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

Por

TMI David A. Hernández Aponte

Tutor: Dr. Jacinto Dávila

Mayo 2017

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

TMI David A. Hernández Aponte

Proyecto de Grado — Sistemas Computacionales, 42 páginas

Resumen: Se desea realizar un estudio que muestre el ciclo de vida del desarrollo de software libre aplicado a una distribución Linux, usando como objeto de investigación la distribución Canaima GNU/Linux, para ello se mostrará la manera en que se ha llevado a cabo su desarrollo través de la historia de la distribución, y se hará una comparación con otras distribuciones, como búsqueda para establecer casos de éxito y/o fracaso en las diferentes fases de desarrollo de la misma.

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

Este trabajo fue procesado en LaTeX.

Índice general

Capítulo 1

Introducción y Antecedentes

Introducción

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

Antecedentes

La Organización Internacional para la Estandarización (ISO en inglés) publica 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" (ISO/IEC 12207:2008, p.4). También se define *modelo de ciclo de vida* como el "marco de trabajo 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, p. 4).

El Institute of Electrical and Electronics Engineers (1997) en su norma para la creación de procesos de ciclo de vida 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." (p. 5). Un *modelo de ciclo de vida del software* es definido 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." (p. 5). Finalmente se define *procesos de ciclo de vida del software* como "la descripción de proyectos específicos de los procesos basados en el ciclo de vida del software de un proyecto, y los activos del proceso de la organización." (p. 5).

Amor et al. (2005a) realizaron una investigación en la que utiliza la versión 3.1 de la distribución Debian como caso de estudio de medición de software libre. Aquí se concluye que el crecimiento del trabajo en Debian se incrementa año tras año siendo uno de los mayores sistemas de software mundial, e independientemente de la envergadura del proyecto la comunidad de mantenedores y voluntarios que rodean el sistema goza de buena salud. Sin embargo, se cuestiona la sostenibilidad del proyecto a futuro, basándose en el tamaño medio de la distribución, señalando que este comportamiento se podría deber a un crecimiento de números de paquetes más rápido que el de mantenedores.

Por su parte, Capiluppi y Michlmayr (2007) realizaron un estudio empírico del ciclo de vida de los proyectos basados en comunidades de voluntarios, concluyendo que los proyectos de software libre comienzan con una fase catedral, de desarrollo cerrado y de pocos desarrolladores, y luego migran a una fase bazar, desarrollo con un gran número de voluntarios y contribuciones[1].

[1]Tomando como referencia el ensayo *La Catedral y el Bazar* de Raymond (1999)

Planteamiento del Problema

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

Actualmente, la distribución Canaima GNU/Linux es poco usada y recibe pocas contribuciones. El presente trabajo pretende mostrar la viabilidad que tiene la distribución Canaima GNU/Linux para recibir contribuciones por parte de la comunidad de desarrolladores, entendiéndose principalmente en el apartado para la posible admisión de nuevos paquetes de software en los repositorios oficiales distribución.

Objetivos

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

Objetivo General

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

Objetivos Específicos

1. Caracterizar los ciclos de vida del software libre, identificando indicadores cuantivativos y cualitativos de cada etapa.

2. Recolectar datos relevantes a esos indicadores para un conjunto de distros referenciales.

3. Comparar la distro Canaima GNU/Linux con esas distribuciones referenciales.

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

5. Conclusiones y recomendaciones.

Capítulo 2

Sobre el ciclo de vida del software libre y métodos para analizarlo

---

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).

La International Organization for Standardization (2008) plantea lo siguiente,

> 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

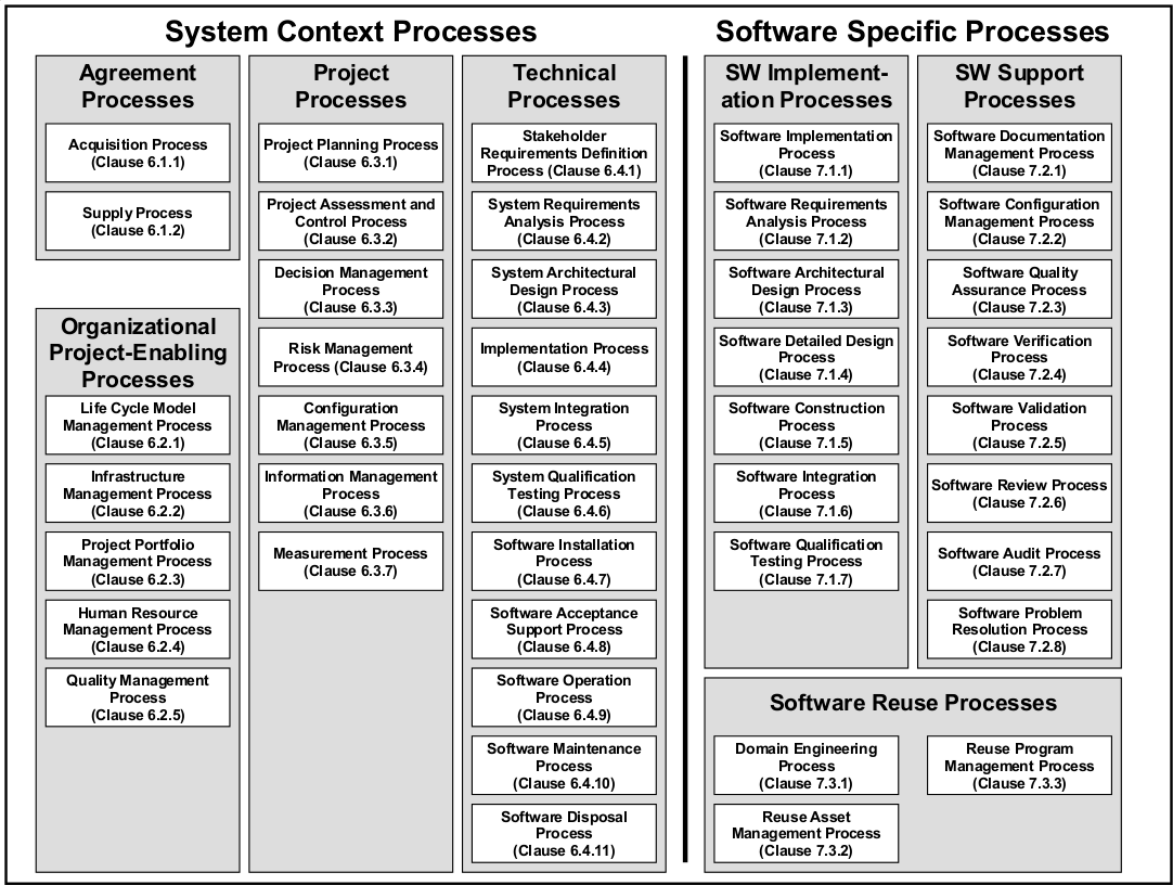
propósito y los resultados de esa fase. (p.12)

Asimismo, la International Organization for Standardization (2008) agrupa las actividades que pueden ser ejecutadas durante el ciclo de vida de un sistema de software en siete grupos de 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. (p.13).

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.

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., sino que pretende ser una referencia general que las organizaciones adaptarían a sus necesidades.

Figura 2.1: Grupos de los procesos del ciclo de vida



Fuente: International Organization for Standardization (2008)

> El modelo de referencia de procesos no representa un enfoque de implementación de procesos particular ni prescribe un modelo, metodología o técnica de ciclo de vida del sistema/software. 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 de la organización es adoptado por los proyectos de la organización en el contexto de los requisitos del cliente.
> (International Organization for Standardization, 2008, p. 14)

Cantidad de líneas de Código Fuente

La 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 (COCOMO) Boehm (1981) en su variante más actualizada COCOMO II Boehm et al. (2000).

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

Para poder obtener el número de líneas de código fuente se ha optado por trabajar con `cloc`[1]. `Cloc` es un programa con licencia libre, multiplataforma, hecho en Perl, que cuenta las líneas en blanco, líneas con comentarios y las líneas físicas de código fuente reconociendo más de 200 lenguajes de programación.

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

Capítulo 3

Datos referenciales de las más importantes distribuciones de software

---

Debian

Definición

Lowe et al. (2015) establecen que el proyecto Debian fue creado por Ian Murdock en agosto de 1993. (p.9). En la página web del proyecto[1] definen a Debian como un sistema operativo libre, dispone de versiones para el núcleo Linux o el núcleo FreeBSD. Además, Debian se presenta con mas de 43000 paquetes de software.

Componentes

Debian categoriza los paquetes incluidos en su sistema dentro de tres repositorios según su compatibilidad con las directrices de software libre de Debian (DFSG)[2],

[1] https://www.debian.org/intro/about
[2] https://www.debian.org/social_contract.es.html#guidelines

estos repositorios 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.

Arquitecturas

Actualmente Debian soporta las arquitecturas mostradas en la tabla 3.1.

Tabla 3.1: 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)
powerpc	Motorola/IBM PowerPC
ppc64el	POWER7+, POWER8
s390x	System z

Fuentes: Debian[3]

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

Roles

En la comunidad de desarrollo de Debian se pueden distinguir diferentes roles:

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

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

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

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

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

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

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

Colaboradores

- 271 Mantenedores Debian[4]
- 1043 Desarrolladores Debian[5]

Política

- Contrato social de Debian https://www.debian.org/social_contract

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

Histórico de distribuciones

- Debian http://cdimage.debian.org/mirror/cdimage/archive/

Versiones Debian

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

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

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

Lanzamientos

La Tabla 3.2 muestra la fecha de lanzamiento de las diferentes versiones de la distribución Debian.

Tabla 3.2: Lanzamientos de versiones de Ubuntu

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	noviembre 1995
1.1	Buzz	17/06/1996
1.2	Rex	12/12/1996
1.3	Bo	05/06/1997
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

Fuentes: Lowe et al. (2015) y Debian[6]

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, The

[6] https://www.debian.org/doc/manuals/project-history/ch-releases.html

Ubuntu Manual Team (2016), 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.

Componentes

Ubuntu categoriza los paquetes incluidos en su sistema dentro de cuatro repositorios, de acuerdo a si son libres o no, según la Filosofía de Software Libre de Ubuntu[7], estos repositorios son:

Main: Software libre y abierto mantenido por Canonical.

Universe: Software libre y abierto mantenido por la comunidad.

Multiverse: Software con restricciones de licencia.

Restricted: Controladores propietarios.

Arquitecturas

Ubuntu está oficialmente soportada y portada para 7 arquitecturas, mostradas en la tabla 3.3.

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

Tabla 3.3: 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[8]

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.

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

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

Colaboradores

- 87 Ubuntu Core Developers[9]
- 159 Ubuntu Contributing Developers[10]
- 148 MOTU[11]
- 734 Ubuntu Members[12]

Política

- Codigo de conducta de Ubuntu https://www.ubuntu.com/about/about-ubuntu/conduct

Tiempos de soporte

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

Estable: Soporte por nueve meses.

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

Histórico de distribuciones

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

[9] https://launchpad.net/~ubuntu-core-dev [Consultado 14 de noviembre de 2016]
[10] https://launchpad.net/~ubuntu-developer-members [Consultado 14 de noviembre de 2016]
[11] https://launchpad.net/~motu [Consultado 14 de noviembre de 2016]
[12] https://launchpad.net/~ubuntumembers [Consultado 14 de noviembre de 2016]

Lanzamientos

La Tabla 3.4 muestra la fecha de lanzamiento de las diferentes versiones de la distribución Ubuntu.

Tabla 3.4: 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 [13]

Estadísticas

Componente *main*

Fecha del archivo Source: 02/04/2017 Número de paquetes descargados: 2.495 Cantidad de lenguajes reconocidos: 151

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

Tabla 3.5: SLOC agrupados por lenguaje (20 primeros)

Lenguaje	**Cantidad de archivos**	**Líneas en blanco**	**Líneas con comentarios**	**Líneas de código fuente**
Archivos de traducción	86.830	26.421.435	35.337.997	96.957.936
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	**1.141.804**	**65.688.235**	**80.327.422**	**337.256.079**

Figura 3.1: Porcentaje de archivos por lenguaje dentro del componente *main* de Ubuntu

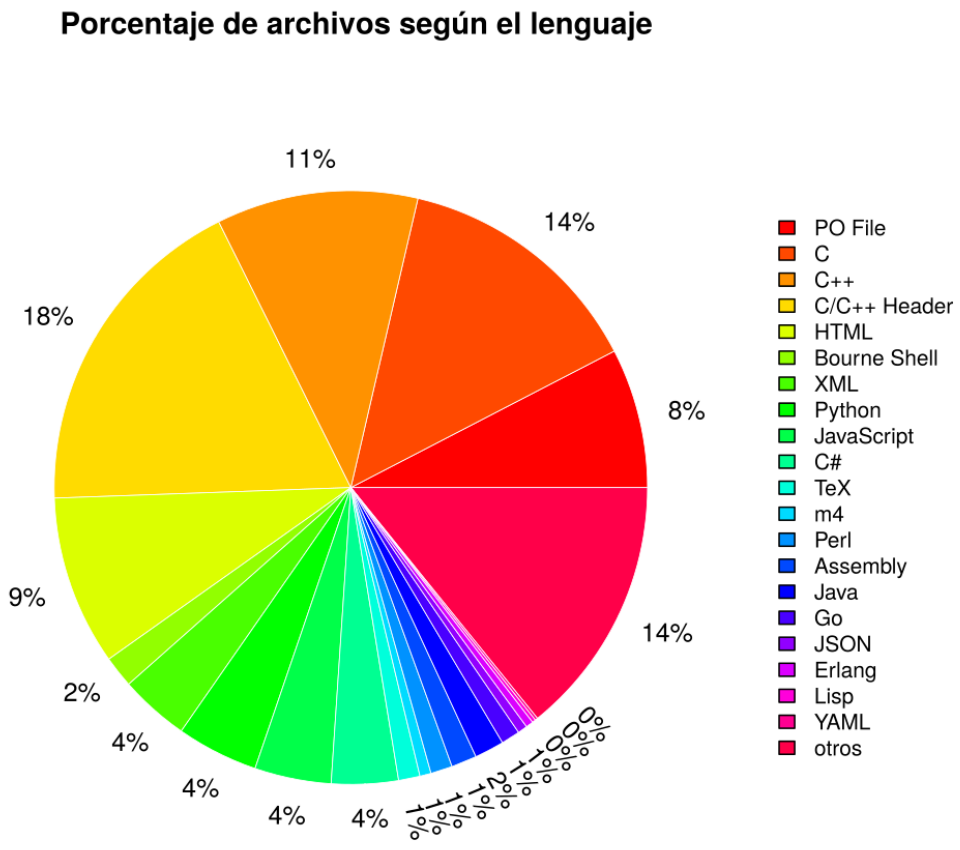
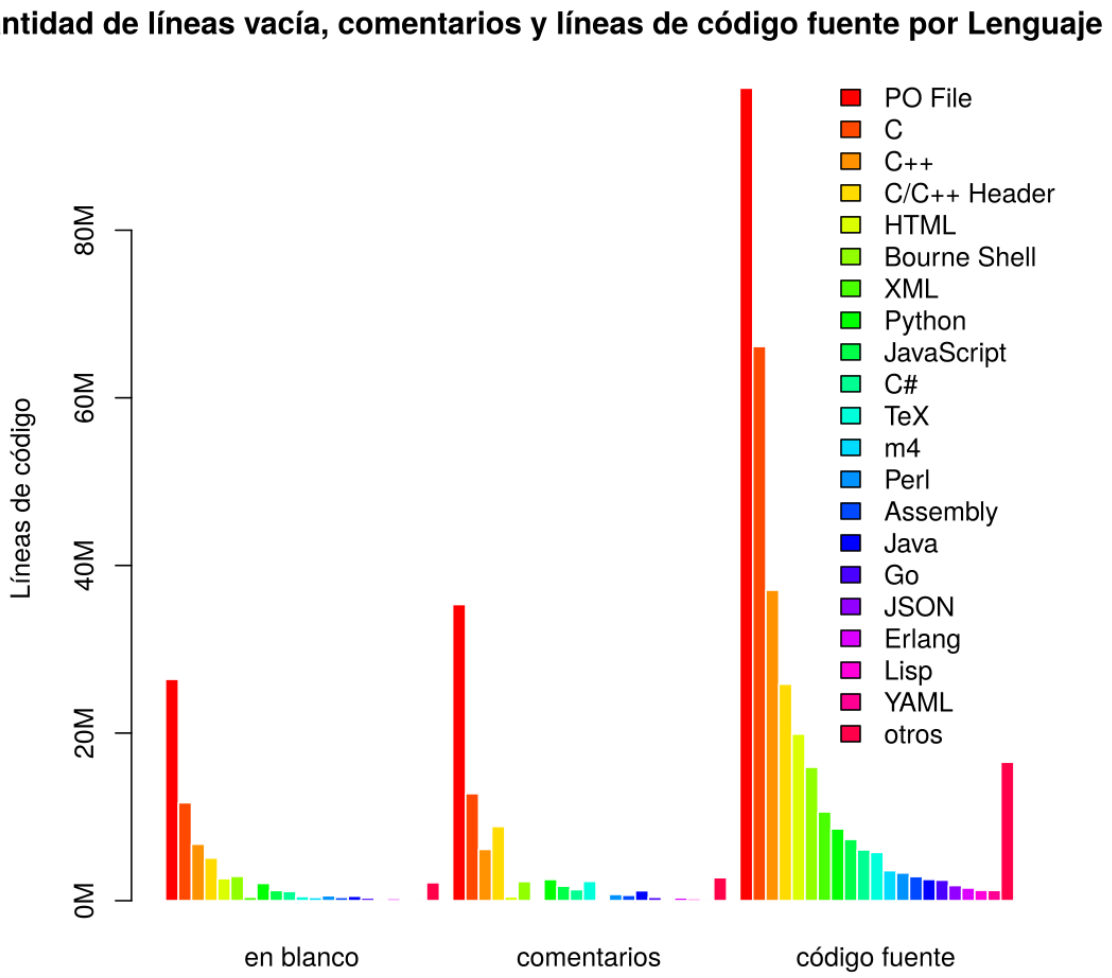


Figura 3.2: Cantidad de líneas vacía, comentarios y líneas de código fuente en los lenguajes del componente *main* de Ubuntu



COCOMO

Tabla 3.6: Estimaciones de esfuerzo y costos para el componente *main* de Ubuntu aplicando COCOMO Básico

Líneas de código fuente:	337.256.079
Esfuerzo estimado de desarrollo (persona–mes):	1.529.568,80
Productividad estimada:	0,22
Tiempo de desarrollo estimado (meses):	559,86
Personas requeridas estimadas (personas):	2.732,08
Costo total estimado del proyecto (US$):	201.728.298,39

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

Canaima GNU/Linux

Definición

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

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

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

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

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

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

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

El desarrollo de Canaima inicia con un estilo del tipo catedral, que poco a poco ha devenido en un estilo del tipo bazar, aunque todavía no lo alcance de manera cabal pues la posibilidad para la colaboración presenta aún algunas limitaciones.

Componentes

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

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

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

no-libres: Paquetes no compatibles con las DFSG.

Arquitecturas

Canaima construye su distro bajo 2 arquitecturas, mostradas en la tabla 3.7.

Tabla 3.7: Arquitecturas soportadas oficialmente por Canaima GNU/Linux

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

Fuentes: Repositorio Canaima[15]

Equipo Canaima GNU/Linux (septiembre de 2016)

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

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

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

Política

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

Histórico de distribuciones

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

Lanzamientos

La Tabla 3.8 muestra la fecha de lanzamiento de las diferentes versiones de la distribución Canaima GNU/Linux.

Tabla 3.8: Lanzamientos de versiones de Canaima

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

Fuentes: Canaima GNU/Linux[16] y Wikipedia [17]

[16] http://canaima.softwarelibre.gob.ve

[17] http://es.wikipedia.org/wiki/Canaima_(distribuci%C3%B3n_Linux)

Figura 3.3: Ciclo de desarrollo de Canaima GNU/Linux

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

Capítulo 4

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

Capítulo 5

Conclusiones y recomendaciones

Apéndices

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 código para automatización de descarga de paquetes fuentes se aprecia en el Código 1. 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.

```
1 #!/usr/bin/env python
2 # -*- coding: UTF-8 -*-
3
4 # Copyright 2016 David Hernández
5 #
6 #
7 # 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/>.

import string
import subprocess
import glob

try:
    f = open('Sources', 'r')
except IOError:
    print "Error al abrir el archivo ", arg
else:
    cont=0
    for line in f:
        if 'Binary: ' in line:
            binarios = string.split(line)[1:]
            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

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 carpeta como un proyecto diferente, se crea un directorio para cada uno de los paquetes, esto ayuda entre otras cosas a evitar sobreescritura de archivos en un directorio de descompresión común debido a nombres de archivos y directorios iguales, y organiza cada subproyecto (paquete de código) en un espacio propio.

```
#!/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/>.

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
```

```
34      if [ ${file: -4} == ".bz2" ]; then
35          bzip2 -dv *
36          tar xfi *
37          rm *.tar
38      fi
39      cd ..
40  done
```

Código 2: mkdirmv.sh

Cantidad de líneas de Código Fuente

Procedimiento

Antes de poder hacer uso de `cloc` se prepara antes un ambiente de trabajo para cada distribución a evaluar. 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 a estudiar a 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.

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

Se entra a la jaula

```
sudo chroot jaula
```

En caso de ser necesario se modifica el `sources.list` para usar solo 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

```
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[1]

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

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

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`[2] para los lenguajes C, java, pascal, php y xml que mejoran el desempeño en tiempo del programa.

`--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 a 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 (`_`)

Datos recolectados

Un ejemplo de los resultados arrojados por `cloc`[3] se muestran en la tabla 1.

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

[3] para los paquetes zygrib, zyn, zynaddsubfx y zypper

Tabla 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

Fórmulas

Tomando los valores bajo el modelo de estimación COCOMO Básico y aplicando los valores establecidos para el modo orgánico, se realizan los cálculos de esfuerzo y tiempo de desarrollo.

COCOMO puede ser aplicado a tres tipos de proyecto:

Orgánicos Proyectos bajo un ambiente familiar, estable, relativamente sin restricciones y con desarrolladores experimentados.

Semi-empotrados Proyecto mezcla de los tipos orgánicos y empotrados, puede haber o no desarrolladores expertos y requerimientos mas o menos rígidos.

Empotrados Proyectos ambiciosos con fuertes restricciones y experiencia moderada.

$$Epm = a \times (KSLOC)^b \tag{1}$$

$$Prod = \frac{KSLOC}{Epm} \tag{2}$$

$$Tdev = c \times (Epm)^d \tag{3}$$

$$Per = \frac{Epm}{Tdev} \tag{4}$$

$$Ctd = Per \times Spa \tag{5}$$

Donde:

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

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

$Prod$: es la productividad del proyecto.

$Tdev$: 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[4].

a, b, c y d: son los coeficientes según el tipo de proyecto, ver tabla 2

Tabla 2: Constantes para el cálculo de distintos aspectos de costes

Tipo de proyecto	a	b	c	d
Orgánico	2,40	1,05	2,50	0,38
Medio	3,03	1,12	2,50	0,35
Embebido	3,60	1,20	2,50	0,32

[4]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

Bibliografía

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.

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

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

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

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

Boehm, B., Clark, Horowitz, Brown, Reifer, Chulani, Madachy, R., y Steece, B. (2000). *Software Cost Estimation with Cocomo II with Cdrom*. 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.

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

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

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).

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)*.

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. Disponible en http://www.iso.org/iso/catalogue_detail?csnumber=43447.

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

Martínez, L. (2012). Re: [Discusion] Que Día Mes Y Año Nació Canaima GNU/Linux. Correo Electrónico. Disponible en: http://listas.canaima.softwarelibre.gob.

ve/pipermail/discusion/2012-August/007569.html. Consultado el 24 de marzo de 2017.

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

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

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. Disponible en https://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html. Consultado en noviembre 2016.