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

Diciembre 2017

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

David A. Hernández Aponte

Proyecto de Grado — Sistemas Computacionales, 116 páginas

Resumen: Este proyecto evalúa el esfuerzo asociado o asociable a una distribución de Software Libre o distro. Para ello, se hace un estudio de los paquetes de software distribuidos en el repositorio principal de 3 distribuciones Linux: Debian, Ubuntu y Canaima. Se presentan datos referenciales de las tres distribuciones y se revisa la participación de la comunidad de usuarios en los procesos del ciclo de vida de la distribución Canaima GNU/Linux. De ello se desprende la importancia que tiene la participación de las comunidades de usuarios para lograr mantener la buena salud de un proyecto de Software Libre.

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 trabajo busca destacar la importancia que tiene la comunidad de usuarios en los procesos que intervienen para el desarrollo de una distribución Linux. Para llegar a ello se evalúa el esfuerzo requerido para el desarrollo de proyectos mediante la aplicación del Modelo Constructivo de Costos. Además, 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

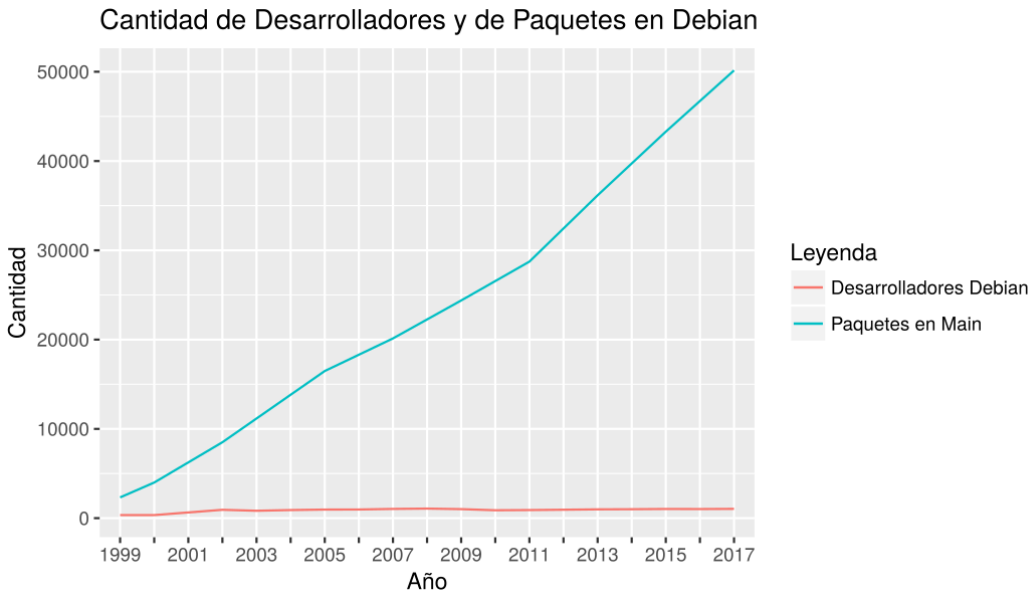
sistema, producto, servicio, proyecto u otra entidad concebida por humanos, desde su concepción hasta su retiro (International Organization for Standardization, 2008). También se explica el *modelo de ciclo de vida* como la descripción de procesos y actividades relacionadas con el ciclo de vida que pueden 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 (2001); González et al. (2001, 2003); Amor et al. (2004, 2005a,b, 2007, 2009); Herraiz et al. (2006) y otros autores.

Por último, es importante recordar el trabajo realizado en Venezuela previo a Canaima en materia de distribuciones y metadistribuciones GNU/Linux. Quizás el ejemplo más sobresaliente fue el nacido desde la Universidad de Los Andes con el Proyecto ULAnux/ULAnix. Los inicios de ULAnux datan de 2004 y la primera versión beta de ULAnix nace en 2005. En el año 2006 es presentada la tercera versión beta de ULAnix en el 2º Congreso Nacional de Software Libre junto a la metadistribución ULAnux[5]. "ULAnux es una metadistribución libre de software libre, con la cual producimos ULAnix la distribución de software de la Universidad de Los Andes" (Molina et al., 2007). La distribución presentaba elementos innovadores: "Esta distribución es la primera creada en el ambiente universitario de Venezuela. También es la primera distribución venezolana disponible por internet" (Wikipedia, 2008). De ella se derivaron otras variantes que valen destacar:

ULAnix Lógica fue la primera versión específica de ULAnix, desarrollada como herramienta de ayuda en las áreas de lógica y matemática (Equipo ULAnux, 2007b).

Bio-ULAnix con aplicaciones para la programación y cálculo científico para el área de Bio-informática (Equipo ULAnux, 2007a).

ULAnix Scientia versión específica de ULAnix dirigida a estudiantes y docentes

[5] http://web.archive.org/web/20061004001640/http://www.cnsl.org.ve/Merida

de las áreas de ciencia y tecnología. Es descrita por Dávila et al. (2007) como “software científico a la medida de los investigadores que requieren herramientas estadísticas, matemáticas y de computación”.

ULAnix Ars fue la última versión específica de ULAnix, esta vez dirigida al área del diseño gráfico.

Planteamiento del Problema

En el software libre el modelo participativo es vital para el éxito y supervivencia 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.

A casi 10 años de la publicación de la primera versión ¿En que medida el esfuerzo invertido en el desarrollo de Canaima ha sido influenciado por la comunidad de usuarios?

La presente investigación está orientada a mostrar por una parte, el esfuerzo requerido para el desarrollo de Canaima GNU/Linux comparándola con otras distribuciones de referencia; y por la otra, presentar la facilidad o dificultad que tiene la misma para recibir contribuciones de parte de la comunidad de desarrolladores; y con ello formular algunas recomendaciones para mejorar el trabajo colaborativo en torno a la distribución Canaima GNU/Linux.

Hipótesis

Las formas de conducción del Proyecto Canaima GNU/Linux bajo las dinámicas de una institución gubernamental venezolana inciden directamente en las contribuciones que pueda recibir de parte de la comunidad.

Objetivos

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

Objetivo General

- Evaluar el esfuerzo estimado para el desarrollo de la metadistribución de Software Libre Canaima GNU/Linux.

Objetivos Específicos

1. Caracterizar el ciclo de vida del software libre de la distribución Canaima.
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 costos y esfuerzo.

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, hipótesis 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 más 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. Para la fecha en que inicia este trabajo, las versiones de las distribuciones estudiadas son: Debian Jessie (8.0), Ubuntu Xenial Xerus (16.04) y Canaima Chimantá (5.0).

Las consideraciones tomadas para la selección de estas distribuciones radican principalmente en que parten de la misma rama. Debian es la distribución madre de Ubuntu y de Canaima, esto significa que las distribuciones son derivaciones directas de Debian, se basan en ella para su construcción y por ello comparten algunas características: 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 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).

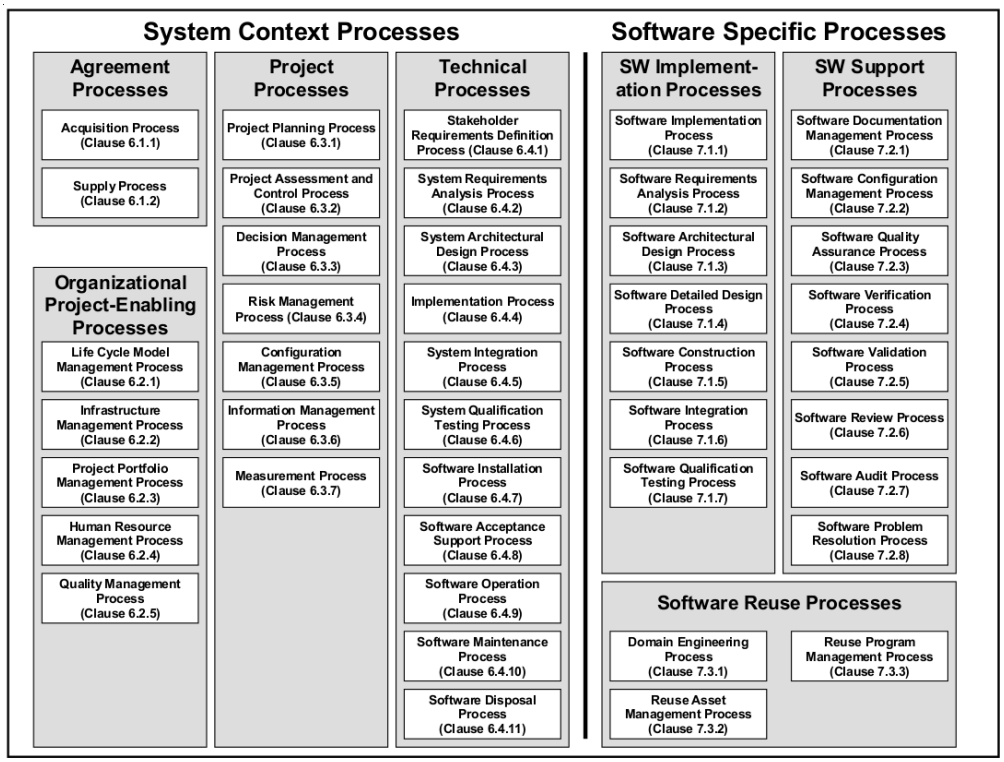
Como se dijo en el capítulo anterior, la vida de un sistema o de un producto de software puede ser modelado a través de un modelo de ciclo de vida basado en fases. Los modelos pueden ser usados para representar toda su vida, desde la concepción hasta su retiro, o la representación de una porción de vida 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).

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

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

[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/

`-csv`: es el argumento que indica al programa que el resultado se debe presentar con 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 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 (Wikimedia 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 más de 43.000 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 Desarrolladores nombrados mediante una Resolución General o elección. Ver Figura 3.1.

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

El Líder del Proyecto (DLP) es el representante oficial del Proyecto Debian. El cargo tiene validez por un año. Cualquier Desarrollador Debian tiene derecho a postularse 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 existiere algún miembro que ha servido por más de 42 meses y además sea uno de los dos miembros más antiguos, entonces su nombramiento expiraría 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 en conjunto con 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.

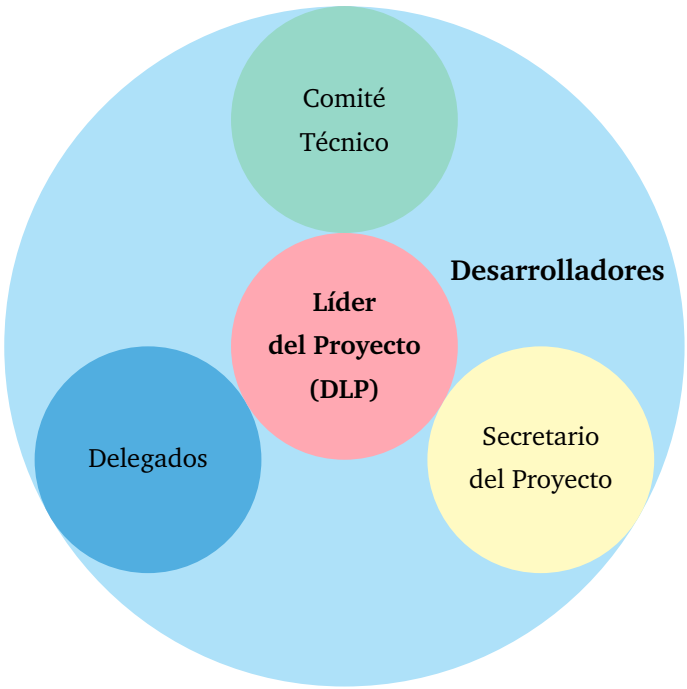
Los Delegados nombrados por el Líder del Proyecto para tareas específicas, son nombrados por el Líder del Proyecto y pueden ser reemplazados a discreción del mismo.

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 pueda 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 hayan sido 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 1.300 colaboradores del proyecto Debian entre Mantenedores Debian y Desarrolladores Debian.

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

[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]

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]

En la tabla se distinguen dos etapas, la primera se presenta 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.

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

Histórico de distribuciones

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/.

Arquitecturas

Debian soporta las arquitecturas[8] 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[9], DistroWatch[10]

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[11] para el funcionamiento de la distribución en otros sistemas como GNU/Hurd y FreeBSD.

[8] Aplica para Debian versión 8.0 (Jessie)
[9] https://www.debian.org/ports/index.es.html#portlist-released
[10] https://distrowatch.com/table.php?distribution=debian

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

Soporte

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*): Pruebas contiene paquetes que aún no han sido aceptados dentro de la versión estable. Este repositorio será la próxima versión estable.

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.

Desde la versión 6 de Debian (Squeeze) existe un proyecto 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)[12] 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.

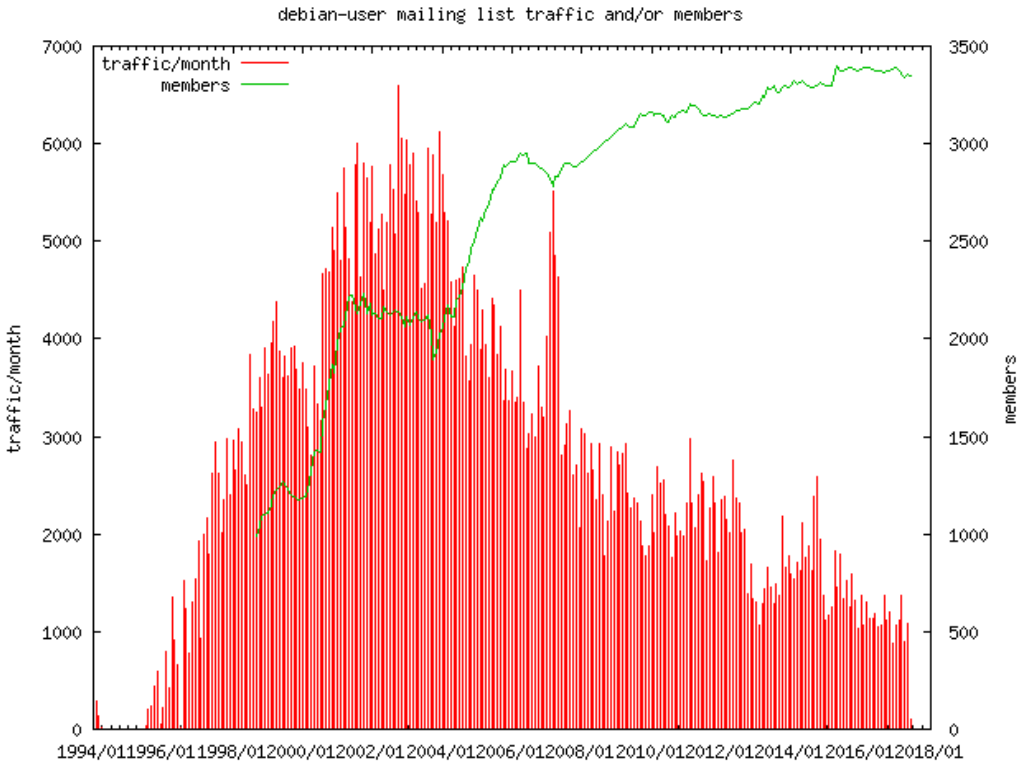
En este trabajo se cuantifica la actividad en dos listas públicas de correos específicas para cada distribución, la lista de discusión de usuarios y la lista de desarrolladores. Esto, con el objeto de mostrar el movimiento y la interacción de la comunidad que integra dichas listas. Las listas de usuarios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

[12] 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[13]. 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



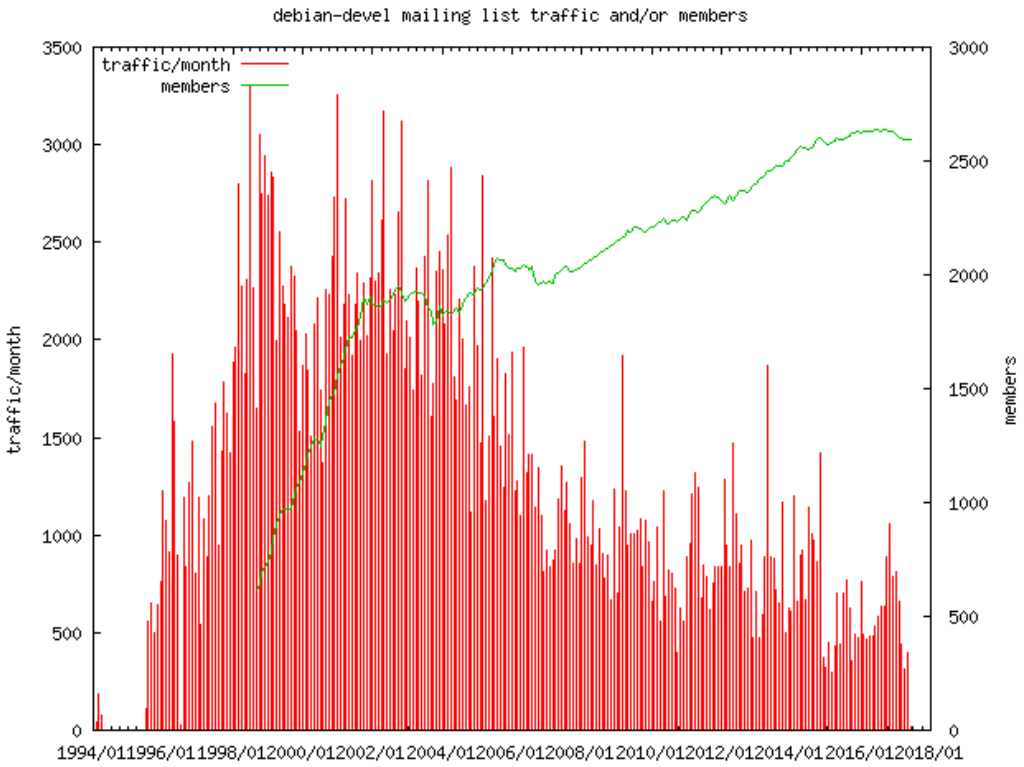
Fuente: https://lists.debian.org/debian-user/ (Recuperado 06/06/2017)

[13] 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 3.500 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 desarrollo

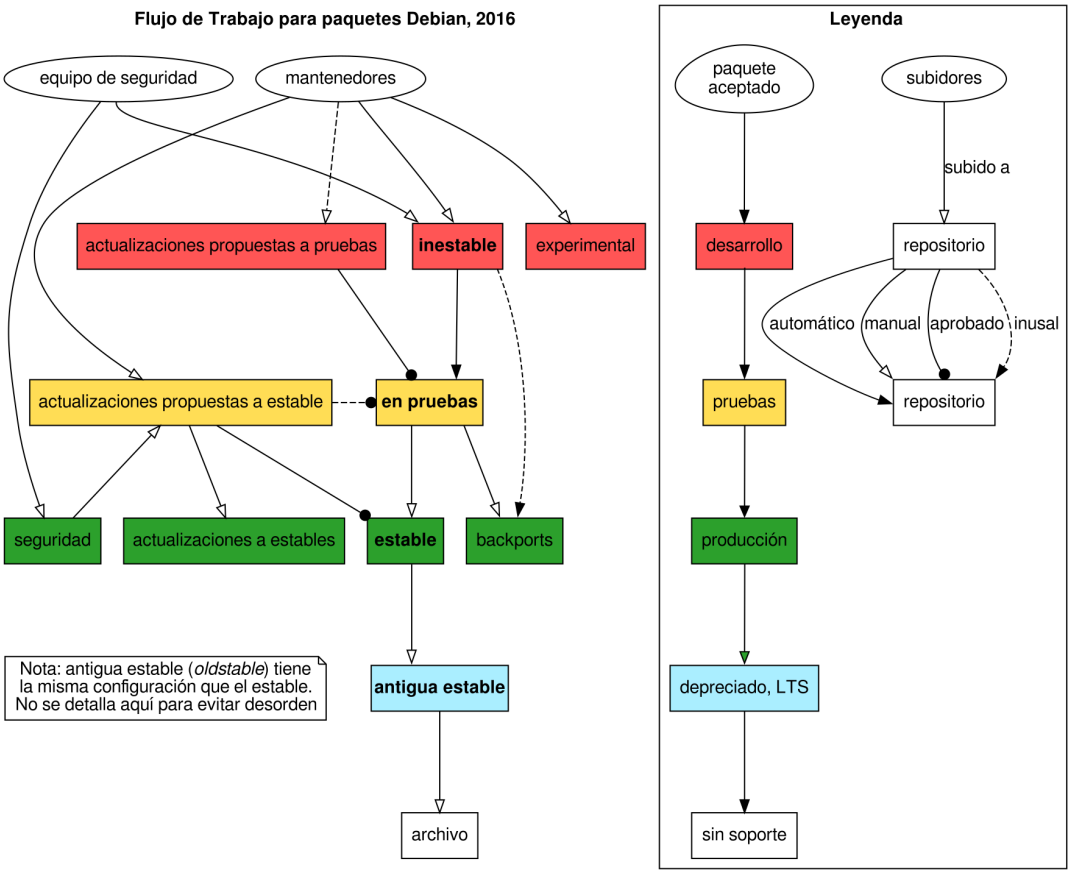
En la figura 3.4 se presenta el diagrama de flujo de paquetes entre las diferentes versiones de Debian.

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[14]

[14] 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 8.0. 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**

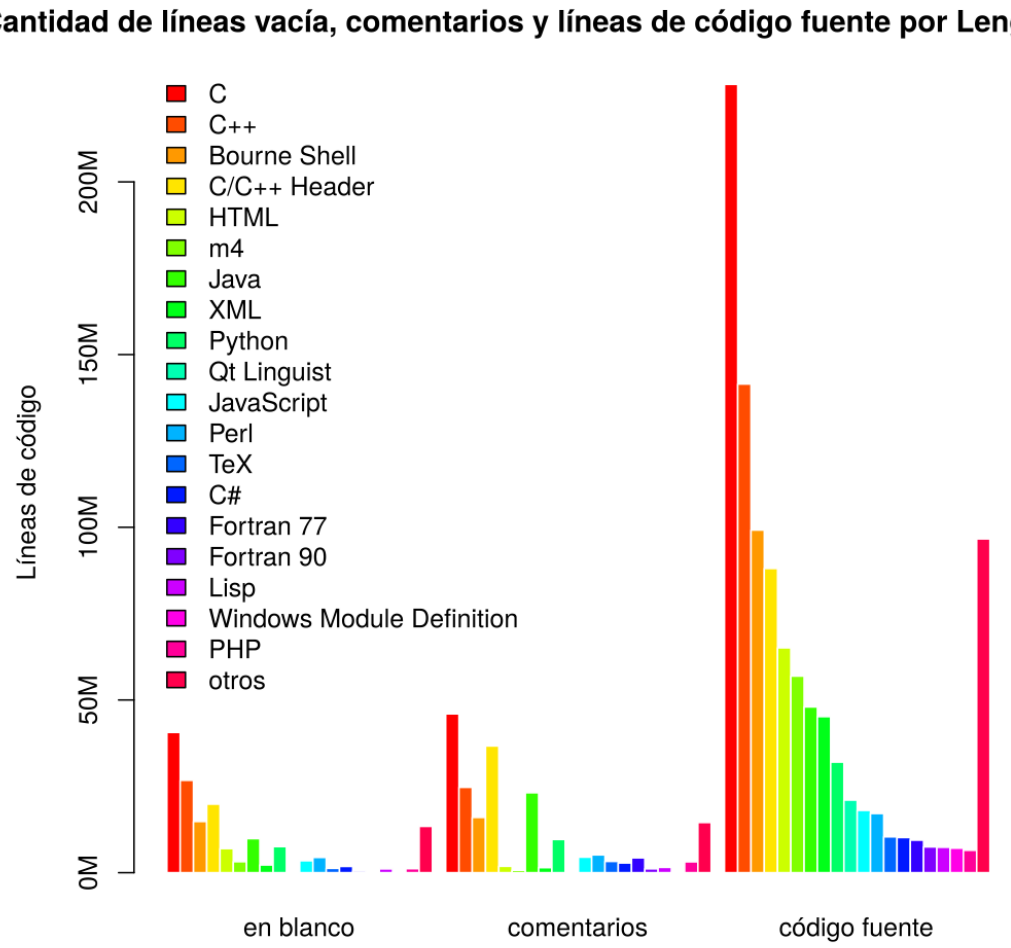
Fecha del archivo Source: 06/05/2017

Número de paquetes descargados: 20.981

Cantidad de lenguajes reconocidos: 179

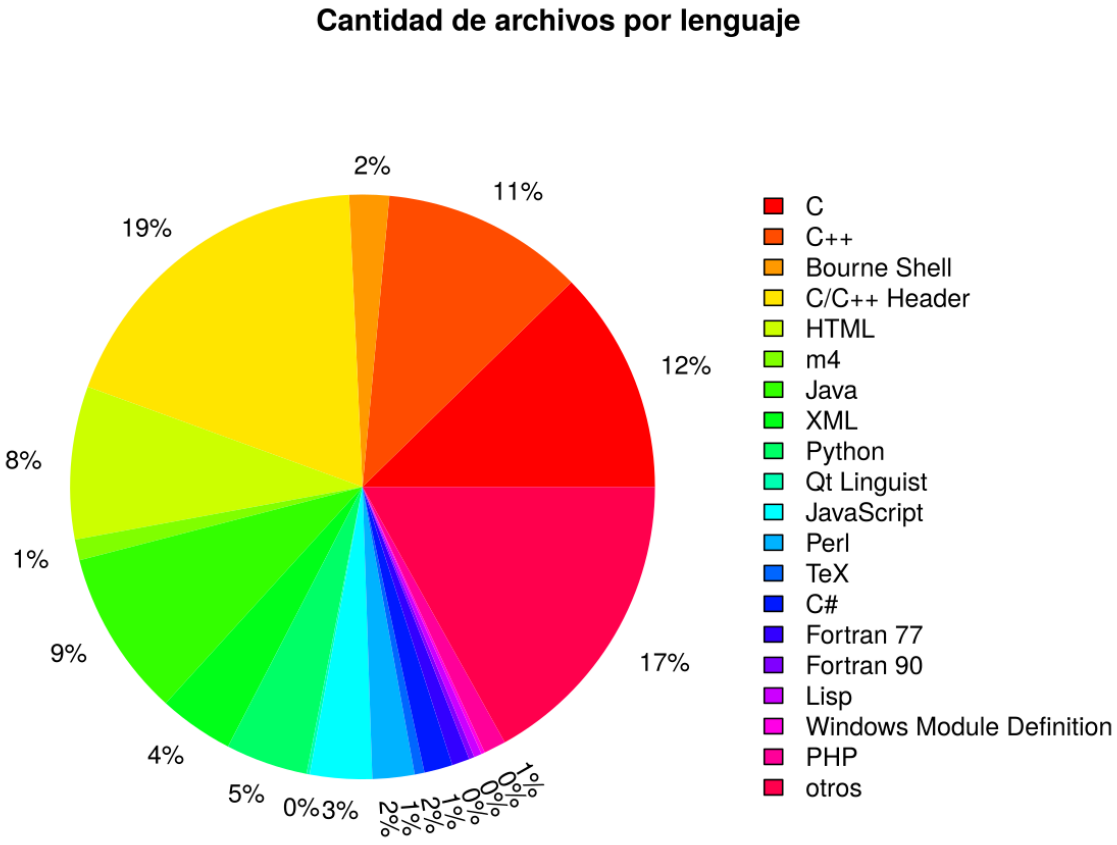
Se aprecia como el lenguaje C domina en cantidad de líneas en blanco, líneas con comentarios y líneas de código fuente. La Figura 3.5 visualiza los resultados de este análisis.

Figura 3.5: Cantidad de líneas vacías, comentarios y líneas de código fuente en los lenguajes del componente *main* de Debian 8.0



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 8.0



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 8.0 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[15].

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 5.600 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*[16] de Ubuntu afirma que, para mayo de 2017, existían más de 40 millones de usuarios de escritorio.

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

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

El soporte comercial de Ubuntu proviene de la empresa Canonical[17], 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[18].

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[19] que sirve como carta de compromiso entre los miembros de Ubuntu para la interacción y comportamiento de los mismos dentro de cualquier espacio relacionado con el proyecto. 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.

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

Organización

La comunidad Ubuntu está integrada por cientos de colaboradores en todos los ámbitos: traducción, arte, desarrollo, etc. Su organización se basa en el trabajo colaborativo por equipos, la estructura gira en torno a ellos. Como se mencionó anteriormente, todo colaborador con el proyecto debe seguir el Código de Conducta. La firma del Código de Conducta es el requisito principal para pertenecer a cualquier equipo de trabajo y así poder colaborar en la comunidad.

A continuación se describe la estructura que conforma la forma de organización del proyecto Ubuntu y se esquematiza en la Figura 3.7.

El Ubuntero es aquel individuo que se ha comprometido a seguir los lineamientos de Ubuntu a través de la firma del Código de Conducta. Es el primer paso para empezar a trabajar dentro de la comunidad y dentro del proyecto como tal.

Equipos de Trabajo. En Ubuntu los ubunteros se pueden incorporar a distintos grupos de trabajo. Es recomendable, al iniciarse como ubuntero, adherirse al equipo de trabajo local. Los Equipos Locales Comunitarios *Local Community teams*, conocidos en la comunidad Ubuntu como *LoCo Teams* en inglés o equipo local/regional en español, son los equipos que están destinados a agrupar a los colaboradores regionales, geográficamente hablando. Y desde allí organizarse para trabajar en la difusión, soporte, representación y mejoramiento de Ubuntu desde la localidad. Al día de hoy existen al menos 203[20] equipos regionales.

Otros equipos de trabajo[21] coexisten para encontrar esfuerzos en áreas específicas. Están los equipos encargados de temas técnicos (núcleo o *kernel* en inglés, escritorio de Ubuntu, seguridad, etc.). De igual forma también se hallan equipos de diseño; equipos enfocados en el aspecto comunitario y comunicacional (accesibilidad, noticias, wiki, planeta, etc.); soporte y documentación; mercadeo; y una gran variedad de equipos más.

[20] http://loco.ubuntu.com/teams/
[21] https://wiki.ubuntu.com/Teams/

Los Miembros Ubuntu (*Ubuntu Members*) son los miembros oficiales del proyecto. Para optar a este nombramiento se debe presentar soporte del trabajo realizado en el último año en favor de la comunidad y de la distribución Ubuntu. Estos soporte no son exclusivos para actividades de desarrollo de software, también se consideran actividades de divulgación como presentación de charlas, organización de eventos, traducciones, participación en foros de ayuda, diseño, soporte, y más.

Al obtener la membresía oficial, se adquieren privilegios de votación para confirmar las nominaciones del Consejo Comunitario de Ubuntu.

Concejos, Juntas y Delegaciones. Son grupos de trabajo más pequeños encargados de resolver conflictos cuando estos no puedan encontrar solución en los grupos de origen. También actúan para encaminar el trabajo en los diferentes espacios del proyecto así como la aprobación de nuevos miembros; entre otras tareas.

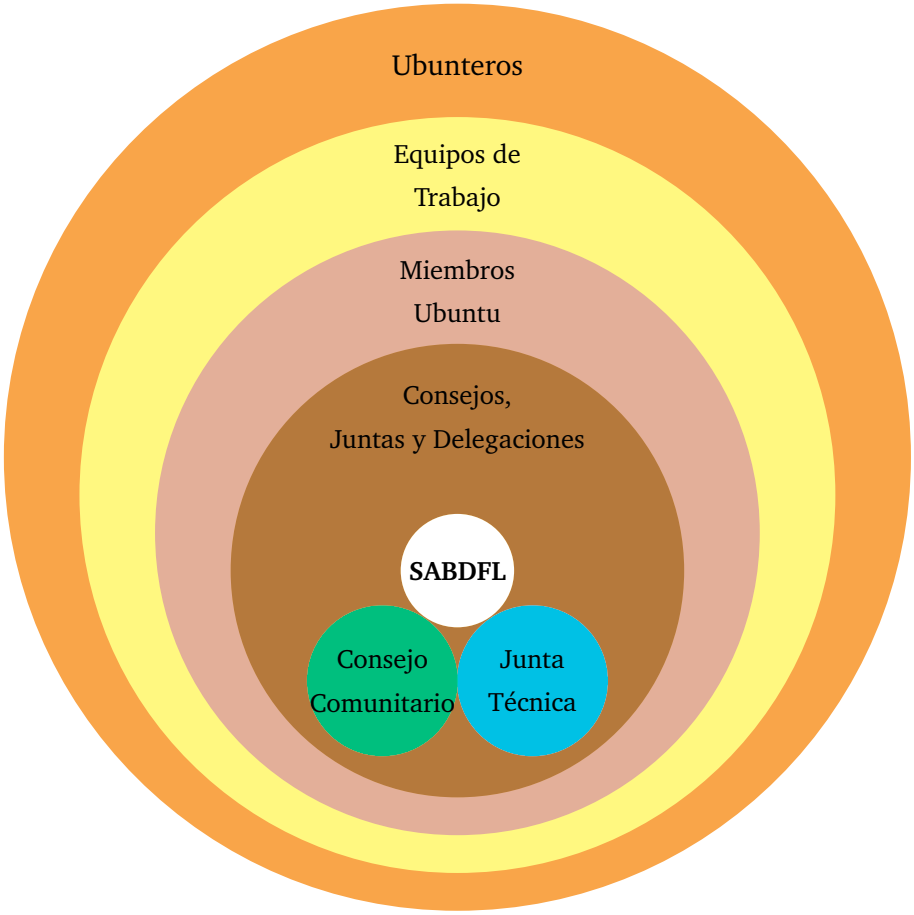
SABDFL. Así como Debian posee un Líder de Proyecto, Ubuntu presenta la variante al Dictador Benevolente de por Vida (*Benevolent Dictator for Life* – BDFL). El BDFL es el "título informal que se otorga a ciertos individuos de la comunidad de desarrolladores de software de código abierto que tienen la tarea de asignar las directrices generales y, en ciertas situaciones, las decisiones finales dentro del ámbito de un proyecto" (Wikipedia, 2017). En Ubuntu hay un Auto-Nombrado Dictador Benevolente de por Vida (Self-Appointed Benevolent Dictator for Life – SABDFL), como se refiere así mismo Shuttleworth. El SABDFL es quien guía el proyecto. Tiene la potestad de solicitar a las personas trabajar en tareas específicas. Finalmente, en caso de ser necesario y ya agotados todos los mecanismos previos, posee la última palabra en algún conflicto en el que no se haya logrado consensuar una solución.

El Consejo Comunitario es el órgano responsable por mantener el Código de Conducta y aprobar modificaciones al mismo. Está encargado de manejar los nombramientos y elecciones a las Juntas y Consejos del proyecto, así como también, crear o disolver equipos. Cuando los equipos no logran solucionar sus disputas internamente es el Consejo Comunitario la entidad encomendada en

buscar la resolución de estas. Los miembros del Consejo son nominados por el SABDFL y electos en votación general por los Miembros Ubuntu[22].

La Junta Técnica es el par del Consejo Comunitario a nivel técnico. Está encargado de tomar las decisiones con respecto a selección de paquetes, políticas de empaquetamiento, bibliotecas, dependencias de paquetes, metas para los próximos lanzamientos y demás asuntos técnicos. Los miembros de la Junta Técnica son nominados por el SABDFL y elegidos en votación por los Desarrolladores Ubuntu con permisos para subir paquetes a los repositorios oficiales de Ubuntu[23].

Figura 3.7: Integrantes de la estructura organizativa de Ubuntu



[22] https://wiki.ubuntu.com/CommunityCouncil
[23] https://wiki.ubuntu.com/TechnicalBoard

En el ámbito de desarrollo, en Ubuntu aparecen los siguiente 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

Para finales del año 2016 existían poco más de 1.100 colaboradores en el proyecto Ubuntu, distribuidos de la siguiente manera.

- 87 *Ubuntu Core Developers*[24]
- 159 *Ubuntu Contributing Developers*[25]
- 148 MOTU[26]
- 734 *Ubuntu Members*[27]

[24] https://launchpad.net/~ubuntu-core-dev [Consultado en noviembre de 2016]
[25] https://launchpad.net/~ubuntu-developer-members [Consultado en noviembre de 2016]
[26] https://launchpad.net/~motu [Consultado en noviembre de 2016]
[27] https://launchpad.net/~ubuntumembers [Consultado en noviembre de 2016]

Lanzamientos

La Tabla 3.5 muestra la fecha de lanzamiento de las diferentes versiones de la distribución Ubuntu.

Tabla 3.5: 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 [28]

Histórico de distribuciones

El registro de todas las versiones publicadas por Ubuntu se encuentran en la siguiente dirección: http://old-releases.ubuntu.com/releases/

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

Arquitecturas

Ubuntu está oficialmente soportada y portada para 7 arquitecturas, mostradas en la tabla 3.6.

Tabla 3.6: 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[29]

Soporte

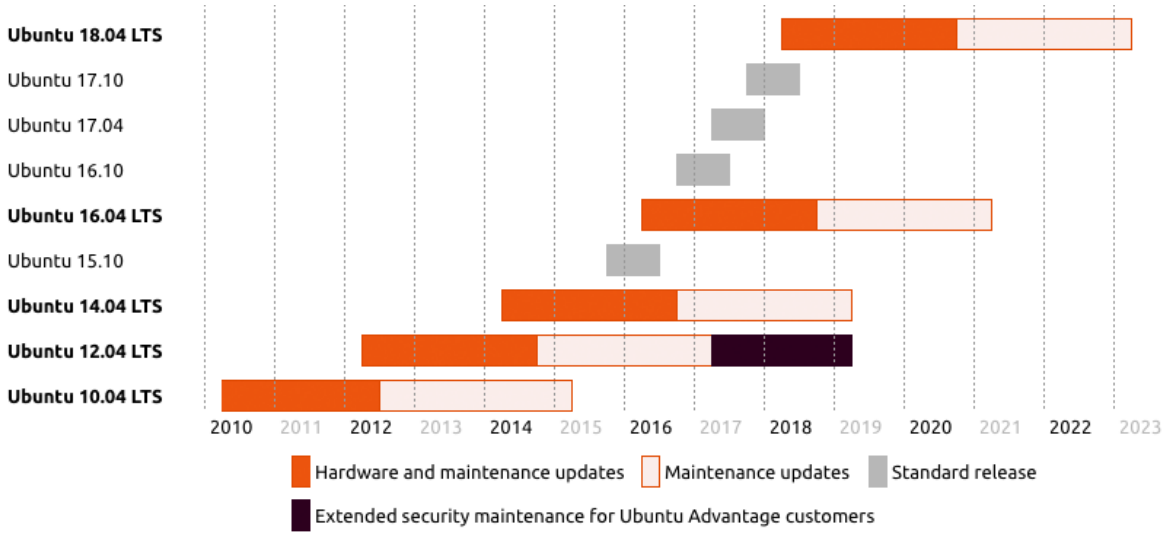
Ubuntu hace lanzamientos con diferentes tiempos de soporte. Cada seis meses se publica una nueva versión, tradicionalmente en los meses de abril y octubre, y las versiones con soporte extendido se publican cada dos años, generalmente es la versión que corresponde salir en el mes de abril. Ver Figura 3.8.

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

No–LTS: Soporte por nueve meses.

[29] https://help.ubuntu.com/community/SupportedArchitectures

Figura 3.8: Tiempo de soporte para las diferentes versiones de Ubuntu



Fuente: https://www.ubuntu.com/info/release-end-of-life/

Componentes

Ubuntu categoriza los paquetes incluidos en su sistema dentro de cuatro repositorios, tomando en cuenta si se ajustan a la categoría de libre o no, según la Filosofía de Software Libre de Ubuntu[30], 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.

Listas de discusión

En Ubuntu manejan una variedad de herramientas para la comunicación entre la comunidad, destacan las listas de correo y el IRC (*Internet Chat Relay*), pero también coexisten otros recursos como páginas de foro, wikis, planetas, entre otros.

[30] https://www.ubuntu.com/about/about-ubuntu/our-philosophy

Para la obtención del número de correos enviados a las listas de Ubuntu y Canaima, se utilizó una versión modificada de la técnica publicada en la entrada *Some statistics on linux-greek-users mailing list and forum.hellug.gr*[31] del blog *Into the Void*.

La técnica consiste en descargar el histórico de archivos de las listas de correos en formato HTML con los registros de cada mes, para luego extraer la información de fecha (mes–año) y cantidad de correos enviados utilizando herramientas propias del sistema operativo GNU/Linux para posteriormente almacenarlos en un archivo csv que es utilizado para la generación de los gráficos.

Lista de usuarios

La Figura 3.9 muestra dos grandes períodos de actividad, el primero entre 2004 y 2007; el segundo entre 2007 y 2011. A partir de 2011 se aprecia un decrecimiento en la actividad para finalmente mostrar una estabilidad desde mediados del año 2013. El mes con la mayor actividad mostrada se presenta durante el lanzamiento de la primera versión de Ubuntu, Warty Warthog 4.10, cuando se acerca a los 7.000 correos en octubre de 2004. En esta lista de soporte se distinguen incremento de actividad durante los meses de lanzamientos (abril y octubre) de cada año.

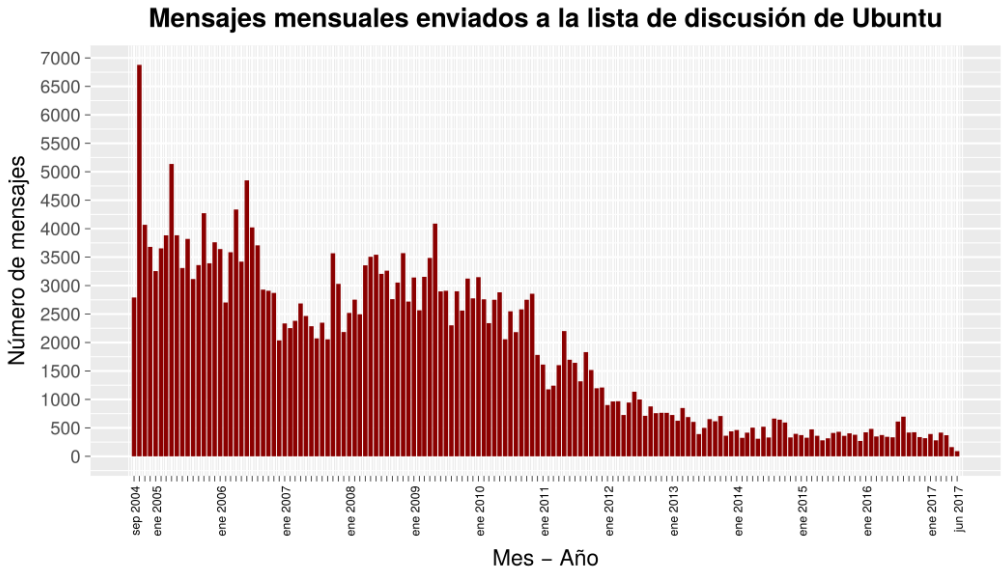
Lista de desarrolladores

La lista de correos de desarrolladores está creada para la discusión entre desarrolladores sobre sus proyectos, interrogantes sobre políticas y procedimientos relacionados con el desarrollo de Ubuntu. Es una lista pública, moderada para aquellos que no son desarrolladores Ubuntu.

El comportamiento de la lista de correos de desarrolladores varía con respecto al de soporte en cuanto que los períodos de incremento de actividad se presentan previo al mes de lanzamiento. Los escenarios con mayor actividad ocurren en los 3 primeros años (finales de 2004 a finales de 2006) con máximos rondando un par de veces los 1.450 mensajes en el 2005. Ver Figura 3.10.

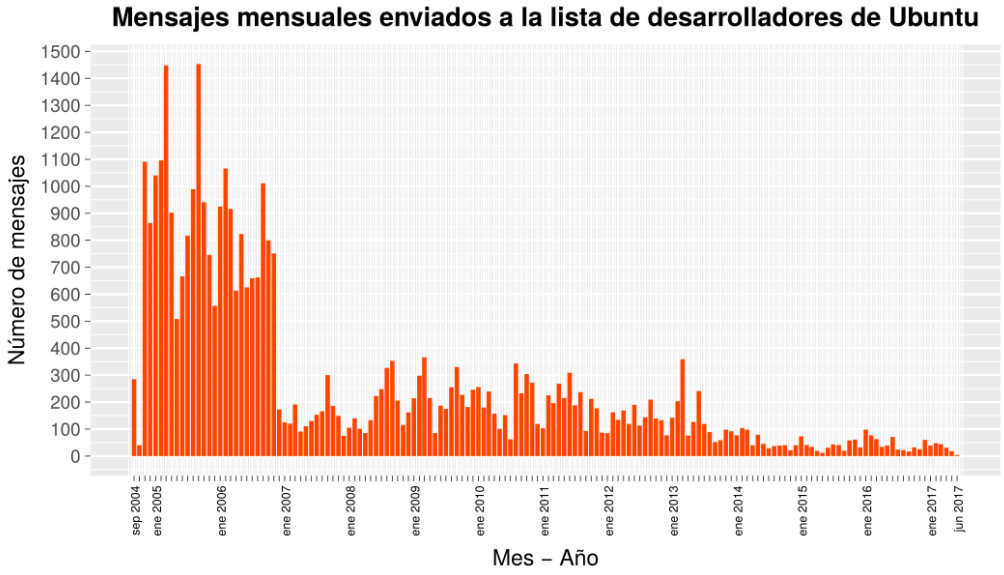
[31] https://www.void.gr/kargig/blog/2009/05/02/some-statistics-on-linux-greek-users-mailing-list-and-forumhelluggr/

Figura 3.9: Actividad de la lista de correos *Users* de Ubuntu



Datos: https://lists.ubuntu.com/mailman/listinfo/ubuntu-users/

Figura 3.10: Actividad de la lista de correos *Developers* de Ubuntu



Datos: https://lists.ubuntu.com/mailman/listinfo/ubuntu-devel/

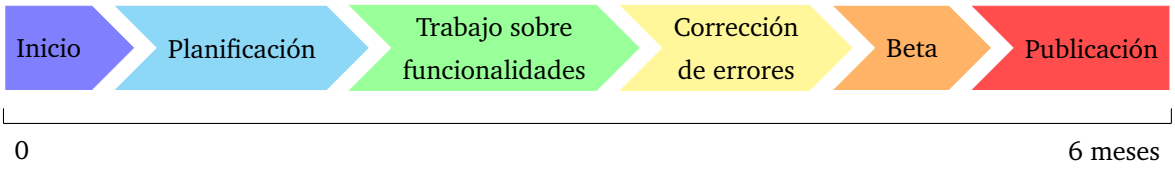
Ciclo de desarrollo

De acuerdo a (Helmke y Graner, 2012), uno de los convenios técnicos acordados en el equipo fundador de Ubuntu fue el de hacer lanzamientos a corto plazo y predecibles. Ubuntu ha adoptado así un ciclo de desarrollo con una duración de 6 meses. Igualmente, (Helmke y Graner, 2012) destacan la importancia de los lanzamientos predecibles; conserva un interés empresarial, para que de este modo las compañías puedan organizar sus planes de negocio en torno a los lanzamientos de nuevas versiones de Ubuntu.

El ciclo de desarrollo de Ubuntu está basado en el tiempo, en lanzamientos fijos, periódicos. Dentro del cronograma de 6 meses existe a su vez diferentes fechas claves o hitos, y fechas de congelamiento donde se evalúan ciertas funciones y características a medida que avanza el desarrollo, para conseguir una estabilización gradual de la distribución hasta su publicación definitiva.

Las diferentes fases del ciclo de desarrollo se dibujan en la Figura 3.11 y se explica a continuación. La fase de desarrollo de cada nueva versión inicia con la puesta a punto de la infraestructura y herramientas de desarrollo que permiten construir la distribución. En la fase de planificación se discuten y definen las características que poseerá la versión. El trabajo sobre las funcionalidades comienza con la importación de paquetes provenientes de los repositorios de Debian en sus versiones de desarrollo. Al mismo tiempo, se trabaja en los proyectos definidos durante la etapa de planificación. Luego entra la etapa de corrección de errores, la meta consiste en lograr un estado en el que no existan errores críticos y así culminar con el lanzamiento de la nueva versión.

Figura 3.11: Procesos de desarrollo de cada versión de Ubuntu



Como se mencionó previamente, durante el proceso de desarrollo existen algunas fechas claves, son las fechas de congelamiento y fechas de lanzamientos de versiones para pruebas.

Fechas de congelamiento

Congelamiento de definición de funcionalidades (*Feature Definition Freeze*): Fecha para la cual las funcionalidades y paquetes definidos en la planificación deben estar en el repositorio *main* y deben ser aprobadas por el equipo de lanzamiento.

Congelamiento de funcionalidades (*Feature Freeze*): Se dejan de añadir paquetes y funcionalidades para enfocarse ahora en la corrección de errores.

Congelamiento de importaciones desde Debian (*Debian Import Freeze*): Se dejan de importar paquetes y funcionalidades de los repositorios de Debian.

Congelamiento para Beta 1 (*Beta 1 Freeze*): Se congela la subida de paquetes y se hace una primera estabilización para pruebas.

Congelamiento de interfaz de usuario (*User Interface Freeze*): Estabilización de la interfaz de usuario.

Congelamiento de documentación (*Documentation String Freeze*): Se estabiliza la documentación para permitir el trabajo de los traductores.

Congelamiento para Beta Final (*Final Beta Freeze*): Segunda estabilización para pruebas.

Congelamiento del núcleo (*Kernel Freeze*): Congelamiento de actualizaciones del núcleo.

Congelamiento de traducciones fuera del paquete de traducciones (*Non Language Pack Translation Deadline*): Se paralizan los trabajos de traducción de los paquetes que se escapan de los paquete de traducciones (*Languages Pack*).

Congelamiento Final (*Final Freeze*): Etapa de congelamiento previa al lanzamiento de la versión final. No deberían existir errores de seguridad ni errores críticos. De haberlos, se resolverán en esta etapa.

Congelamiento del paquete de traducciones (*Language Pack Translation Deadline*): Finaliza las traducciones que serán incluidas en los paquetes de traducciones.

Hitos

Alpha 1: Primera versión de pruebas donde se maqueta la distribución. Aquí se prueban las funcionalidades y paquetes escogidos en la fase de planificación.

Alpha 2: Versión de pruebas para corregir errores encontrados durante la fase previa a la *Congelación de funcionalidades*.

Beta 1: Versión de pruebas que debería estar libre de errores mayores.

Beta Final: Versión de pruebas que debería estar libre de errores críticos.

Lanzamiento de la versión candidata (*Release Candidate*): Si todo ha salido perfecto, la versión candidata debería terminar siendo la versión final. Es la última versión de pruebas.

Lanzamiento de la versión final: Publicación definitiva de la nueva versión de Ubuntu.

Estimación de costos

La Tabla 3.7 expone un resumen con los datos obtenidos del análisis realizado sobre el componente *main* de Ubuntu 16.04. Se estudiaron 2.495 paquetes de software y se detectaron 150 lenguajes.

Tabla 3.7: SLOC agrupados por lenguaje para Ubuntu 16.04.

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**

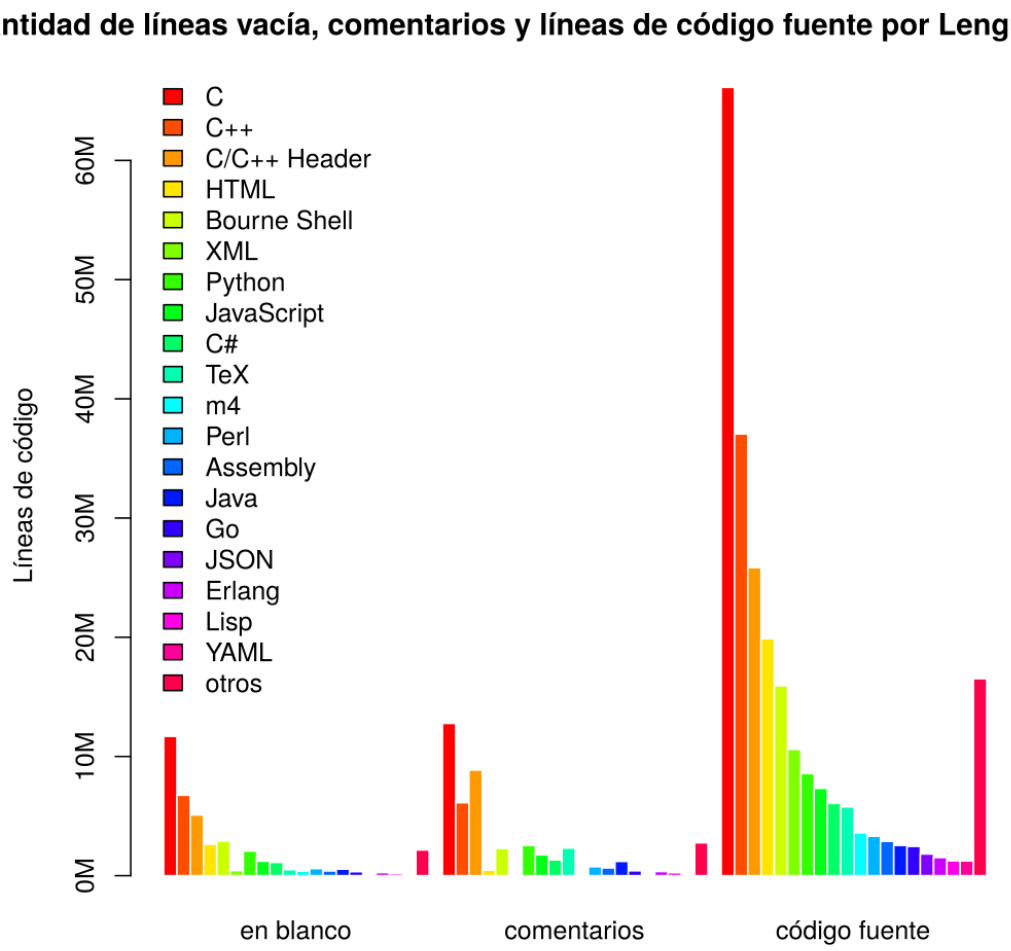
Fecha del archivo Source: 02/04/2017

Número de paquetes descargados: 2.495

Cantidad de lenguajes reconocidos: 150

Lo primero que se percibe es la diferencia en la cantidad de paquetes incluidos en el repositorio principal de Ubuntu, esta cantidad representan el 11,9% de los paquetes que se encuentran en el repositorio equivalente de Debian. Por otro lado, al igual que en Debian, el lenguaje C domina en cantidad de líneas en blanco, líneas con comentarios y líneas de código fuente, ver Figura 3.12.

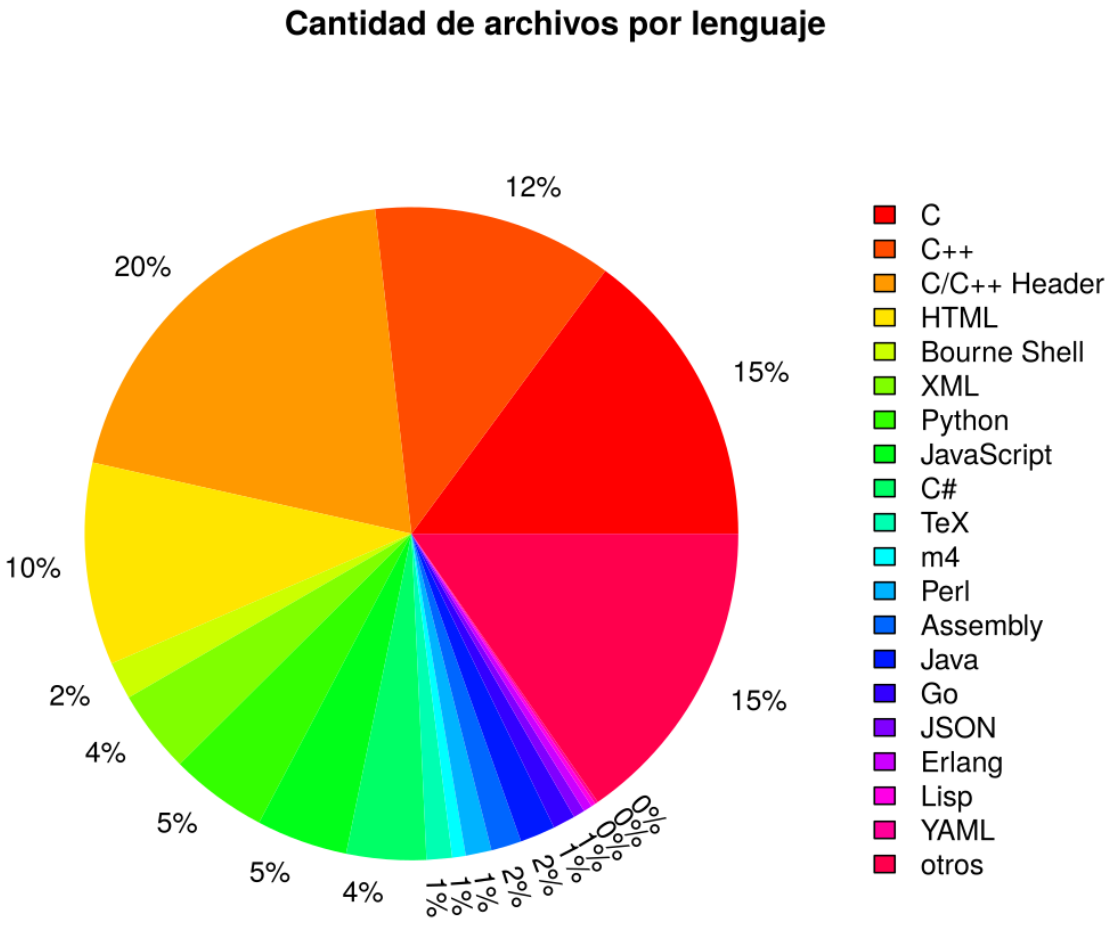
Figura 3.12: Cantidad de líneas vacía, comentarios y líneas de código fuente en los lenguajes del componente *main* de Ubuntu 16.04.



En el renglón de cantidad de archivos totales, también como en Debian son los archivos de cabecera para los lenguajes C y C++ quienes prevalecen como los más numerosos, 208.053 archivos que representan el 20% de totalidad del repositorio.

La Figura 3.13 muestra el resto de porcentajes de la cantidad de archivos por cada lenguaje detectado.

Figura 3.13: Porcentaje de archivos por lenguaje dentro del componente *main* de Ubuntu 16.04.



COCOMO

Para el componente *main* de Ubuntu, el total de código fuente de todos los paquetes analizados en la versión 16.04 supera los 240 millones de líneas, esto significa el 26,7% de líneas de código fuente que en el caso de Debian, es decir, poco más de una cuarta parte del total de líneas de código que el componente *main* de Debian 8.0. La Tabla 3.8 presenta los valores obtenidos aplicando COCOMO Básico.

Tabla 3.8: Estimaciones de esfuerzo y costos para el componente *main* de Ubuntu 16.04 aplicando COCOMO Básico.

Líneas de código fuente:	240.297.751
Esfuerzo estimado de desarrollo (persona–mes):	1.071.515,44
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[32].

Por tanto, para enero de 2017, el desarrollo de Ubuntu costaría aproximadamente US$ 161.783.066,47, teniendo la referencia del salario calculado a US$ 73.837 y una cantidad de 2.191 personas trabajando en el proyecto para completarlo en 40 años aproximadamente.

Canaima GNU/Linux

Definición

La página oficial de la distribución describe a Canaima como “un proyecto socio-tecnológico-productivo abierto, construido de forma colaborativa por un conjunto de actores de la vida nacional que incorporan elementos de orden tecnológico, comunitario y estratégico, desarrollando herramientas y modelos productivos basados en Tecnologías de Información (TI) Libres” (Centro Nacional de Tecnologías de Información, 2017).

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

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) (Decreto Nº 3.390, 2004). En el Artículo 7 de la mencionada Ley 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" (Decreto Nº 3.390, 2004).

El músculo financiero y laboral proviene del Estado Venezolano, con apoyo de la comunidad, destacando en tareas de difusión, planificación y coparticipación en el manejo de algunas plataformas.

Gobernanza

Política

Las normas que definen la distribución están supeditadas a las leyes venezolanas. La primera acontece, como se mencionó antes, con el Decreto 3.390. Luego se empiezan a dibujar los lineamientos en varios documentos que servirían como base para lo que luego se convertiría el Proyecto Canaima.

En el Plan Nacional de Ciencia, Tecnología e Innovación 2005–2030 (PNCTI) se logra percibir objetivos que serán pilares de la política del proyecto Canaima. Por ejemplo, una de las *Metas Estratégicas* dentro del *Marco Político Estratégico* del PNCTI dice, "Migración de los sistemas de la administración pública nacional a los sistemas de software libre hasta alcanzar absoluta sustitución dentro de las plataformas tecnológicas del Estado, en un plazo no mayor de cinco años" (Ministerio de Ciencia y Tecnología, 2005, p.91). Esta meta, aún por alcanzar, se vio respaldada legalmente a través de la Resolución Nº 025 que decreta el uso de Canaima en las estaciones de trabajo de la APN.

El Proyecto Nacional Simón Bolívar en su Primer Plan Socialista (PPS) del Desarrollo Económico y Social de la Nación para el período 2007–2013, en uno de sus objetivos habla de "Fomentar la ciencia y la tecnología al servicio del desarrollo nacional y reducir diferencias en el acceso al conocimiento" (Presidencia de la

República Bolivariana de Venezuela, 2007). En el Plan Nacional que lo sucede, (Ley del Plan de la Patria, 2013), Segundo Plan Socialista de Desarrollo Económico y Social de la Nación 2013–2019, se divisa aún mas el apoyo a las tecnologías libres y estándares abiertos dedicándole varios *Objetivos Estratégicos y Generales* enmarcados en el *Objetivo Nacional* que busca desarrollar las capacidades científico-tecnológicas vinculadas a las necesidades del pueblo, dentro del *Gran Objetivo Histórico Nº 1*. Asimismo, también se ve reflejada la tecnología en la sección dedicada a la *Política y Programas del Sector Ciencia y Tecnología (telecomunicaciones)* del referido escrito.

Y por supuesto, están las leyes de Infogobierno, instrumento que al decretarse deroga el decreto 3.390; y la Ley Orgánica de Ciencia, Tecnología e Innovación (LOCTI) que avalan la misión y objetivos del proyecto Canaima.

Documentos conceptuales y organizativos sobre el proyecto se han venido publicando a través de una *Enciclopedia Colaborativa de la Comunidad Canaima*[33]. Esta wiki congrega una base de conocimientos construidos en comunidad que incluye guías técnicas y organizacionales del proyecto.

De los varios documentos se extraen los siguientes puntos que ayudan a perfilar el camino que debe seguir la distribución.

- Canaima es el sistema operativo nacional.
- Está basado en tecnologías libres.
- Está orientado a suplir las necesidades de la APN.
- Su desarrollo se basa en la versión estable de Debian[34].
- Canaima es un proyecto colaborativo y comunitario. En consecuencia, busca la cooperación y coparticipación de la Comunidad Canaima.

La Comunidad Canaima es definida como un “colectivo constituido por miembros de la vida nacional e internacional, asidos al manifiesto del Conocimiento Libre, trabajando en favor de la soberanía e independencia tecnológica como fin común”

[33] http://wiki.canaima.softwarelibre.gob.ve/

[34] A partir de la versión 5.0 de Canaima se añaden componentes de Linux Mint Debian Edition (LMDE).

(Solano et al., 2011). Esto es: comunidad de usuarios, instituciones públicas, empresas privadas, cooperativas, activistas, etcétera.

En la 6ª Cayapa[35] realizada en la ciudad de Barinas en el año 2012 se trabajó en una propuesta para redactar el Contrato Social de Canaima[36] tomando el Contrato Social de Debian y adaptándolo a Canaima.

Los lineamientos para el manejo colaborativo de la plataforma tecnológica de Canaima se encuentra esbozado en el wiki que alberga la *Propuesta de política sobre gestión colaborativa de la plataforma tecnológica de Canaima*[37].

Entre el 12 y 14 de noviembre de 2014, en la población de Bailadores, Estado Mérida, se llevó a cabo la 8ª Cayapa Canaima. Esta cayapa cobra especial interés por dos razones principales, la primera porque ha sido la única cayapa hasta el momento autogestionada y organizada exclusivamente por la comunidad de usuarios; y la segunda, porque fue una cayapa reflexiva, filosófica, de revisión, conceptualización y fundamentación del proyecto desde el colectivo.

De esta actividad, sobre la toma de decisiones en lo relativo al proyecto y sus justificaciones, se convinieron las siguientes premisas que recoge Colina (2014b) en correo a la lista de discusión de Canaima,

- Cualquier propuesta no debe eliminar funcionalidades existentes.
- Cualquier decisión debe incluir la visión del usuario.
- Cualquier propuesta generada debe estar lo suficientemente documentada y sustentada siguiendo la metodología del proyecto.
- Cualquier propuesta que sea rechazada por el equipo de trabajo, debe estar lo suficientemente justificada.
- Cualquier decisión tomada con respecto a propuestas aceptadas debe ser documentada.

[35] Las Cayapas Canaima son espacios de encuentro que persiguen el intercambio de saberes socio-tecnológicos para la construcción del proyecto Canaima.

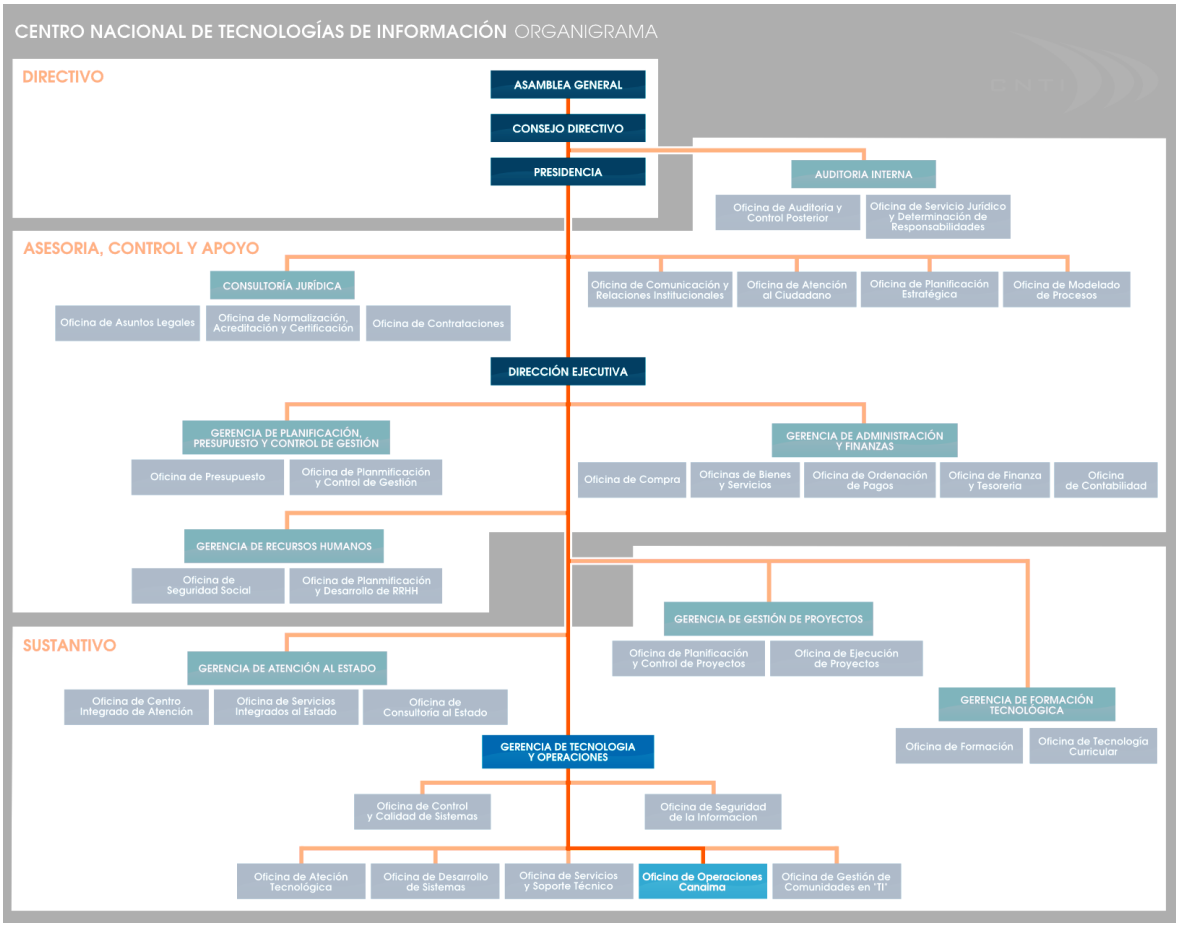
[36] http://wiki.canaima.softwarelibre.gob.ve/index.php/Contrato_Social

[37] http://wiki.canaima.softwarelibre.gob.ve/index.php/Papel_de_Trabajo_para_la_creaci%C3%B3n_de_las_Pol%C3%ADticas_de_Participaci%C3%B3n_en_la_Plataforma_Colaborativa_Canaima

Organización

Primeramente entiéndase que Canaima es un proyecto concebido desde el Estado que busca atender las necesidades de la Administración Pública Nacional. Hoy en día el ente encargado del proyecto Canaima es el Centro Nacional de Tecnologías de Información (CNTI), desde la *Oficina de Operaciones Canaima*, dependencia de la *Gerencia de Tecnología y Operaciones*, que depende a su vez de la *Dirección Ejecutiva* y esta, de los cuerpos directivos del ente, ver Figura 3.14.

Figura 3.14: Organigrama CNTI



Fuente: CNTI

Canaima además se entiende como proyecto de construcción comunitaria, con aportes constantes, debates constantes; en torno al trabajo que implica construir una distribución con participación de la comunidad de usuarios pero sujeto a los

compromisos organizacionales que dictan los objetivos y metas planteadas en un Plan Operativo Anual (POA) de una institución gubernamental.

Por todo esto, tomando en cuenta las características propias de las estructuras gubernamentales de Venezuela, la disposición organizativa de Canaima pinta un modelo jerárquico en el que participan dos actores principales: primero el Gobierno y luego la Comunidad.

El Ministerio de Ciencia y Tecnología o su equivalente, es el representante por parte del Estado de evaluar y aprobar el POA así como el presupuesto solicitado por el CNTI. El Estado es el ente financiador del proyecto y el que aprueba las políticas que direccionan el proyecto.

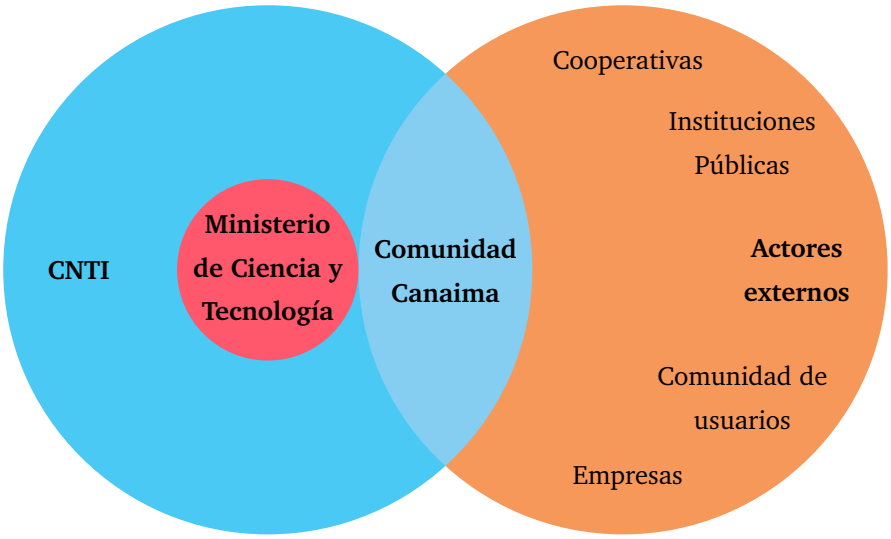
CNTI es el organismo designado por el Ministerio de Ciencia y Tecnología para diseñar la planificación anual del proyecto y llevar a cabo su ejecución. El CNTI es la entidad que tiene entre sus funciones administrar, dirigir, planificar, ejecutar, decidir, desarrollar, mantener, difundir y promover el proyecto Canaima por medio de la Oficina de Operaciones Canaima.

Actores externos son todos aquellos cuerpos participantes del proyecto que no forman parte del CNTI.

La Comunidad Canaima "se conforma por muchos actores y actrices quienes asumen diversos roles, entre los cuales pueden existir usuarios finales, tecnólogos, socializadores, mentores, aprendices, el rol de apoyo estratégico y el institucional" (Solano et al., 2011). La Comunidad Canaima es la unión del Gobierno, representado por el CNTI y los actores externos. Se desenvuelve como colaborador en tareas de desarrollo, soporte, difusión entre otras actividades, así como también labores de administración de ciertas plataformas con el acompañamiento de personal del CNTI.

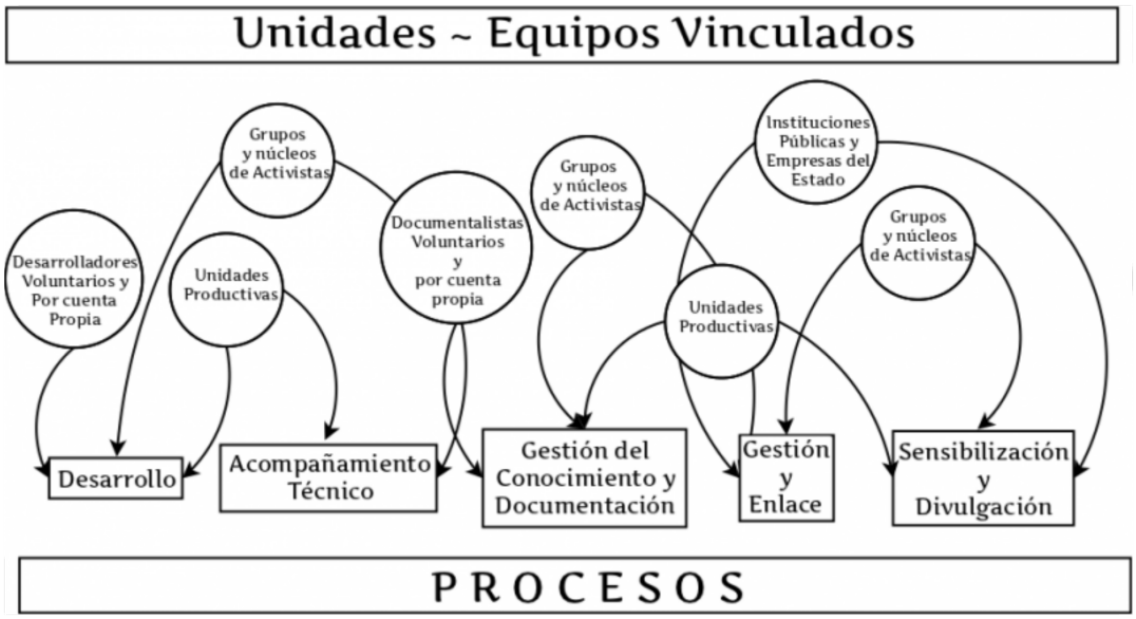
Este ecosistema se bosqueja en la Figura 3.15

Figura 3.15: Integrantes de la estructura organizativa de Canaima



En añadidura, Petrizzo y Muñoz (2012) proponen un modelo matricial donde identifican y clasifican actores y procesos dentro del Proyecto Canaima y las relaciones entre sí. El modelo se presenta en la Figura 3.16.

Figura 3.16: Modelo matricial para el Proyecto Canaima GNU/Linux



Fuente: (Petrizzo y Muñoz, 2012)

Colaboradores

Para septiembre de 2016 el Equipo Canaima del CNTI[38] está conformado por:

- **1 Jefe de Oficina Canaima GNU/Linux.**
    - Coordinar el equipo de trabajo del Proyecto Canaima.
    - Promover el cambio del modelo productivo del proyecto.
    - Producción de software centrado en el usuario final.
    - Diseño y construcción de una fábrica de ensamblaje de software en Venezuela.
- **2 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.
    - Levantamiento y análisis de requerimientos técnicos.
    - Formación tecnológica.
    - Sistematización de experiencias, planificación y gestión.
- **4 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.

[38] Fuente: http://canaima.softwarelibre.gob.ve/canaima/equipo-de-trabajo

- **2 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.
- **1 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.
- **1 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.
- **1 Gestión Administrativa.**
  - Procesos administrativos del proyecto Canaima GNU/Linux.

En cuanto a los roles identificados a nivel de desarrollo, en la 5ª Cayapa celebrada en Cumaná en 2011 se reconocieron 7 niveles de participación. De una nota del Jefe de Prensa (2011) del Ministerio del Poder Popular para Educación Universitaria Ciencia y Tecnología (MPPEUCT) se desglosan los siguientes perfiles,

Miembro es el participante de la comunidad más novel, el aprendiz.

Canaimero es un articulador, defensor y activista del proyecto Canaima GNU/Linux.

Desarrollador es aquel miembro que contribuye con elementos de programación y desarrollo.

Mantenedor apoya en la tarea de sustentabilidad del sistema y empaquetamiento de software.

Documentador es la persona encargada de llevar la documentación.

Probador es el responsable de realizar las pruebas a la plataforma tecnológica con la presentación de un reporte de resultados.

Chamán es un mentor que conoce las dimensiones tecnológicas, sociales, ideológicas y legales del proyecto.

Cayapas

A nivel comunitario las cayapas son quizás el elemento más extraordinario que se da dentro del proyecto. Las cayapas son para Canaima lo que para Ubuntu es un *Ubuntu Local Jam*[39] y para Debian una *Bug Squashing Party*[40].

Las cayapas son organizadas por la propia comunidad y existe toda una metodología para su estructuración. Con cronogramas para definir sedes, elecciones para determinar la sede ganadora vía canal IRC del proyecto, cronogramas de planificación y organización, y la generación de la respectiva documentación.

En la 8ª Cayapa se acuerdan las denominaciones para la organización de las mismas[41]. De este modo se definen:

Cayapa Maestra: Aquellas en las cuales se estabilizan los elementos claves del Proyecto en lo tecnológico, metodológico y organizativo dando un nuevo perfil estratégico a éste.

Estas cayapas son organizadas por la totalidad de los actores y sus agendas de trabajo son concordadas de forma pública persiguiendo objetivos estratégicos-prospectivos del proyecto y no puntuales, es decir el Plan Maestro del Proyecto.

Minicayapas: Son aquellos evento de índole técnico (*hacklabs*, corrección de errores, divulgación y formación).

[39] Eventos donde las personas se reúnen para hacer algo en torno a Ubuntu. https://wiki.ubuntu.com/Jams/ES.

[40] Eventos donde desarrolladores y entusiastas de Debian se reúnen, de manera virtual o en persona, para intentar solucionar la mayor cantidad de errores de código posibles. https://wiki.debian.org/BSP.

[41] http://listas.canaima.softwarelibre.gob.ve/pipermail/discusion/2014-November/011878.html

Lanzamientos

La Tabla 3.9 muestra la fecha de lanzamiento de las diferentes versiones de la distribución Canaima GNU/Linux.

Tabla 3.9: 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 VC1	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	Meru[42]	17/10/2009
2.1 VC1	Aponwao	21/05/2010
2.1 VC4	Aponwao	02/07/2010
3.0 VC1	Roraima	10/02/2011
3.0 VC2	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[43], Wikipedia [44] y Listas de correo[45] [46]

[42]Bautizada así post lanzamiento para adecuar la versión a una nueva estructura de repositorios.
[43]http://canaima.softwarelibre.gob.ve
[44]http://es.wikipedia.org/wiki/Canaima_(distribuci%C3%B3n_Linux)
[45]http://listas.canaima.softwarelibre.gob.ve/pipermail/discusion/
[46]http://listas.canaima.softwarelibre.gob.ve/pipermail/desarrollo/

Histórico de distribuciones

La página que recoge todas las versionas de Canaima publicadas durante el transcurso de su historia se encuentra en: http://canaima.softwarelibre.gob.ve/descargas/canaima-gnu-linux/repositorio-de-distribuciones

Arquitecturas

Canaima construye su distro bajo 3 arquitecturas, mostradas en la tabla 3.10. Para cada una de las arquitecturas existe además una versión con soporte UEFI.

Tabla 3.10: Arquitecturas soportadas oficialmente por Canaima GNU/Linux

Adaptación	Arquitectura
i586	PC Pentium de 32 bits (x86)
i686	PC Pentium Pro de 32 bits (x86)
amd64	PC de 64 bits (amd64,x86_64)

Fuentes: Repositorio Canaima[47], DistroWatch[48]

Soporte

Canaima, está basado en Debian y su ciclo de desarrollo también está fuertemente ligado con el ciclo de desarrollo y de soporte de Debian, con la salvedad que los trabajos de desarrollo en Canaima se efectúan con las versiones estables de Debian. Esta característica hace que los soportes de seguridad en la distribución nacional abarquen tanto como los cubra el ciclo de soporte de la distribución Debian. Así por ejemplo, el soporte de seguridad para la versión 4.1 de Canaima (Kukenán) estará disponible mientras exista soporte de seguridad para la versión 7 de Debian (Wheezy).

[47] http://repositorio.canaima.softwarelibre.gob.ve/
[48] https://distrowatch.com/table.php?distribution=canaima

En Canaima la modalidad de soporte a largo plazo (LTS) lleva el nombre “soporte largo” (“SL”), y está reservado a las versiones mayores de Canaima, es decir, a las versiones primeras de su serie (1.0, 2.0, 3.0,. . .).

En la Tabla 3.11 se precisa la relación de soporte entre las versiones de Canaima y de Debian.

Tabla 3.11: Relación de versiones Canaima – Debian

Versión	Nombre Clave	Basado en
1.0	Canaima	Debian Etch (4.0)
2.0.X	Canaima	Debian Lenny (5.0)
2.0.4	Meru	Debian Lenny (5.0)
2.1	Aponwao	Debian Lenny (5.0)
3.0	Roraima	Debian Squeeze (6.0)
3.1	Auyantepui	Debian Squeeze (6.0)
4.0	Kerepakupai	Debian Wheezy (7.0)
4.1	Kukenán	Debian Wheezy (7.0)
5.X	Chimantá	Debian Jessie (8.0) y LMDE 2 (Betsy)

Fuentes: Descargas Canaima[49], Hunting Bears[50], Alba Ciudad[51] y Listas de correo[52] [53]

Componentes

Canaima, siendo una distribución derivada de Debian, comparte una estructura similar a esta. Así se han creado 3 componentes que se explican a continuación:

usuarios: Paquetes creados o adaptados para el proyecto Canaima, también denominados paquetes nativos.

[49] http://canaima.softwarelibre.gob.ve/descargas/canaima-gnu-linux/repositorio-de-distribuciones

[50] http://huntingbears.com.ve

[51] http://albaciudad.org/2016/12/canaima-gnu-linux-5/

[52] http://listas.canaima.softwarelibre.gob.ve/pipermail/discusion/

[53] http://listas.canaima.softwarelibre.gob.ve/pipermail/desarrollo/

aportes: Paquetes compatibles con las DFSG pero que tienen dependencias fuera de *usuarios*, incluida la posibilidad de tener dependencias en el componente *no-libres*.

no-libres: Paquetes no compatibles con las DFSG.

Al día de hoy sólo el componente *usuarios* está siendo usado para alojar paquetes. Los componentes *aportes* y *no-libres* fueron creados y pensados para albergar el software desarrollado por la comunidad, pero aún se mantienen vacíos, no porque no haya software hecho por terceros sino que se sigue utilizando el componente *usuarios* para disponer de todos los paquetes de software que no son importados de Debian.

Listas de discusión

Entre los temas que se abordaron en la 8ª Cayapa, las listas de discusión también fueron tema. Las normas de comportamiento en las listas de Canaima fueron discutidas por la comunidad asistente. De un hilo iniciado por Colina (2014a) quien se encargó de compilar la discusión de la cayapa, se rescata lo siguiente:

Normas de comportamiento para la lista Discusión

Las siguientes son una serie de reglas reconocidas en internet para el buen uso de las listas de discusión, trate de recordarlas cuando escriba en esta lista.

- Nunca olvide que la persona que lee el mensaje es en efecto humana con sentimientos que pueden ser lastimados: a veces es fácil olvidar que hay personas al otro lado de la red. Si por cualquier motivo estamos alterados lo más conveniente será esperar un poco antes de enviar mensajes. También es recomendable leer el texto antes de enviarlo.

- Trate de ser breve: Las frases breves tendrán más impacto que los largos párrafos, los cuales, probablemente serán menos leídos.

- Adhiérase a los estándares de comportamiento cónsonos con la buena educación, el respeto, la moral y buenas costumbres.

- Escribir todo en mayúsculas se considera como gritar y además, dificulta la lectura.
- Respete el tiempo y ancho de banda de las otras personas.
- Muestre el lado bueno de su persona mientras se mantenga en línea.
- Comparta su conocimiento con la comunidad.
- Ayude a mantener los debates en un ambiente sano y educativo.
- No abuse de su poder.
- Perdone los errores ajenos.

En esta lista no está permitido:

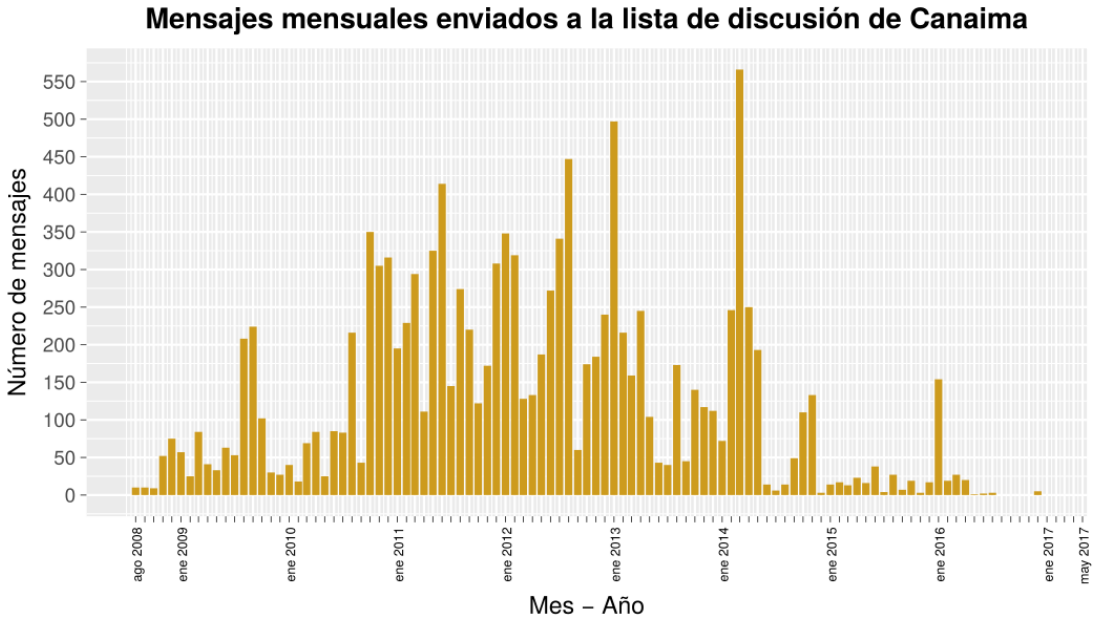
- Publicaciones que irrespeten, violenten, vulneren o discriminen a cualquier persona en función de su género, orientación política, religiosa o por su color de piel.
- Publicaciones que sean ataques directos o que maltraten moralmente a otros colaboradores del proyecto.
- Pornografía.
- Incitación a la guerra, magnicidio u otro similar.
- Escritura de instrucciones que podrían dañar equipos ajenos.

Lista de discusión

La coyuntura política también se hace presente en la comunidad tecnológica venezolana. Esto se hace manifiesto en enero de 2013 (497 correos) y en marzo de 2014 cuando se alcanza el techo más alto de mensajes enviados a la lista (566 correos) como consecuencia de una serie de disentimientos sobre posturas relacionadas con el clima político que atraviesa el país.

En general, la lista de discusión estuvo bastante activa entre mediados de 2010 y mediados de 2014. A partir de 2016 empieza la merma con periodos completos de inactividad, ya sea por nulo envíos de mensajes o debido a fallas presentadas en la plataforma. Ver Figura 3.17.

Figura 3.17: Actividad de la lista de correos *Discusión* de Canaima

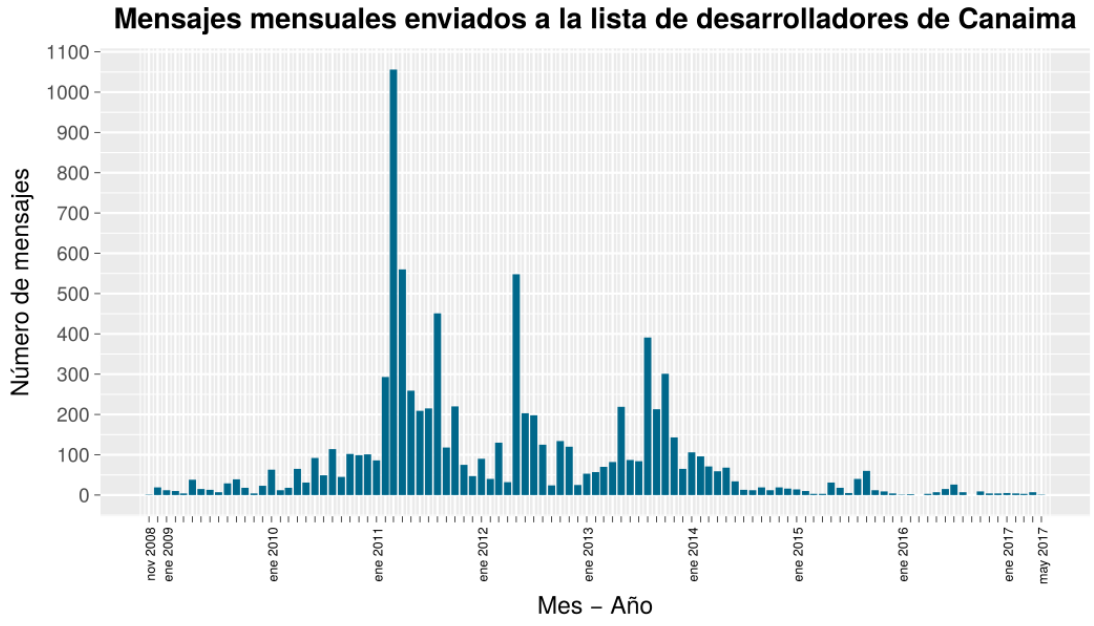


Datos: http://listas.canaima.softwarelibre.gob.ve/pipermail/discusion/

Lista de desarrolladores

En cuanto a la actividad en la lista de desarrollo, se distingue una conducta diferente en la frecuencia de envíos comparada con la lista de *Discusión*. En la Figura 3.18 se advierten 3 momentos de intensa actividad, 2011, 2012 y 2013. En marzo de 2011 se alcanza la marca histórica con 1.056 mensajes. Destacan los repuntes de actividad en meses donde se anuncian lanzamientos de versiones de Canaima y aquellos cuando coinciden con alguna Cayapa Canaima.

Figura 3.18: Actividad de la lista de correos *Desarrolladores* de Canaima



Datos: http://listas.canaima.softwarelibre.gob.ve/pipermail/desarrolladores/

Cabe mencionar que el sistema de gestión y seguimiento de proyectos que se usó en un principio (Trac[54]) estaba asociado con la lista de desarrollo. De este modo, cada vez que se reportaba una incidencia en el sistema, esta era publicada en la lista de correos para que los desarrolladores pudieran enterarse de las incidencias a través de este medio. El último correo enviado a la lista de *Desarrolladores* desde la cuenta *Sistema de Reporte de Tickets de Canaima* data de mayo de 2017.

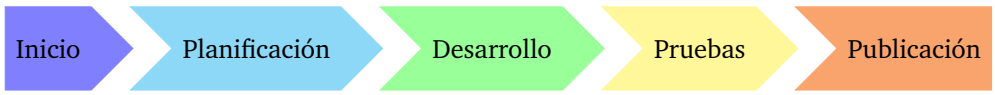
A finales de 2015 finaliza el proceso de preparación de una nueva plataforma de gestión, se adopta Gitlab[55], y en enero de 2016 se empieza a hacer pública la invitación[56] a los desarrolladores a participar en ese espacio.

[54] http://trac.edgewall.org/
[55] http://gitlab.canaima.softwarelibre.gob.ve
[56] http://listas.canaima.softwarelibre.gob.ve/pipermail/discusion/2016-January/012244.html

Ciclo de Desarrollo

Se pueden reconocer 5 situaciones clave durante el ciclo de desarrollo de Canaima, representadas en la Figura 3.19. La fase de inicio del ciclo, la de planificación de tareas y objetivos, la fase de desarrollo de la distribución, la fase de pruebas y finalmente la fase de publicación luego de asegurar una versión estable libre de errores fatales.

Figura 3.19: Procesos de desarrollo de cada versión de Canaima

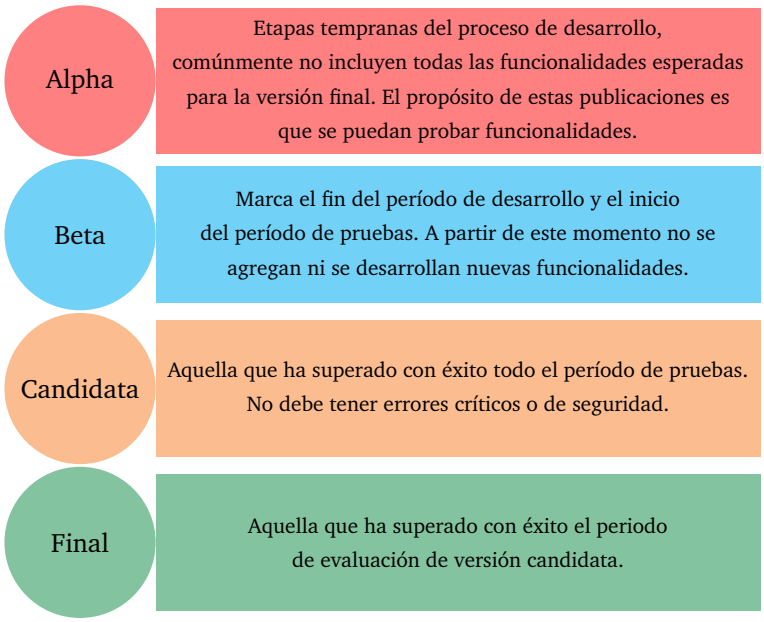


Al iniciar el ciclo de desarrollo se identifican las necesidades a través de distintos métodos, tales como discusiones y acuerdos efectuados en las cayapas, incidencias reportadas a través de la plataforma de gestión, listas de correos, sistemas de servicios de redes sociales, y en algún tiempo pasado, discusiones en el canal IRC. En esta fase la participación de la Comunidad Canaima cobra gran importancia porque dibuja la ruta que debería seguir la nueva versión.

Sobre la base de estos insumos, durante la fase de planificación, se definen las metas que se deben abordar para la entrega del producto y se conforman los grupos de trabajo para dar paso a la fase de desarrollo.

En la fase de desarrollo se adecua el trabajo publicando versiones alpha para comprobar el avance del proceso de definición de la distribución antes de continuar a la fase de pruebas. Es aquí donde se terminan de corregir los errores hasta asegurar una distribución libre de errores críticos a través de la prueba de versiones beta y versiones candidatas justo antes de la publicación de la versión definitiva.

Figura 3.20: Hitos



Fuente: http://wiki.canaima.softwarelibre.gob.ve/index.php/Protocolo_para_el_lanzamiento_de_versiones_de_Canaima_Popular

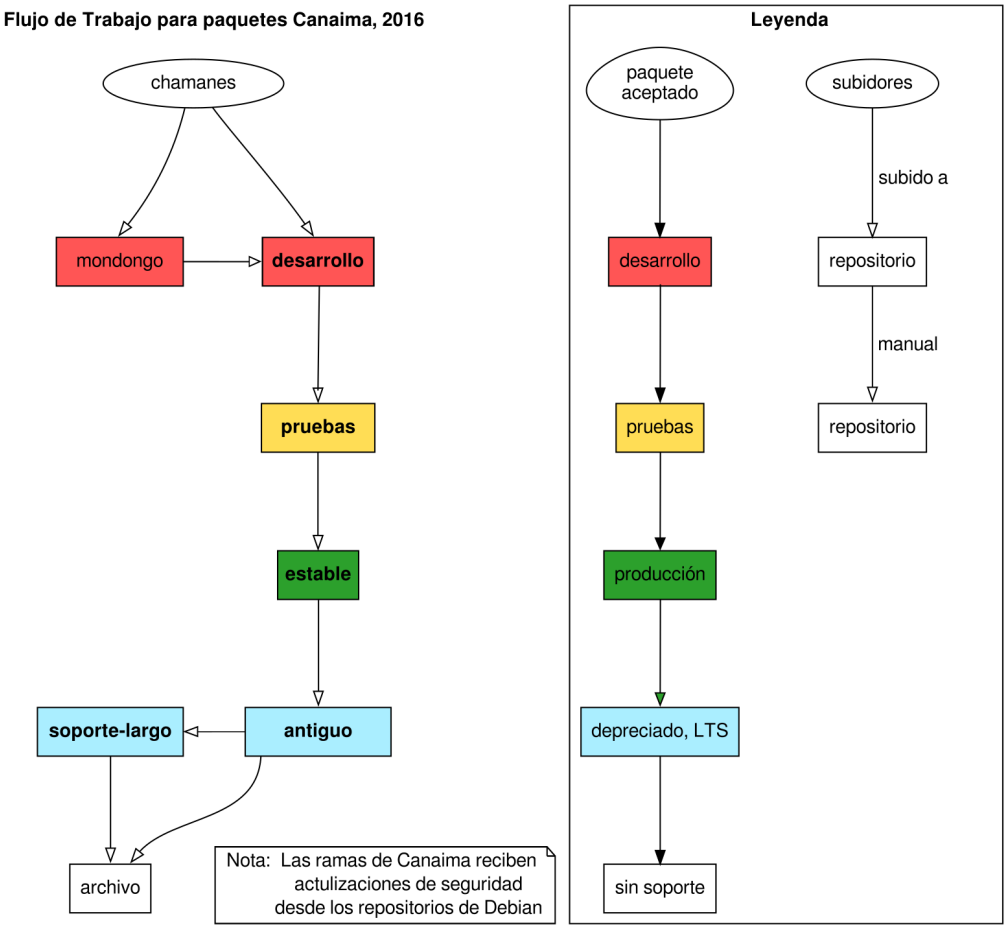
El procedimiento para que un paquete sea asimilado en el sistema Canaima se representa en la Figura 3.21. Los repositorios en negrita representan aquellos que contienen el sistema base, es decir, ese repositorio contiene los paquetes requeridos para poder tener un sistema operativo funcional, mientras que el resto de repositorios solamente contienen algunos paquetes y no podrían funcionar por sí solos.

Anteriormente el procedimiento implicaba, como *mantenedor*, solicitar a un *chamán*, vía correo electrónico a la lista de desarrollo, la subida de su paquete al repositorio *Mondongo* o *Desarrollo*. Se recuerda que *Mondongo* es el homólogo al repositorio *Expermiental* de Debian, y *Desarrollo* el homólogo de *Inestable*. Todos los traslados de un repositorio a otro se hacen de forma manual por algún *chamán*.

Martínez (2011a) explica el procedimiento en correo enviado a la lista de desarrollo, “se colocarán por primera vez todos los paquetes que deseen entrar al ciclo de desarrollo de Canaima (desarrollo > pruebas > estable). Los paquetes pasarán de mondongo a desarrollo a través de un “chamán” [...] que los probará hasta que estén listos para pasar.”

El empaquetado debe cumplir los requisitos mínimos de calidad guiándose por los mecanismos de desarrollo de Debian. De igual forma se deben disponer de las fuentes para poder llevar a cabo el proceso.

Figura 3.21: Flujo de trabajo de paquetes Canaima para 2016



Al día de hoy el procedimiento se realiza por medio de la plataforma de desarrollo colaborativo GitLab del proyecto[57]. El desarrollador debe registrarse en el sistema y subir allí sus código debianizado[58], informando posteriormente a los chamanes quienes se encargarán de enlazar el proyecto con el *buildbot* configurado para

[57] http://gitlab.canaima.softwarelibre.gob.ve/
[58] Con formato de empaquetamiento compatible con el sistema de paquetes Debian.

aplicar pruebas de calidad y empaquetar luego el código para su integración en los repositorios de Canaima.

Una vez cumplidos los hitos previos al lanzamiento de una nueva versión, se preparan los repositorios para establecer los cambios de nombre. Así, la versión que se encontraba en *soporte-largo* se convierte en una versión sin soporte y es almacenada en los *archivos* del proyecto, esto sucede sólo en caso de que en el repositorio *antiguo* se encontrase una versión mayor, en el caso de que en el repositorio *antiguo* existiese una versión menor, esta pasaría directamente al *archivo* sin llegar a ser *soporte largo*.

Para el resto de repositorios, como en Debian, la versión en el repositorio *estable* pasa a ser *antiguo*, *pruebas* será la versión *estable* y *desarrollo* toma el rol de *pruebas*.

Los repositorios reciben actualizaciones de seguridad automática desde Debian, dispuesto como se explicó en la sección Componentes de Debian.

Derivados

Canaima se define como metadistribución, lo que quiere decir que de ella provee los mecanismo para construir otras distribuciones derivadas. En Canaima estas derivaciones se les conoce como "Sabores". A lo largo de su historia son varias las distribuciones hijas que se derivaron de Canaima, entre ellas destacan.

Canaima Educativo

Proyecto bandera de Canaima. Destinado a suplir el sistema operativo que es entregado en las portátiles del programa Canaima Educativo[59].

Página: http://wiki.canaima.softwarelibre.gob.ve/index.php/Canaima_Educativo

Canaima Comunal

Versión de Canaima destinada a los Consejos Comunales.

Página: http://wiki.canaima.softwarelibre.gob.ve/index.php/Canaima_Comunal

[59] http://www.canaimaeducativo.gob.ve/

Canaima Colibrí

Derivación de Canaima cuya característica principal es la ligereza, lo que le permite correr en equipos de bajos recursos.

Página: http://wiki.canaima.softwarelibre.gob.ve/index.php/Canaima_Colibri

Canaima Caribay

Distribución enfocada a los medios de comunicación. Las primeras versiones estuvieron dirigidas a radios y medios impresos[60], las últimas versiones han apuntado a la televisión digital abierta incluyendo una versión para arquitecturas ARM[61].

Página: http://caribay.cenditel.gob.ve/

Canaima Comunitario

Aún no es una realidad. Es la versión que pretende ser construida por la comunidad, dirigida al público en general y basada en Canaima Pruebas para brindar una opción con paquetes más actualizados.

Estimación de costos

La Tabla 3.12 expone un resumen con los datos obtenidos del análisis realizado sobre el componente *usuarios* de Canaima. Se estudiaron 35 paquetes de software y se reconocieron 19 lenguajes de programación.

El estudio se hace sobre el repositorio *repositorio.canaima.softwarelibre.gob.ve*, lugar donde residen los paquetes nativos de Canaima.

[60] http://caribay.cenditel.gob.ve/fundamentacio/que-contiene-caribay/
[61] https://canaima.cenditel.gob.ve/trac/wiki/proyectos/caribaytda

Tabla 3.12: SLOC agrupados por lenguaje para Canaima 5.0

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**

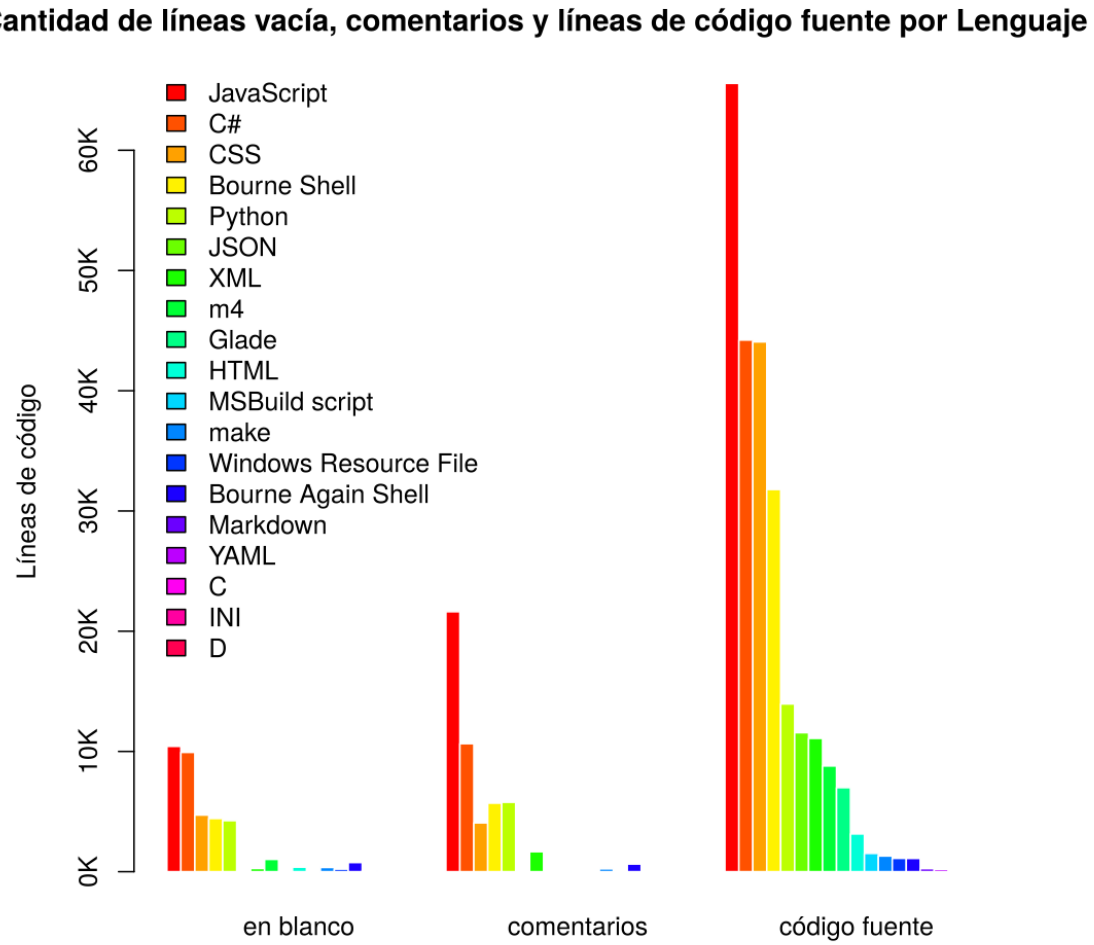
Fecha del archivo Source: 01/06/2016

Número de paquetes descargados: 35

Cantidad de lenguajes reconocidos: 19

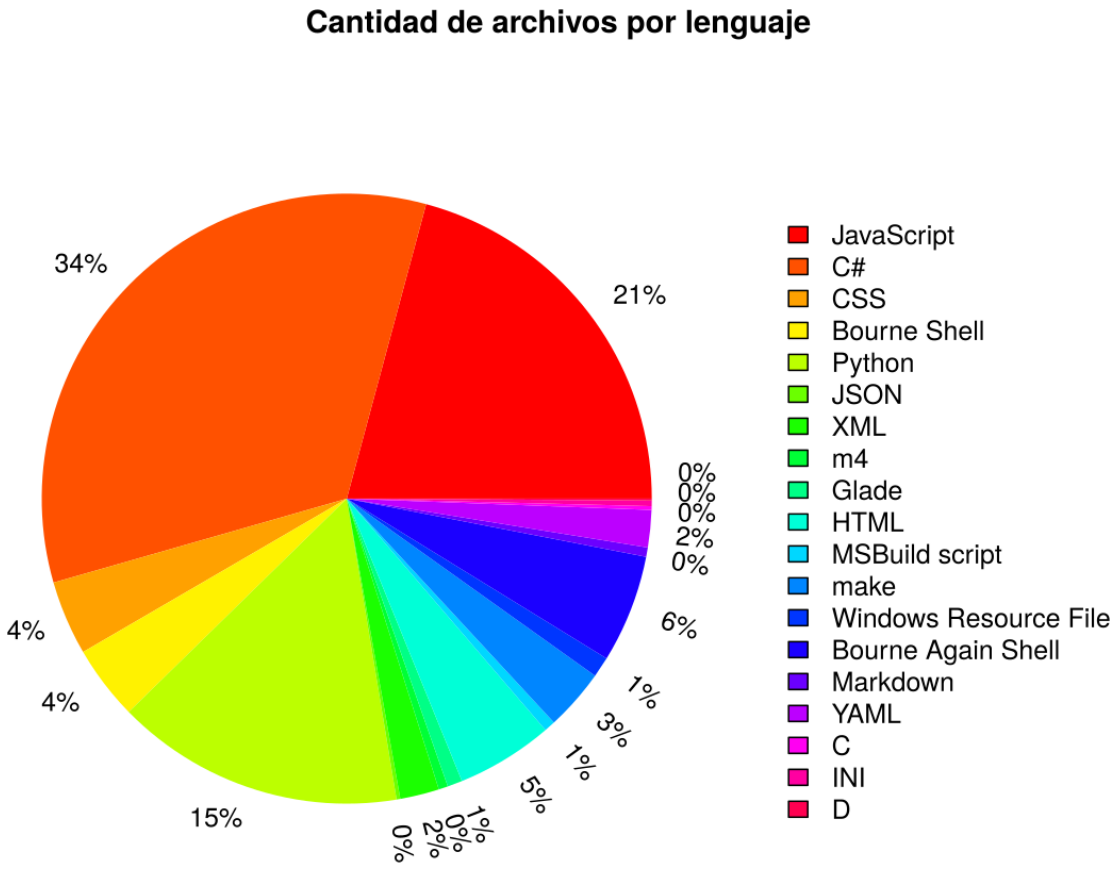
Se puede observar que JavaScript es el lenguaje que sobresale en cantidad de líneas en blanco, líneas con comentarios y líneas de código fuente. Ver Figura 3.22.

Figura 3.22: Cantidad de líneas vacía, comentarios y líneas de código fuente en los lenguajes del componente *usuarios* de Canaima 5.0



En cuanto a cantidad de archivos, es C# quien puntea haciéndose de un tercio aproximadamente (34%) en proporción de la totalidad de archivos, seguido de los archivos de JavaScripts con un 21% y los scripts de Bourne Shell con 15% para completar los archivos mas populosos dentro de los paquetes nativos de Canaima 5.0. Ver Figura 3.23.

Figura 3.23: Porcentaje de archivos por lenguaje dentro del componente *usuarios* de Canaima 5.0



COCOMO

Para el componente usuarios de Canaima GNU/Linux, el total de código fuente de todos los paquetes analizados en la versión 5.0 (Chimantá) es de 246.648 líneas. La Tabla 3.13 presenta los valores obtenidos aplicando COCOMO Básico.

Tabla 3.13: Estimaciones de esfuerzo y costos para el componente *usuarios* de Canaima aplicando COCOMO Básico 5.0

Líneas de código fuente:	246.648
Esfuerzo estimado de desarrollo (persona–mes):	779,64
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[62].

Por consiguiente para enero de 2017, el desarrollo de la paquetería nativa de Canaima tendría un costo estimado de US$ 1.833.612,33; teniendo la referencia del salario calculado a US$ 73.837,00 y necesitaría un equipo de 25 personas aproximadamente trabajando en el proyecto para completarlo en 2 años y medio aproximadamente.

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

Capítulo 4

Estado de desarrollo y esfuerzo de la Distribución Canaima GNU/Linux

---

Un poco de historia

Luego del paro petrolero ocurrido entre los años 2002 y 2003, el gobierno de Venezuela inicia una política hacia la tecnología que busca el control e independencia de su aparataje científico-tecnológico. Canaima es uno de esos proyectos sobrevenidos de la nueva política nacional contemplada en los planes de ciencia y tecnología.

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

Tres años luego del Decreto 3.390, 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) 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)

En Gaceta Oficial número 39.633 de fecha 14 de marzo de 2011, se publica la Resolución Nº 025 con fecha de 1 de marzo de 2011 donde resuelve en su Artículo 1,

> Establecer el uso de Canaima GNU/Linux como sistema operativo de Software Libre en las estaciones de trabajo de los órganos y Entes de la Administración Pública Nacional de la República Bolivariana de Venezuela, con el propósito de homogeneizar y fortalecer la plataforma tecnológica del Estado Venezolano, así como dar cumplimiento al marco legal vigente.
> (Resolución Nº 025, 2011)

La Resolución establece el uso de la distribución Canaima GNU/Linux así como el proceso de migración en todas las estaciones de trabajo de la Administración Pública Nacional.

Proceso evolutivo

2007 – Versión 1.0

"La primera versión de Canaima (1.0) fue una remasterización[1] de Debian que sirvió para migrar las oficinas internas del CNTI y comenzar el plan piloto de migración" (Martínez, 2013). Fue generada por un pequeño equipo del CNTI, liderado por Ernesto Hernández-Novich (Figuera, 2013).

Hernández-Novich (2009), involucrado en el desarrollo de esta primera versión explicaba, "Canaima es una adaptación de Debian, utilizando prácticamente el mismo instalador sólo que haciendo menos preguntas y tomando decisiones a priori".

2009 – Versión 2.0

Para la segunda versión la tarea de crear la distribución fue encomendada a la Cooperativa ONUVA RL. Con esta versión empieza a germinar la plataforma tecnológica del proyecto. Se dispone de "un repositorio propio de paquetes, un método de construcción optimizado y una imagen remozada representaban un proceso de innovación del entonces proyecto Canaima" (Muñoz y Mejías, 2010).

2009 – Versiones 2.0.X

Martínez (2010) afirma que las innovaciones incorporadas a Canaima GNU/Linux en las versiones 2.0.1, 2.0.2, 2.0.3 y 2.0.4 tendían a subsanar detalles funcionales menores, proteger al sistema contra fallas de seguridad y adoptar nuevas aplicaciones.

En principio, para las dos 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. Adicionalmente, los equipos de desarrollo eran compuestos por un pequeño grupo de personas y la participación de la comunidad no era aún significativa.

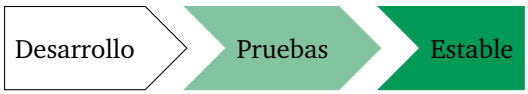
[1] Se conoce como remasterización en los ambientes de construcción de Sistemas Operativos basados en Linux a la modificación/personalización de una imagen ISO.

2010 – Versión 2.1

A partir de la versión 2.1 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 de toma de decisiones y construcción de la distribución. Inicia de este modo el proceso de apropiación del conocimiento para la construcción de un proyecto nacional.

Con La versión 2.1 empiezan a mejorar los canales de construcción colectiva, dice Martínez (2010) "Con el lanzamiento de la versión candidata Canaima GNU/Linux 2.1, comienza a capitalizarse un nuevo estilo tecnológico, afecto a prácticas comunitarias y esquemas de producción social. Los desarrollos tecnológicos de la versión 2.1 representan la aceleración, maduración y tecnificación del ciclo productivo de Canaima".

Figura 4.1: Ciclo de desarrollo Canaima Aponwao 2.1



2011 – Versión 3.0

De acuerdo a Martínez (2011b), para esta versión se implementa el sistema de empaquetamiento basado en git-buildpackage así como también el versionamiento basado en git. Añade Martínez (2011b) que, "Se sistematizó también la creación de imágenes ISO para Canaima, mediante un desarrollo propio basado en el Proyecto Debian Live. Además, en el camino se documentó la gran mayoría del proceso de desarrollo". Otra novedad incorporada fue la implementación del sistema de gestión de tickets (TRAC) para el reporte de errores.

Figura 4.2: Ciclo de desarrollo Canaima Roraima 3.0

2012 – Versión 3.1

A partir de la versión 3.1 se adopta un nuevo modelo de desarrollo. Figura 4.3

Figura 4.3: Ciclo de desarrollo Canaima Auyantepuy 3.1

Este modelo se acerca a la estructura de repositorios de Debian. En esta versión se añaden dos ramas: un repositorio bautizado *Mondongo* que hará las veces del repositorio experimental de Debian; y la rama *Soporte-Largo* para hospedar por el plazo de dos años aproximadamente una versión marcada como "SL" (Martínez, 2011a). Todas las versiones mayores (3.0, 4.0, 5.0) serán marcadas como "SL" (soporte-largo) al momento que les toque abandonar la versión *Antiguo*.

El anuncio[2] de la publicación de la segunda versión candidata de Canaima 3.1 (3.1 VC2) vino acompañado de la información sobre los cambios implementados en los repositorios, tal como se había acordado en la 6ª Cayapa de Barinas.

Se crea un repositorio que unifica aquellos dos con los que se venía trabajando, quedando de la siguiente forma:

$$universo + repositorio = paquetes$$

universo.canaima.softwarelibre.gob.ve: repositorio con la paquetería Debian

repositorio.canaima.softwarelibre.gob.ve: paquetes nativos de Canaima

paquetes.canaima.softwarelibre.gob.ve: nuevo repositorio que fusiona el repositorio con la paquetería Debian y el repositorio de paquetes nativos.

Además, se dividen los paquetes en tres secciones:

main: Contiene los paquetes de Debian (*main*) + Canaima.

aportes: Contiene paquetes de Debian (*contrib*) + Debian Multimedia.

no-libres: Contiene paquetes de Debian (*non-free*).

[2] http://listas.canaima.softwarelibre.gob.ve/pipermail/discusion/2012-July/007021.html

2013 – Versiones 4.X

Para el inicio del ciclo de desarrollo se anuncia la intención de crear una plataforma colaborativa para subsanar las falencias en esta área. Birbe (2013) explica la meta planificada para este aspecto de la siguiente manera, “Implementación e Integración de una plataforma que permita la participación efectiva de la Comunidad Canaima en el desarrollo y orientación de la distribución (Incluye valoración y seguimiento para fortalecer valores como la iniciativa y la responsabilidad)”. En un principio esta plataforma era llamada Chamanes pero finalmente fue bautizada con el nombre Tribus.

Sobre Tribus, el Equipo de desarrollo de Canaima (2013) expone,

> Tribus es nuestra apuesta por conectar a todos los colaboradores que deseen aportar sus contribuciones en algunas de las múltiples áreas de trabajo de la Metadistribución Canaima GNU/Linux. Es también, además, un sistema que nos permitirá automatizar muchas tareas repetitivas o predecibles que forman parte del proceso productivo de las Comunidades de Software Libre.

El desarrollo del sistema Tribus se encuentra en este momento en estado de inactividad.

2015 – Actualidad

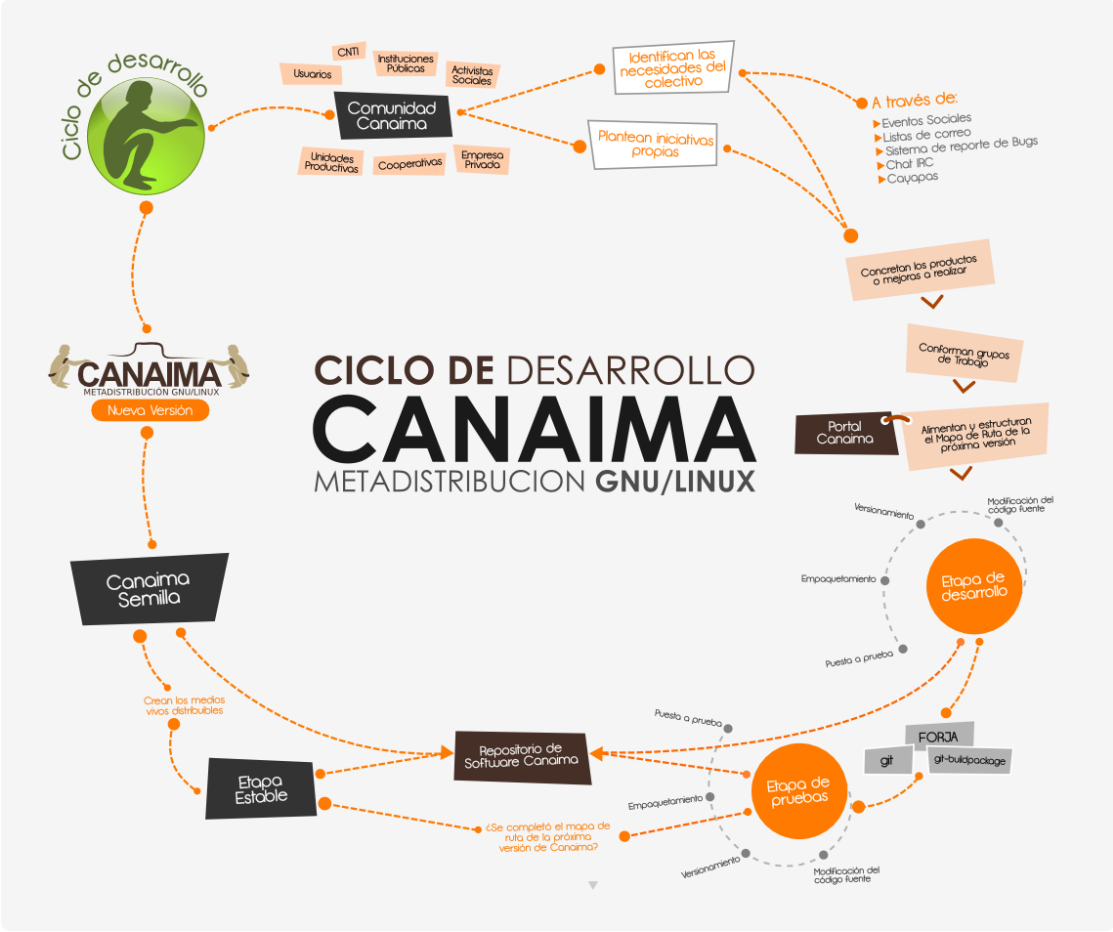
A través de las cayapas se ha intentado moldear los mecanismo para el desarrollo de la distribución. Se han realizado propuestas y en algunos casos con retrasos se han logrado implementar poco a poco.

En la mini-cayapa realizada en Caracas se realizó un trabajo en comunidad para la adecuación de los paquetes base para la versión 5.0, además se configuró un *buildbot* que ayudaría a automatizar el proceso de generación de la ISO.

El cuadro actual del ciclo de desarrollo de Canaima se presenta en la Figura 4.4 y fue explicado en la sección correspondiente al ciclo de desarrollo de Canaima de este documento.

Aún quedan deudas pendientes en el proceso. A nivel técnico, continuar la automatización de los procesos de adquisición de paquetes que serán integrados en la distribución. En lo organizativo, la definición de reglas y procedimientos para poder mejorar la cohesión de la comunidad en los procesos del proyecto.

Figura 4.4: Ciclo de desarrollo de Canaima GNU/Linux



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

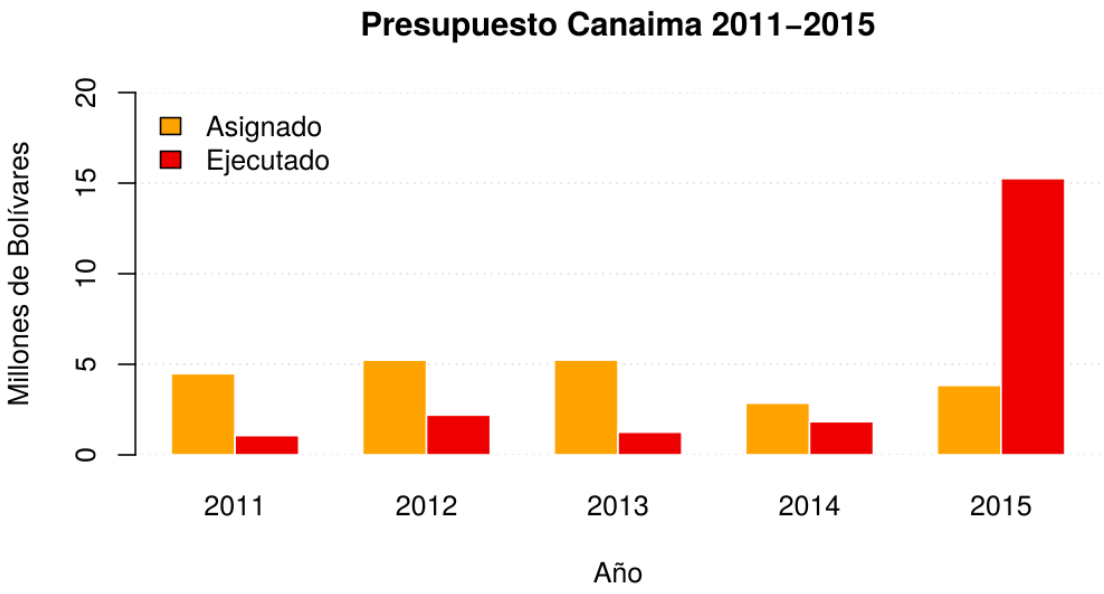
Presupuesto Canaima

Tomando datos de la página del Centro Nacional de Tecnologías de Información (CNTI)[3], se muestra en la figura 4.5 la relación del presupuesto asignado y el

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

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.

Figura 4.5: Presupuesto asignado al desarrollo de la distribución Canaima GNU/Linux



Datos: CNTI

El proyecto Guadalinex, una distribución GNU/Linux con apoyo financiero de la Junta de Andalucía, España; en el año 2016[4] 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[5].

Si se compara el presupuesto otorgado a Canaima con el presupuesto otorgado para el desarrollo de Guadalinex, se aprecia la gran diferencia de inversión en cada proyecto.

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

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

A continuación se hace un ejercicio para visualizar la diferencia de inversión entre ambos proyectos. Se plantean dos escenarios con fecha 1 de julio de 2016, el primer escenario con un presupuesto Bs. 5.000.000,00; valor cercano al máximo otorgado para el proyecto (2013). El segundo escenario corresponde a un presupuesto de Bs. 15.000.000,00; monto cercano al máximo ejecutado (2015) reflejado en la tabla. Para ambos casos se aplican los cálculos con cambio tasado a Bs. 630,00 por US$ 1,00; valor cercano al cambio oficial para mediados de 2016.

El escenario con presupuesto de Bs. 5.000.000,00 arroja una inversión de US$ 7.936,51; mientras que para el segundo escenario, con presupuesto de Bs. 15.000.000,00; la inversión otorgada corresponde a US$ 23.809,52. El mayor monto sigue siendo aproximadamente $\frac{1}{4}$ del monto otorgado al proyecto Guadalinex[6].

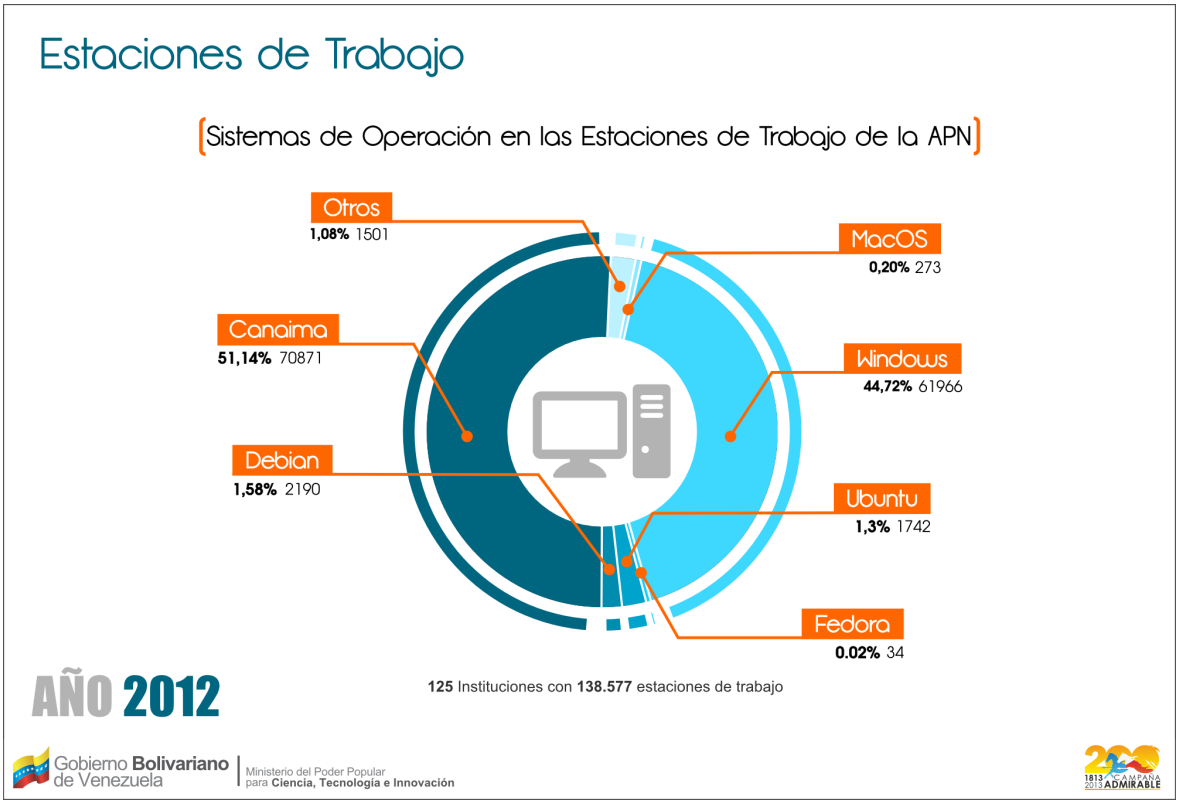
Migración nacional

El proceso de migración en Venezuela ha sido complejo, los datos y la información sobre este proceso es escaso o nulo. Por ejemplo, 2012 fue el último año donde se hicieron públicos los datos sobre el proceso de migración de software libre en la APN[7]. 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 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 4.6).

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 de algún Sistema Operativo distinto a este, y aquí cabría la pregunta, de haber maquinas con múltiples sistemas ¿cuál de los sistemas instalados era el sistema operativo de uso regular para el momento del censo?

[6]Para el 1 de julio de 2016 un Euro era equivalente a US$ 1,1105 al momento de la apertura.
[7]http://www.cnti.gob.ve/til-venezuela/estadisticas-de-migracion/estadisticas-de-migracion.html

Figura 4.6: 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)

Esfuerzo

La Tabla 4.1 muestra un resumen con los datos relacionados con el esfuerzo estimados en este trabajo. De esta tabla comparativa se obtienen los siguientes datos:

- Canaima incluye en su repositorio principal $\frac{1}{600}$ veces la cantidad de paquetes compartidos por Debian, y aproximadamente $\frac{1}{71}$ la de Ubuntu.
- La cantidad de líneas de código fuente de Canaima equivalen a un 0,02% de las escritas en el código entregado por los paquetes en Debian y del 0,10% del encontrado en Ubuntu.

- Si una persona fuese asignada para desarrollar todo el proyecto, le tomaría alrededor de 65 años completar los códigos proporcionados por el repositorio de paquetes nativos de Canaima; aproximadamente 89.293 años para los de Ubuntu y 405.431 años para escribir todo el código distribuido en el repositorio principal de Debian.
- El número de personas estimadas para completar el proyecto en Canaima es de aproximadamente 25 personas y les tomaría unos 2 años y medio.
- El número de personas estimadas para completar el proyecto en Ubuntu es 2.191 y les tomaría 40 años y 9 meses.
- El número de personas estimadas para completar el proyecto en Debian 5.598 y les tomaría 72,42 años.
- Los costos totales de la tabla indican sólo un estimado para codificación, no se toma en cuenta costos de electricidad, mercadeo, la construcción de las distribuciones como tal, equipos, gastos varios.

Tabla 4.1: Cuadro resumen de Estimaciones de Esfuerzo, Tiempo de Desarrollo, Personal requerido y Costo Total de los proyectos evaluados

	Debian	**Ubuntu**	**Canaima**
Paquetes procesados	20.981	2.495	35
Archivos procesados	4.326.711	1.054.974	1.014
Líneas de código fuente	1.015.218.011	240.297.751	246.648
Esfuerzo (personas–mes)	4.865.175,62	1.071.515,44	779,64
Tiempo de desarrollo (meses)	869,04	489,03	31,39
Personas requeridas	5.598,36	2.191,08	24,83
Costo total (US$)	413.366.347,61	161.783.066,47	1.833.612,33

Usuarios Linux a nivel mundial

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 más cuesta arriba, solamente se podría 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, la tabla 4.2 muestra el porcentaje de máquinas con Linux que han visitado alguna página monitorizada por estos sistemas.

Tabla 4.2: 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[8] como el número base de usuarios de internet para

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

mayo de 2017, y usando los valores máximos y mínimos de la tabla 4.2 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.

Usuarios Canaima

Para obtener 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, a 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 preciso alegar que Canaima “tiene la comunidad de Software Libre más grande del mundo con los más de 5 millones de usuarias y usuarias [sic] de todo el territorio nacional que tienen su canaimita.” (Hernández, 2016). Sin embargo, es un gran indicativo de la cantidad de usuarios a los que se le ha distribuido el sistema operativo nacional. Un número nada despreciable, sin dejar atrás la importancia significativa que representa el programa Canaima Educativo en el ámbito mundial del Software Libre. Es fundamental además, sumarle el número de computadoras que incorporaron Canaima en el plan “Internet equipado” de CANTV, plan que funcionó hasta hace pocos años; y por supuesto, las estaciones de trabajo que han sido migradas con éxito dentro de la APN.

Debian, Ubuntu y Canaima

Las distribuciones evaluadas, Debian, Ubuntu y Canaima, así como el resto de distribuciones GNU/Linux son proyectos que se encargan de reunir y hacer funcionar una gran cantidad de proyectos individuales, entiéndase paquetes software. La generación de una distribución es un trabajo de integración de cientos e incluso miles de componentes de software.

Tabla 4.3: Cuadro Comparativo

	Debian	**Ubuntu**	**Canaima**
Fuente de financiamiento	Comunitario	Privado	Gubernamental
Paquetes en repo principal[9]	20.981	2.495	35
Cantidad de desarrolladores	1.314	1.128	5
Periodicidad de lanzamientos	Bienal	Semestral	Anual
Soporte LTS (Años)	5	5	5

Canaima GNU/Linux es una distribución cuyo soporte financiero, técnico y humano proviene fundamentalmente 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 ser variado. Los objetivos de cada proyecto varían. El de uno podría ser práctico: entregar a la sociedad un producto útil que satisfaga las necesidades del usuario final. El del otro podría estar buscando lo mismo pero desde un sentido comercial. Y entre otros está aquel que buscaría suplir las necesidades de un usuario en concreto como podrían ser los funcionarios y empleados públicos de un país.

Para las versiones evaluadas, Debian dispone de poco más de 1.300 desarrolladores para integrar los casi 20.500 paquetes que posee el repositorio.

[9] Para las versiones evaluadas.

Ubuntu cuenta con un aproximado de 1.100 desarrolladores para la integración 2.495 paquetes. Canaima, por su parte, tiene 5 desarrolladores para la integración de 35 paquetes en la imagen ISO final.

Recuerde que la integración es sólo una de las muchas tareas que se requieren para la creación de una distribución.

Acerca de la participación de la comunidad en las listas de correo. Todas las listas evaluadas muestran una actividad tímida durante los últimos años. Un fenómeno interesante que podría tener múltiples interpretaciones. Por ejemplo, podría indicar que la usabilidad en los sistemas basados en Linux ha mejorado en los últimos años disminuyendo la necesidad de recibir soporte vía correo electrónico. También podría ser el indicativo de que la utilización del correo electrónico para la búsqueda de ayuda y soporte ha sido reemplazado por otros mecanismo alternos, y quizás más expeditos, como lo sería una búsqueda web.

Participación comunitaria

El desarrollo de Canaima inicia con un estilo del tipo catedral, con la contratación de minúsculos grupos de personas para el desarrollo de las dos primeras versiones. Pero además, todavía no termina de trascender por completo al estilo del tipo bazar, aún transita en esa etapa de migración de la que hablan Capiluppi y Michlmayr (2007).

Canaima es un ejercicio de construcción colaborativa bajo dos modelos que suelen ser antagónicos, el del colaborador y el del empleado que trabaja de entre 7 y 8 horas en una oficina, y a veces más, para cumplir los objetivos anuales de un proyecto. Se le añade además que en el caso de las dinámicas institucionales públicas nacionales es muy seguro que ese personal no se encuentre asignado exclusivamente a un solo proyecto.

Hoy día Canaima responde a un proyecto POA, un proyecto POA continuado, pues cada año es renovado y cuyas metas anuales deben ser cumplidas.

De este modo surgen varias interrogantes, los acuerdos conciliados en reuniones comunitarias (cayapa) en donde también participa el CNTI ¿son tomados en cuenta

a la hora de construir la planificación anual? En el caso de que el CNTI no creyera viable alguno de los puntos de acuerdo en las cayapas, ¿no es mejor sincerarse en la reunión y manifestarlo allí? Es preferible escuchar la argumentación de manera inmediata que luego de hacer planes sobre acuerdos que no serán cumplidos, en todo caso, no son prioritarios.

Es sabido que en los últimos años la mengua de desarrolladores en organismos de la APN se acentúa. La variación de desarrolladores en Canaima va entre 7 como máximo (2013–2014) y 2 como mínimo (2016). Este factor maximiza el riesgo de retraso, acumulación de tareas y sobrecarga de trabajo que inciden de manera negativa sobre el trabajador potenciando la posibilidad de acrecentar la migración de personal.

Una distribución nacional no debería contar con un personal de 12 personas[10] para atender un proyecto que implica la atención de una plataforma e infraestructura tecnológica de gran envergadura que capaz de asegurar el correcto funcionamiento de los servicios que se prestan y que son utilizados. Adicionalmente, no se deben descuidar el resto de tareas operativas del proyecto, como las labores de diseño, soporte, desarrollo, documentación, sistematización, difusión, formación, enlaces institucionales, etcétera.

Todas estas condiciones deben valorarse al momento de preparar el plan de trabajo. Evitar en lo posible caer en la proyección de metas con alto riesgo de fracaso.

Birbe (2013) hace un señalamiento sobre los acuerdos que recaen sobre el CNTI luego de las decisiones en comunidad. Comenta, “Las Cayapas son y han sido eventos de gran utilidad para Canaima, sin embargo, debemos notar que muchas de las decisiones, aunque son comunitarias, terminan quedando como responsabilidad del Equipo Canaima que labora en el CNTI”. Se percibe que luego de los acuerdos la comunidad deja de prestar un interés, al menos por un tiempo.

Caben más preguntas, ¿por qué sucede esto? ¿Dónde recae la responsabilidad? ¿Dónde está el seguimiento a los compromisos adquiridos por las partes concordantes?

Por parte del equipo Canaima se ha esgrimido la falta de apoyo a nivel de código.

[10] Según página del CNTI.

Es justo mencionar también que históricamente la mayor parte de contribuciones de código ha venido del ente responsable de Canaima.

La participación y distribución de paquetes para la versión 5.0 de Canaima se expone a continuación. Esta versión aloja 35 paquetes nativos en el repositorio *Usuarios* de Canaima. Estos 35 paquetes son mantenidos entre 10 personas. De las 10 personas, 5 son trabajadores del CNTI y 5 de la comunidad. El mantenimiento de los paquetes se distribuye en 8 paquetes por la comunidad y 27 por parte del equipo del CNTI. La proporción por consiguiente queda con 77,14% de los paquetes mantenidos por CNTI y un 22,86% mantenidos por la comunidad para una distribución igual en el número de mantenedores representados con el 50% cada uno.

De los mantenedores de la versión en estudio, el 20% es de género femenino, el 60% de género masculino y un 20% son equipos de trabajo.

En cuanto a la participación de la comunidad, cabe evaluar ¿no se prestan las condiciones para ello por parte de los responsables de la plataforma? ¿Está bien documentado y visible los procesos para la colaboración a nivel de desarrollo? ¿El proceso para hacer los aportes es engorroso?

Birbe (2013) nuevamente apunta sobre el tema, y plantea lo que podría ser uno de los motivos de esta situación,

> El constante cuestionamiento y autocrítica que nos hacemos en lo que respecta a este tema de la participación comunitaria nos ha llevado a entender que, entre otras cosas, la comunidad no participa tan activamente como anhelamos, **No por falta de ánimo e iniciativa** como cualquiera podría concluir aceleradamente, sino **más bien por falta de las herramientas adecuadas de participación**."

Por todo esto es imperativo un trabajo conjunto y coordinado entre los actores; con objetivos claros y alcanzables; con las condiciones tecnológicas y humanas acordes a los objetivos de la nación que se quiere; en un ambiente de confianza, cooperación y respeto.

Para lograr un verdadero ambiente colaborativo libre no solo hace falta la apropiación del software sino también de los procesos.

Capítulo 5

Conclusiones y recomendaciones

El ciclo de desarrollo que presenta la Distribución Canaima dibuja un comportamiento cíclico de 5 fases para cada ciclo (o versión). Estas fases abarcan los procesos de inicialización, planificación, desarrollo, pruebas y publicación.

Pero este ciclo se ve afectado cuando variables imprevistas son introducidas en algún o algunos de los procesos del ciclo. Las variables más comunes en las instituciones de tecnología del sector publico nacional como el Centro Nacional de Tecnologías de Información (CNTI) o el Centro Nacional de Desarrollo e Investigación en Tecnologías Libres (CENDITEL) son entre otras:

- Introducción de nuevas metas en la planificación pautada.
- Asignación del trabajador a nuevos proyectos que no estaban contemplados al inicio del plan y que manejan mayores prioridades repercuten en el tiempo que le queda para dedicarle a los proyectos originales.
- Salida de personal calificado de los proyectos/institución.
- Pérdidas de equipo por fallas imprevistas y no hay forma de reponer.

Adicional a esto, la problemática presentada en el caso de Canaima donde aún no se ha terminado de dar el salto a un modelo del estilo bazar atenta contra el objeto colaborativo del proyecto.

Es evidente que el nivel de participación de los usuarios en el proyecto nacional es bajo. Al comparar el nivel de participación que tienen otras comunidades de distribuciones Linux queda claro que algo está fallando en la forma como se ha manejado el proyecto a nivel comunitario. Algunas políticas llevadas en el pasado por quienes han dirigido las instituciones no han permitido incorporar en la planificación los acuerdos convenidos, por la comunidad y en comunidad. Añadido a esto, las plataformas instaladas para la cooperación aún no son suficientes o no están cumpliendo los objetivos para la incorporación y participación de nuevos colaboradores.

Canaima tiene una gran historia que contar, tiene un banco de información que debe ser organizado, y otro más que debe rescatarse. La documentación es esencial para poder llevar a cabo los procesos de apropiación, formación y divulgación en torno al proyecto.

Para cumplir los propósitos para lo que fue concebido Canaima, el Estado debe buscar asegurar las condiciones máximas posibles (y no mínimas como se suele decir) para su funcionamiento. Esto abarca, planificación, infraestructura, plataforma, empleo digno y comunidad.

Recomendaciones

Anterior expuesto se propone una serie de recomendaciones:

En lo técnico:

- Continuar con la automatización de procesos de construcción y generación de la distribución.
- Ir trabajando en los acuerdos alcanzados en las cayapas.

En lo organizacional:

- Definir estrategias claras de colaboración.

- Poner en práctica el seguimiento a las tareas, actividades y asignaciones.

En lo documental:

- Documentar, documentar y documentar.
- Actualizar la base de conocimiento publicada en los diferentes portales de Canaima.
- Continuar con el trabajo de organización de todos estos documentos.
- Completar aquellos documentos que aún quedan por concluir.
- Rescatar en lo posible los documentos históricos de las cayapas. El wiki[1] utilizado históricamente para registrar las actividades y acuerdos realizados en las cayapas dejó de funcionar y toda la documentación que compilada allí está inaccesible. Existía otro portal[2] con dominio del CNTI, con información, al menos hasta la VII Cayapa que también está inaccesible.

Respecto CNTI:

- Aumentar el personal para el proyecto.
- Ofrecer un salario justo y competitivo.
- Establecer la dedicación exclusiva a las tareas de Canaima.
- **Respetar los acuerdos alcanzados con la comunidad.**
- Mantener las metas planificadas (no añadir a destiempo nuevas metas).
- **No añadir requerimientos nuevos fuera del plazo de planificación.**
- El plan de migración nacional debe ser redefinido.

[1] http://cayapa.canaima.net.ve/
[2] http://cayapa.canaima.softwarelibre.gob.ve/

Respecto del Gobierno:

- Asegurar las condiciones tecnológicas favorables para poder mantener una plataforma estable.
- Proveer la infraestructura que permita el desarrollo tecnológico de la población.
- Saldar la deuda jurídica, por ejemplo, el Reglamento de la Ley de Infogobierno.

A futuro:

- Deslindar el proyecto de la red burocrática nacional y constituir una figura independiente, como por ejemplo una Fundación. Esta ha sido una propuesta que lleva años en el tapete.

  Ya en 2010 se asomó un primer intento de conformar una fundación[3].

Otra opción:

- Dedicar el máximo esfuerzo a la creación de paquetes y aplicaciones libres que satisfagan las necesidades de la APN y dar un paso atrás con la idea de una distribución. En otras palabras, nutrir un repositorio nacional de aplicaciones libres compatibles con la mayoría de las distribuciones y que tengan como objetivo dar solución a los requerimientos informáticos de las instituciones públicas.

[3] http://listas.canaima.softwarelibre.gob.ve/pipermail/discusion/2010-November/002227.html

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 raíz.


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$code), digits=2, nsmall=2,
       decimal.mark=",", big.mark=".", scientific=FALSE),"\n")
45 cat("Líneas de código fuente (K): \t", format(KSLOC, digits=2, nsmall=2, decimal
       .mark=",", big.mark=".", scientific=FALSE),"\n\n\n")
46 cat("Estimaciones aplicando el Modelo COCOMO Básico\n")
47 cat("============================================================\n")
48 cat("Esfuerzo estimado de desarrollo:\t", format(Epm, digits=2, nsmall=2,
       decimal.mark=",", big.mark=".", scientific=FALSE),"\n")
49 cat("Tiempo de desarrollo estimado:\t\t", format(Tdev, digits=2, nsmall=2,
       decimal.mark=",", big.mark=".", scientific=FALSE),"\n")
50 cat("Personas requeridas estimadas:\t\t", format(Per, digits=2, nsmall=2,
       decimal.mark=",", big.mark=".", scientific=FALSE),"\n")
51 cat("\nCosto total estimado del proyecto(*):\t", format(Ctd, digits=2, nsmall=2,
        decimal.mark=",", big.mark=".", scientific=FALSE), "\n\n")
52 cat("(*) con US$", format(Spa, digits=2, nsmall=2, decimal.mark=",", big.mark=".
       ", scientific=FALSE), "anual como referencia para enero de 2017\n")
53 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.

Birbe, E. (2013). [Desarrolladores] Disponible primera versión alfa de Canaima 4.0. Correo Electrónico. Recuperado de: http://listas.canaima.softwarelibre.gob.ve/pipermail/desarrolladores/2013-May/006614.html. Consultado el 03 de abril de 2017.

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.

Centro Nacional de Tecnologías de Información (2017). ¿Qué es Canaima? Recuperado de http://canaima.softwarelibre.gob.ve/canaima/que-es-canaima.

Colina, H. (2014a). [Discusion] Normas de discusión de las listas de Canaima. Correo Electrónico. Recuperado de: http://listas.canaima.softwarelibre.gob.ve/pipermail/discusion/2014-November/011836.html. Consultado el 27 de marzo de 2017.

Colina, H. (2014b). [Discusion] Noticias de la 8ava Cayapa Canaima. Correo Electrónico. Recuperado de: http://listas.canaima.softwarelibre.gob.ve/pipermail/discusion/2014-November/011835.html. Consultado el 27 de marzo de 2017.

Dávila, J., Carrero, J., Molina, J., Díaz, G., y Hernández, D. (2007). ULAnix Scientia: Software a la medida de Científicos y Tecnólogos. Memorias, II Encuentro Nacional de Actores de Popularización de la Ciencia. pp.133-134.

Decreto Nº 3.390 (2004). República Bolivariana de Venezuela. Gaceta Oficial Nº 38.095 del 28/12/2004. Recuperado de http://historico.tsj.gob.ve/gaceta/diciembre/281204/281204-38095-08.html. Consultado el 24 de marzo de 2017.

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

Equipo de desarrollo de Canaima (2013). [Desarrolladores] Lanzamiento de la primera versión alfa de Tribus. Correo Electrónico. Recuperado de: http://listas.canaima.softwarelibre.gob.ve/pipermail/desarrolladores/2013-December/008014.html. Consultado el 03 de abril de 2017.

Equipo ULAnux (2007a). ULAnix. Recuperado de https://web.archive.org/web/20070611050145/http://nux.ula.ve/index.php?option=com_content&task=view&id=14&Itemid=48.

Equipo ULAnux (2007b). ULAnix Lógica (Beta4 r1). Recuperado de https://web.archive.org/web/20070611050016/http://nux.ula.ve/index.php?option=com_content&task=view&id=13&Itemid=2.

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).

Helmke, M. y Graner, A. (2012). *The Official Ubuntu Book*. Prentice Hall, Upper Saddle River, NJ, USA, 7ma edición.

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

Hernández-Novich, E. (2009). Pregunta sobre Canaima - sin ánimos de polémica. Correo Electrónico. Recuperado de: http://l-linux.velug.org.narkive.com/C9O6OzLH/pregunta-sobre-canaima-sin-animos-de-polemica#post4. Consultado el 03 de abril de 2017.

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 línea. 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.

Jefe de Prensa (2011). Esquemas de trabajo colaborativo se acordaron en primera Jornada de 5ta Cayapa Canaima. Ministerio del Poder Popular para Educación Universitaria Ciencia y Tecnología. Recuperado de https://www.mppeuct.gob.ve/actualidad/noticias/esquemas-de-trabajo-colaborativo-se-acordaron-en-primera-jornada-de-5ta-cayapa.

Ley del Plan de la Patria (2013). República Bolivariana de Venezuela. Ley del Plan de la Patria. Segundo Plan Socialista de Desarrollo Económico y Social de la Nación 2013–2019. Gaceta Oficial Nº 6.118 Extraordinario del 4/12/2013. Recuperado de https://www.mppeuct.gob.ve/sites/default/files/descargables/proyecto-nacional-simon-bolivar.pdf. Consultado el 24 de marzo de 2017.

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. (2010). [Discusion] Lanzamiento de la Versión Candidata Canaima GNU/Linux 2.1. Correo Electrónico. Recuperado de: http://listas.canaima.softwarelibre.gob.ve/pipermail/discusion/2010-May/001325.html. Consultado el 28 de marzo de 2017.

Martínez, L. (2011a). [Desarrolladores] Estado de los Repositorios para el 30/12/11. Correo Electrónico. Recuperado de: http://listas.canaima.softwarelibre.gob.ve/pipermail/desarrolladores/2011-December/004615.html. Consultado el 02 de abril de 2017.

Martínez, L. (2011b). [l-discusion] Lanzamiento de la primera Versión Candidata (VC1) de Canaima 3.0. Correo Electrónico. Recuperado de: http://listas.canaima.softwarelibre.gob.ve/pipermail/discusion/2011-February/003014.html. Consultado el 02 de abril de 2017.

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.

Martínez, L. (2013). Tribus: Concretando el esquema de participación comunitaria en Canaima. Recuperado de http://huntingbears.com.ve/tribus-concretando-el-esquema-de-participacion-comunitaria-en-canaima.html.

Ministerio de Ciencia y Tecnología (2005). Plan Nacional de Ciencia, Tecnología e Innovación. Construyendo un futuro sustentable Venezuela 2005–2030. Recuperado de www.cenditel.gob.ve/static/biblioteca/2005/pncti/pncti.pdf. Consultado el 24 de marzo de 2017.

Molina, J., Díaz, G., Carrero, J., y Dávila, J. (2007). ULAnuxULAnix: Software Académico a la Medida. Recuperado de http://webdelprofesor.ula.ve/ingenieria/jacinto/publica/2007/ulanix/ulanix.pdf.

Muñoz, J. y Mejías, J. (2010). Gestion colaborativa de la plataforma tecno productiva de Canaima. Recuperado de http://wiki.canaima.softwarelibre.gob.ve/index.php/Gestion_colaborativa_de_la_plataforma_tecno_productiva_de_Canaima#La_historia_incompleta_de_proyecto_Canaima.

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/.

Petrizzo, M. y Muñoz, J. (2012). Propuesta de modelo organizacional para el Proyecto Canaima GNU/Linux. Recuperado de https://www.academia.edu/2438143/Modelo_Organizacional_para_Canaima_GNU_Linux_Extenso_.

Presidencia de la República Bolivariana de Venezuela (2007). Proyecto Nacional Simón Bolívar. Primer Plan Socialista -PPS- Desarrollo Económico y Social de la Nación 2007–2013. Recuperado de https://www.mppeuct.gob.ve/sites/default/files/descargables/proyecto-nacional-simon-bolivar.pdf. Consultado el 24 de marzo de 2017.

Pressman, R. (2010). *Ingeniería del Software. Un enfoque práctico*. 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.

Resolución Nº 025 (2011). República Bolivariana de Venezuela. Gaceta Oficial Nº 39.633 del 14/03/2011. Recuperado de http://historico.tsj.gob.ve/gaceta/marzo/1432011/1432011-3093.pdf. Consultado el 24 de marzo de 2017.

Solano, S., Reaño, Y., Vázquez, F., Jerez, Y., Mora, G., López, M., Guerrero, C., Infante, J., Martínez, L., Mejías, J., Aguilera, D., Barragán, N., Palma, M., Parra, C., Bruzzo, E. L., González, C., y Muñoz, J. (2011). Papel de Trabajo para la Creación de las Políticas de Participación en la Plataforma Colaborativa Canaima. Recuperado de http://wiki.canaima.softwarelibre.gob.ve/index.php/Papel_de_Trabajo_para_la_creaci%C3%B3n_de_las_Pol%C3%ADticas_de_Participaci%C3%B3n_en_la_Plataforma_Colaborativa_Canaima.

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.

Wikimedia Commons (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. [Internet; visitada 01-julio-2017].

Wikipedia (2008). ULAnix. Recuperado de https://web.archive.org/web/20080405095002/https://es.wikipedia.org/wiki/ULAnix.

Wikipedia (2017). Benevolent Dictator for Life — Wikipedia, La enciclopedia libre. Recuperado de https://es.wikipedia.org/w/index.php?title=Benevolent_Dictator_for_Life&oldid=101858617. [Internet; visitada 14-septiembre-2017].