada vez que la página web es visitada el script recoge información de la computadora visitante, entre los datos recolectados se pueden mencionar: la dirección IP del visitante, país, cantidad de visitas previas a la página, el nombre del navegador, versión del navegador, tipo de dispositivo que visita la página (tableta, teléfono móvil, computadora de escritorio), resolución de pantalla y sistema operativo instalado en el dispositivo.

Algunas de estas compañías hacen público parte de estos datos y de allí la información en la tabla 5.1 que muestra el porcentaje de máquinas con Linux que han visitado alguna página monitorizada por estos sistemas.

Tabla 5.1: Cuota de mercado para mayo de 2017 de sistemas operativos basados en Linux instalados en computadoras de escritorio con acceso a internet.

Fuente de datos	**Cuota de mercado (%)**
W3Counter	2,64
Net Market Share	1,99
StatCounter	1,66
statista	1,53

Ahora, se puede hacer una estimación mínima de usuarios Linux, para ello se toma la cantidad 3.739.698.500[1] como el número base de usuarios de internet para marzo de 2017, y usando los valores máximos y mínimos de la tabla 5.1 se obtiene que existen aproximadamente, al menos, entre 57.217.387 y 98.728.040 usuarios con acceso a internet que usan alguna distribución Linux para escritorio.

[Continúa]

Antes que todo, debe tenerse en cuenta que Canaima GNU/Linux es una distribución cuyo soporte financiero, técnico y humano proviene de un Estado, Debian, en cambio, es una distribución cuyo soporte proviene principalmente de la comunidad de desarrolladores y usuarios, finalmente, Ubuntu es una distribución patrocinada principalmente por una empresa, Canonical. Dicho esto, el interés que mueve a cada proyecto puede variar, puede ir desde lo estratégico a lo meramente económico.

[Continúa]

[1]Dato recuperado de `http://www.internetworldstats.com/stats.htm` en mayo de 2017

Apéndices

Códigos fuentes

Script para descargar paquetes fuente

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

# Copyright 2016 David Hernández
#
#
# This software is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This software is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with paquetes.py. If not, see <http://www.gnu.org/licenses/>.

""" Programa que lee los paquetes de un archivo Source y descarga el archivo
    fuente de cada uno de estos paquetes.
"""

# Carga de bibliotecas
import string
import subprocess
import glob

# Carga el archivo "Sources", este debe estar en el directorio de corrida
```

```
try:
    f = open('Sources', 'r')
except IOError:
    print "Error al abrir el archivo ", arg
else:
    #contador de paquetes procesados
    cont=0
    #Evalúa cada línea de Sources
    for line in f:
        # Si en la línea encuentra la palabra "Binary: " almacena los nombres
    de
        # cada paquete encontrado en la línea dentro de la lista "binarios"
        if 'Binary: ' in line:
            binarios = string.split(line)[1:]
            # Descarga las fuentes si no existen ya, y borra archivos no
            # necesarios.
            for paquete in binarios:
                paquetes = string.strip(paquete,',')
                if (glob.glob(paquetes + '_*')):
                    break;
                subprocess.call("apt-get --download-only source " + paquetes,
                                shell=True)
                subprocess.call("rm *.dsc " + "> /dev/null 2>&1", shell=True)
                subprocess.call("rm *.debian.tar.* " + "> /dev/null 2>&1",
                                shell=True)
                subprocess.call("rm *.diff.gz " + "> /dev/null 2>&1",
                                shell=True)
                cont = cont + 1
    f.close()

    print
    print "Cantidad de paquetes procesados: " + str(cont)
```

Código 1: paquetes.py

Script para ordenar y descomprimir paquetes fuentes

```
#!/usr/bin/env bash

# Copyright 2017 David Hernández
#
#
# This software is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
```

```
# (at your option) any later version.
#
# This software is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with mkdirmv. If not, see <http://www.gnu.org/licenses/>.

# Programa que crea un directorio por cada archivo comprimido, allí  descomprime
# este archivo y limpia el directorio raí z.


for file in *.*; do
  directorio="$(basename "${file}")"
  mkdir "_${directorio}"
  mv "$file" "_${directorio}"
  cd "_${directorio}"
    if [ ${file: -3} == ".gz" ]; then
        gzip -t *
        gunzip *
        tar xfi *
        rm *.tar
    fi
    if [ ${file: -3} == ".xz" ]; then
        tar xfi *
        rm "$file"
    fi
    if [ ${file: -4} == ".bz2" ]; then
        bzip2 -dv *
        tar xfi *
        rm *.tar
    fi
    cd ..
done
```

Código 2: mkdirmv.sh

Script para correr `cloc` en el caso Debian

```
#!/usr/bin/env bash

# Copyright 2017 David Hernández
#
```

```
#
# This software is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This software is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with cloc_recargado. If not, see <http://www.gnu.org/licenses/>.

# Programa para correr cloc individualmente en cada subdirectorio de un
# directorio dado. Genera un reporte por cada subdirectorio.

# Para cada subdirectorio dentro del directorio dado se crearí¡ un reporte
# por el programa cloc y estos serí¡n almacenados dentro del directorio
# ./debiancloc. Si ya existe un reporte previo, entonces se ignora el aní¡lisis
# para ese subdirectorio.

for i in $(realpath "$1"/*); do
    if [ ! -f debiancloc/$(basename "$i").csv ]; then
        echo File: "$i"
        cloc --use-sloccount --exclude-ext=po --csv --report-file=debiancloc/$(basename "$i").csv "$i"
        echo
    else
        echo File $(basename "$i").csv already exist
        echo
        continue
    fi
done
```

Código 3: cloc_recargado.sh

Script para el análisis de los datos generados por el script `cloc_recargado.sh`

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

# Copyright 2017 David Herní¡ndez
```

```
#
#
# This software is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This software is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with analisis_cloc_debian If not, see <http://www.gnu.org/licenses/>.

""" Programa que se encarga de analizar los archivos csv generados por
    analisis_cloc_debian. Agrupa y suma los lenguajes detectados en los
    diferentes paquetes y genera un archivo csv con los resultados.
"""

# Carga de bibliotecas
import glob
import pandas as pd

# Lista de archivos a analizar
archivos = glob.glob('debiancloc/*.csv')

# Se inicializa el dataframe
df = pd.DataFrame()

# Se agrega al dataframe los datos de los archivos importados
for archivo in archivos:
    df = df.append(pd.read_csv(archivo, usecols=[0,1,2,3,4]))

# Se agrupan los lenguajes y se suman
final = df.groupby(df.language).sum()

# Se reinicia el índice
final = final.reset_index()

# Se reordenan las columnas el dataframe
final = final[['files','language','blank','comment','code']]

# Se ordenan los datos segun la cantidad de codigo
final = final.sort_values("code",ascending=False)

# Se genera el csv final
```

```
final.to_csv('debiancloc.csv',index=False)
```

Código 4: analisis_cloc_debian.py

Script para calcular las estimaciones de esfuerzo

```
#! /usr/bin/env Rscript

# Copyright 2017 David Hernández
#
#
# This software is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This software is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with esfuerzo.R. If not, see <http://www.gnu.org/licenses/>.

# Cálculo de estimaciones aplicando el Modelo COCOMO Básico

library(tcltk)

#Filtro de extensiones con las que se trabajará en este script
Filters   <- matrix(c("csv",".csv"),1, 2, byrow = TRUE)

#Cargar el archivo de entrada
entrada <- tk_choose.files(default = "", caption = "Seleccione el archivo de
    entrada",multi = FALSE, filters = Filters, index = 1)

cloc <- read.csv(entrada, header=TRUE)

a <- 2.40
b <- 1.05
c <- 2.5
d <- 0.38
KSLOC <- sum(cloc$code)/1000
Spa <- 73837
Epm <- a*(KSLOC)^b
Prod <- KSLOC/Epm
```

```
39 Tdev <- c*(Epm)^d
40 Per <- Epm/Tdev
41 Ctd <- Per*Spa
42
43 cat("\n\n")
44 cat("Líneas de código fuente: \t", format(sum(cloc$code), digits=2, nsmall=2,
       decimal.mark=",", big.mark=".", scientific=FALSE),"\n")
45 cat("Líneas de código fuente (K): \t", format(KSLOC, digits=2, nsmall=2, decimal
       .mark=",", big.mark=".", scientific=FALSE),"\n\n\n")
46 cat("Estimaciones aplicando el Modelo COCOMO Básico\n")
47 cat("============================================================\n")
48 cat("Esfuerzo estimado de desarrollo:\t", format(Epm, digits=2, nsmall=2,
       decimal.mark=",", big.mark=".", scientific=FALSE),"\n")
49 cat("Productividad estimada:\t\t\t", format(Prod, digits=2, nsmall=2, decimal.
       mark=",", big.mark=".", scientific=FALSE),"\n")
50 cat("Tiempo de desarrollo estimado:\t\t", format(Tdev, digits=2, nsmall=2,
       decimal.mark=",", big.mark=".", scientific=FALSE),"\n")
51 cat("Personas requeridas estimadas:\t\t", format(Per, digits=2, nsmall=2,
       decimal.mark=",", big.mark=".", scientific=FALSE),"\n")
52 cat("\nCosto total estimado del proyecto(*):\t", format(Ctd, digits=2, nsmall=2,
        decimal.mark=",", big.mark=".", scientific=FALSE), "\n\n")
53 cat("(*) con US$", format(Spa, digits=2, nsmall=2, decimal.mark=",", big.mark=".
       ", scientific=FALSE), "anual como referencia para enero de 2017\n")
54 cat("============================================================\n\n\n")
```

Código 5: esfuerzo.R

Referencias

Amor, J. J., González, J. M., Robles, G., y Herráiz, I. (2005a). Debian 3.1 (sarge) como caso de estudio de medición de software libre: resultados preliminares. *Novática*, S/V(175):11–14.

Amor, J. J., Robles, G., y González, J. M. (2005b). GNOME como Caso de Estudio de Ingeniería del Software Libre. Recuperado de https://dramor-research-files.firebaseapp.com/research/papers/2005-guadeces-amor-robles-barahona.pdf.

Amor, J. J., Robles, G., y González-Barahona, J. M. (2004). Measuring Woody: The Size of Debian 3.0. *Reports on Systems and Communications*, V(10).

Amor, J. J., Robles, G., González-Barahona, J. M., y Peña, J. F.-S. (2007). Measuring Etch: the size of Debian 4.0.

Amor, J. J., Robles, G., M., J., González-Barahona, y Rivas, F. (2009). Measuring Lenny: the size of Debian 5.0.

Boehm, B. (1981). *Software Engineering Economics*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1a edición.

Capiluppi, A. y Michlmayr, M. (2007). De la catedral al bazar: un estudio empírico del ciclo de vida de los proyectos basados en comunidades de voluntarios. *Novática*, S/V(189):9–16.

Commons, W. (2017). File:linux distribution timeline.svg — wikimedia commons, the free media repository. Recuperado de https://commons.wikimedia.org/

w/index.php?title=File:Linux_Distribution_Timeline.svg&oldid=256252861. [Online; visitada 01-julio-2017].

Equipo de desarrollo de Canaima (2009). Canaima un nuevo paradigma socio-tecnológico. *laTItud*, S/V(2):23.

Figuera, C. (2013). El Proyecto Canaima. *Revista de Tecnología de Información y Comunicación en Educación*, 7(Especial):197–212.

Gómez, A., del C. López, M., Migani, S., y Otazú, A. (2000). COCOMO. Un modelo de estimación de proyectos de software. Recuperado de https://www.academia.edu/4853590/UN_MODELO_DE_ESTIMACION_DE_PROYECTOS_DE_SOFTWARE.

González, J. M., Ortuño, M. A., de las Heras Quirós, P., Centeno, J., y Matellán, V. (2001). Contando patatas: el tamaño de Debian 2.2. *Novática*, 2(6):30–37.

González, J. M., Robles, G., Ortuño-Pérez, M., Rodero-Merino, L., Centeno-González, J., Matellán-Olivera, V., Castro-Barbero, E., y de-las Heras-Quirós, P. (2003). Analyzing the anatomy of GNU/Linux distributions: methodology and case studies (Red Hat and Debian).

Hernández, E. (2016). Kenny Ossa: Canaima GNU/Linux tiene la comunidad más grande del mundo. Centro Nacional de Tecnologías de Información. Recuperado de http://www.cnti.gob.ve/noticias/actualidad/cnti/5620-kenny-ossa-canaima-gnu-linux-tiene-la-comunidad-de-usuarios-mas-grande-del-mundo.html.

Herraiz, I., Robles, G., Gonzalez-Barahona, J. M., Capiluppi, A., y Ramil, J. F. (2006). Comparison between SLOCs and Number of Files as Size Metrics for Software Evolution Analysis. In *Proc. European Conf. Software Maintenance and Reengineering (CSMR)*.

Indian Institutes of Technology Kharagpur (2006). Software Project Planning. COCOMO Model. Curso en lÃnea. Recuperado de http://nptel.ac.in/courses/106105087/pdf/m11L28.pdf.

Institute of Electrical and Electronics Engineers (1997). IEEE Standard for Developing Software Life Cycle Processes.

Instituto Nacional de Tecnologías de la Comunicación (2009). *Ingeniería del software: metodologías y ciclos de vida*.

International Organization for Standardization (2008). ISO/IEC 12207:2008(E). Systems and software engineering – Software life cycle processes. Recuperado de http://www.iso.org/iso/catalogue_detail?csnumber=43447.

Lowe, W., Schulze, M., y Garbee, B. (2015). A Brief History of Debian [Una breve historia de Debian]. Recuperado de https://www.debian.org/doc/manuals/project-history/project-history.en.pdf.

Mardones, K. (2008). Gestión de riesgos para el sistema de simulación de redes de gasoducto para PDVSA.

Martínez, L. (2012). Re: [Discusion] Que Día Mes Y Año Nació Canaima GNU/Linux. Correo Electrónico. Recuperado de: http://listas.canaima.softwarelibre.gob.ve/pipermail/discusion/2012-August/007569.html. Consultado el 24 de marzo de 2017.

Oria, G. (2016). Entregadas 5.177.000 computadoras canaima a niños y jóvenes de todo el país. Últimas Noticias. Recuperado de http://www.ultimasnoticias.com.ve/noticias/comunidad/entregadas-5-177-000-computadoras-canaima-ninos-jovenes-pais/.

Pressman, R. (2010). *Ingeniería del Software. Un enfoque prÃictico*. México, 7ma edición edición.

Raymond, E. (1999). *The Cathedral and the Bazaar*. O'Reilly & Associates, Sebastopol, CA, EEUU.

Rentería, M. A. (2012). Módulo multiagente de mejora de procesos para el desarrollo de software.

República Bolivariana de Venezuela (2004). Decreto 3.390. Gaceta Oficial Nro. 38.095 del 28/12/2014. Recuperado de http://historico.tsj.gob.ve/gaceta/diciembre/281204/281204-38095-08.html. Consultado el 24 de marzo de 2017.

Somerville, I. (2005). *Ingeniería del Software*. Madrid, 7ma edición edición.

The Ubuntu Manual Team (2016). *Getting Started with Ubuntu 16.04*. S/P.

Wheeler, D. A. (2001). More than a Gigabuck: Estimating GNU/Linux's Size. Recuperado de https://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html. Consultado en noviembre 2016.