Centro Nacional de Desarrollo e Investigación en Tecnologías Libres (CENDITEL)
Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC), Mérida – Venezuela
ISSN: 2244-7423

Aplicando el Modelo Constructivo de Costos a Canaima GNU/Linux

David A. Hernández Aponte.[1]
Asociación Cooperativa Hoatzin de Base Tecnológica[1]
Mérida, Venezuela
dhernandez@hoatzin.org

Fecha de recepción: 10/09/2018
Fecha de aceptación: 05/10/2018
Pág: 73 – 83

Resumen

En este trabajo evalúa el Esfuerzo requerido para el desarrollo de las dos últimas versiones de Canaima GNU/Linux, Canaima 5.1 Chimantá y Canaima 6.0 Kavac, empleando el Modelo Constructivo de Costos. Mediante la utilización de la herramienta "cloc" de Al Danial, se determinó la cantidad de líneas de código fuente de los paquetes que componen el repositorio principal de cada una de las dos versiones de Canaima en estudio. El repositorio principal de Canaima 5.1 arrojó estar constituido por 264.255 líneas de código mientras que para Canaima 6.0 el número de líneas de código aumentó a 633.062. El Esfuerzo estimado de desarrollo para Chimantá fue de 778,33 persona–mes en tanto que para Kavac este valor se ubicó en 2.097,63 persona–mes. Finalmente, el Costo estimado para el desarrollo del Proyecto Canaima en su versión 5 se estimó en US$ 1.873.361,59 y para la versión 6 fue de US$ 3.463.948,65. Los análisis realizados reflejaron que aún cuando el número de paquetes en el repositorio nativo de Canaima disminuyó de una versión a la otra, los lenguajes utilizados, las líneas de código fuente y las estimaciones de costo del proyecto incrementaron sus valores.

Palabras clave: Modelo Constructivo de Costos, COCOMO, líneas de código fuente, SLOC, Canaima GNU/Linux, Software Libre.

Introducción

Canaima GNU/Linux es una distribución Linux basada en Debian. Esta distribución ha sido ideada, impulsada y respaldada por el Estado venezolano desde su nacimiento en el año 2007. Actualmente Canaima GNU/Linux se encuentra en su sexta versión y ha sido bautizada con el nombre Kavac.

Canaima trabaja con varios repositorios de software. Tiene un repositorio principal propio y también se alimenta de los repositorios característicos del proyecto Debian. La idea del repositorio propio es disponer de los paquetes de software desarrollados y/o adaptados por

el equipo y los colaboradores de Canaima para la versión correspondiente, sin embargo, se han venido incluyendo igualmente algunos paquetes externos no mantenidos por la comunidad Canaima.

Ya en ocasiones previas se ha hecho el ejercicio de medir sistemas de software utilizando la cantidad de líneas de código fuente (*Source Lines of Code –SLOC–* en inglés) como lo demuestran, entre otros, los trabajos de Wheeler (2001,2002)[14],[15] con Red Hat Linux en sus versiones 6.2 y 7.1; González et al. (2001, 2003)[7], [8] estudiando las distribuciones Red Hat Linux en sus versiones 5.2 a 8.0 y Debian en las versiones 2.0 a 3.0; Amor et al. (2004, 2005a,b, 2007)) [3],[1], [2], [4], [5] con sus trabajos sobre Debian en las versiones 3.0 a 5.0 y Gnome (algunas aplicaciones del proyecto); Herraiz et al. (2006) [11] quienes también analizan varias aplicaciones del proyecto Gnome; o Robles et al. (2005)[12] en su indagación sobre los núcleos Linux y *BSD.

Hernández (2018)[9] evalúa el Esfuerzo estimado para el desarrollo de Canaima Chimantá (5.0), Debian Jessie (8.0) y Ubuntu Xenial Xerus (16.04) aplicando el modelo Constructivo de Costos (COCOMO) (Boehm, 1981)[6]. Una de las tareas para lograr el objetivo consiste en el estudio de los paquetes incluidos en los repositorios principales de las tres distribuciones.

Entiéndase este artículo como una ampliación del trabajo antes mencionado al revaluar la versión 5 de Canaima, extender el estudio a la versión 6 de Canaima y comparar ambos resultados. De igual modo, por su condición de trabajo derivado, algunos conceptos ya tratados serán omitidos.

Cabe mencionar que las diferencias entre las versiones Canaima Chimantá 5.0 y 5.1 no afectan a las aplicaciones que habitan en el repositorio principal[1], por lo que para efectos de este trabajo es indistinto hablar de Canaima 5.0 o 5.1, se utiliza en algunas ocasiones la nomenclatura 5 o 5.X para referirse al estudio sobre esta serie de versiones.

Metodología

Código fuente

El procedimiento para el cálculo de estimaciones parte inicialmente con la descarga del código fuente de cada paquete a estudiar. Para ello, se ubican los repositorios donde se encuentran las fuentes y se procede a descargar los archivos respectivos apoyándose en el índice de paquetes que el mismo sistema de repositorios ofrece. En Debian y derivados, los índices que contienen la lista de archivos se encuentran con el nombre `Sources` para las fuentes de los programas y `Packages` para los paquetes binarios.

En Canaima 6.0 el método de adquisición del código a analizar varía con respecto al utilizado en la serie Canaima 5.0 y Canaima 5.1, donde los 35 paquetes de software del repositorio principal se encuentran acompañados de sus códigos fuente correspondientes, descargables desde

[1] https://canaima.softwarelibre.gob.ve/descargas/canaima-gnu-linux/repositorio-de-distribuciones/canaima-gnulinux-serie-5x-oficiales

el repositorio establecido para tal fin. Con Canaima Kavac esto no pasa, el repositorio de fuentes sólo dispone de dos paquetes de los 13 que integran el repositorio principal de la distribución.

Dicho esto, por la vía del repositorio de fuentes se descargan las dos aplicaciones disponibles, `minergatecli` y `petrowallet`. También se hace una búsqueda de las fuentes en el sistema de control de versiones utilizado por el proyecto Canaima[2], allí se logran ubicar las fuentes de los paquetes `canaima-certificados-firefox` y `canaima-nuevos-paquetes`. Otros paquetes se encuentran en versiones anteriores a las utilizadas en el último lanzamiento de la distribución y se descarta su descarga por ser versiones obsoletas que no corresponden a las publicadas en Canaima 6. A través del buscador de paquetes de Debian[3] se obtienen las fuentes de los paquetes `electrum`, `hello-world`, `python3-jsonrpclib-pelix` y `python3-pyaes`. Finalmente, el resto de paquetes se descargan del repositorio de binarios. No es lo ideal puesto que algunos programas de este repositorio incluyen archivos binarios a los que no se les podrá realizar el análisis de líneas de código fuente.

Se permite hacer el análisis sobre estos paquetes, en teoría binarios, porque el contenido de algunos ellos admite, por los lenguajes utilizados, analizarlos. Es el caso de archivos de configuración y los módulos escritos en lenguaje Python, por citar solamente dos ejemplos.

Conteo de SLOC

Para obtener el número de archivos, número de líneas físicas de código, número de líneas en blanco y número de líneas de comentarios se utiliza la herramienta de software libre `cloc`[4]. Se opta por esta aplicación principalmente por la cantidad de lenguajes que pueden ser identificados, 241 lenguajes en su versión 1.78. A mayor número de lenguajes identificables mayor será el número de líneas de código que se pueden considerar.

En el caso de los archivos provenientes del repositorio binario, una vez desempaquetados los archivos DEB, se procede a descomprimir el archivo `data.tar.gz`, este paquete contiene los archivos que componen el programa y es sobre este contenido que se efectúa el conteo.

Ya, con todos los directorios ordenados, se ejecuta el programa `cloc` para efectuar el análisis de los archivos.

Estimación de valores

Como ya se mencionó, el modelo COCOMO ha sido ampliamente usado en proyectos de Software Libre y Código Abierto para la estimación de Esfuerzo, Tiempo, Personal y Costos de desarrollo de un software. Uno de sus atractivos reside en la utilización de dos variables para generar estas estimaciones: el número de Líneas de Código Fuente y el Salario Promedio Anual.

Afirman Robles y González-Barahona (2004)[13] que COCOMO es un modelo pensado para los procesos clásicos de generación de software. Esto debe entenderse como los modelos

[2] http://gitlab.canaima.softwarelibre.gob.ve/
[3] http://packages.debian.org
[4] https://github.com/AlDanial/cloc

de desarrollo de software privativo. Sin embargo, es importante como punto de partida para comparaciones entre sistemas similares. Asimismo, genera ideas sobre el costo de desarrollo que tomarían estos proyectos de haberse adoptado un modelo privativo para su elaboración.

De este modo, aplicando el modelo COCOMO Básico y utilizando los valores establecidos en el modo orgánico de este modelo, se realizan los cálculos para estimar el *Esfuerzo* (ecuación 1), *Tiempo de desarrollo* (ecuación 2), *Personas requeridas para el desarrollo* (ecuación 3) y el *Costo total estimado* para el desarrollo del proyecto (ecuación 4).

Ecuaciones

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

$$Tdev = c \times (Epm)^d \tag{2}$$

$$Per = \frac{Epm}{Tdev} \tag{3}$$

$$Ctd = Per \times Spa \tag{4}$$

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[5].

a, b, c y d: son los coeficientes según el tipo de proyecto, ver tabla 1.

Tabla 1: 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

[5]Se toma como referencia el valor US$ 75.516,00 para el mes de septiembre de 2018, dato obtenido desde https://www.payscale.com/research/US/Job=Software_Engineer_/_Developer_/_Programmer/Salary

Resultados

Paquetes

El repositorio principal de Canaima Chimantá[6] contiene 35 paquetes de software que se muestran en la tabla 2 mientras que Canaima Kavac presenta 13 paquetes, referidos estos en la tabla 3.

Tabla 2: Paquetes incluidos en el repositorio principal de Canaima Chimantá

Nombre del paquete	Descripción corta
canaima-ayuda	Ayuda de Canaima GNU/Linux
canaima-base	Metapaquete que agrupa las aplicaciones básicas para Canaima
canaima-bienvenido-gnome	Aplicación de bienvenida para Canaima
canaima-cinnamon	Escritorio Cinnamon
canaima-cinnamon-edu	Configuración de limitación ideal para uso en aulas de clase
canaima-cinnamon-pp	Configuración escritorio Cinnamon del Poder Público
canaima-cinnamon-restringido	Configuración de limitación ideal para uso en aulas de clase
canaima-constructor	Script para completar configuración de live-build
canaima-control-parental	Control de contenido que pueda ser perjudicial para niños
canaima-controladores	Paquete de controladores para Canaima
canaima-desktop-base	Paquetes comunes al escritorio
canaima-espanol	Traducción al español
canaima-icon-theme	Iconos multientornos para Canaima
canaima-info	Marca Canaima
canaima-lanzadores	Configuración de nombres de aplicaciones
canaima-llaves	Llaves de los repositorios de Canaima
canaima-mate	Escritorio Mate para Canaima
canaima-mate-edu	Configuración de limitación ideal para uso en aulas de clase
canaima-mate-pp	Escritorio Mate del Poder Público
canaima-mate-restringido	Configuración de limitación ideal para uso en aulas de clase
canaima-mdm-themes	Tema MDM para Canaima
canaima-multimedia	Paquetes multimedia
canaima-multimedia-pp	Paquetes multimedia para Canaima Poder Público
canaima-navegador	Modificaciones del navegador
canaima-oficina	Paquetes ofimáticos esenciales
canaima-oficina-pp	Paquetes ofimáticos esenciales para Canaima Poder Público
canaima-servicios	Servicio de APIs
canaima-themes	Conjunto de temas Canaima
classmate-laptop-keys	Configuraciones para activar las teclas especiales de función en portátiles Classmate
kds	Proyecto Kit de servicios
live-installer	Instalador en modo vivo
mintinstall	Gestor de software
mintsources	Herramienta de configuración de fuentes de software
pinta	Programa básico de dibujo
sexy-python	Enlaces del lenguaje Python para la biblioteca libsexy

Se aprecia inmediatamente que en la versión 6 de Canaima ocurre una disminución de los paquetes incluidos en su repositorio principal con respecto a la versión anterior, pasan de 35 paquetes a 13. Esto se puede aducir por el enfoque dado a la nueva versión, tal como lo explica Muñoz (2018)[10] al esbozar algunas aspectos adoptados para la más reciente entrega de Canaima. El primer aspecto que destaca esta versión es que Kavac no presenta diferentes

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

orientaciones como otrora cuando existían las versiones "Poder Público" y "Poder Popular", el primero orientado a usuarios de las instituciones de la Administración Pública Nacional (APN), mientras que el segundo estaba dirigido al usuario general. También, se abandona la utilización de repositorios de Linux Mint Debian Edition y se retorna a un sistema derivado exclusivamente en Debian.

Tabla 3: Paquetes incluidos en el repositorio principal de Canaima Kavac

Nombre del paquete	Descripción corta
canaima-certificados-firefox	Paquete que agrega certificados venezolanos a firefox-esr
canaima-desktop-base	Paquetes comunes al escritorio
canaima-icon-theme	Iconos multientornos para Canaima
canaima-nuevos-paquetes	Mecanismo para integrar nuevos paquetes a sistemas en producción
electrum	Cliente bitcoin de fácil uso
hello-world	Paquete ficticio con un script hello world
minergate-cli	Minero basado en CryptoNote
minergatecli	Minero basado en CryptoNote adaptado por Canaima
nem-nis	Servidor de infraestructura NEM
petrowallet	Cartera ligera para la criptomoneda petro
python3-electrum	Módulo Python para electrum
python3-jsonrpclib-pelix	Implementación de la especificación JSON-RPC
python3-pyaes	Implementación Python del cifrado AES

Lenguajes

El lenguaje más utilizado en Canaima 5.X es C# con 341 archivos, seguido de Javascript con 211 archivos y Python con 155 archivos. Por su parte, Canaima 6.0 muestra su preferencia con archivos de cabecera C/C++, hallándose 600, muy cerca le sigue el lenguaje Python con 582 archivos y en tercer lugar C++ con 340 archivos. En las figuras 1 y 2 se expresa la cuota de lenguajes utilizados en las dos versiones de Canaima representadas en porcentaje.

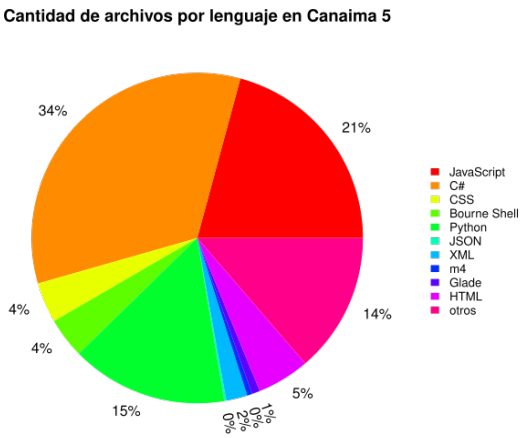


Figura 1: Porcentaje de archivos por lenguaje dentro del componente principal de Canaima Chimantá

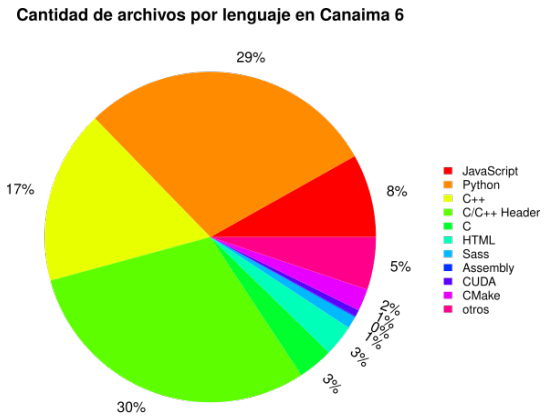


Figura 2: Porcentaje de archivos por lenguaje dentro del componente principal de Canaima Kavac

Líneas de código

Las tablas 2 y 3 exponen la cantidad de archivos y cantidad de código escrito para cada versión.

Javascript es el lenguaje predominante en ambas versiones, con una representación del 26,6 % del código fuente en la versión 5.1 y 43,6 % en la versión 6.0.

En las figuras 3a y 3b se resumen los resultados del análisis mostrando líneas en blanco, líneas de comentario y líneas de código por cada lenguaje, tanto en Canaima Chimantá como en Canaima Kavac.

Tabla 4: SLOC agrupados por lenguaje para Canaima Chimantá

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	6.154	13.570
JSON	2	0	0	11.560
XML	21	265	1.662	11.079
m4	5	1.020	118	8.781
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**	**50.934**	**246.255**

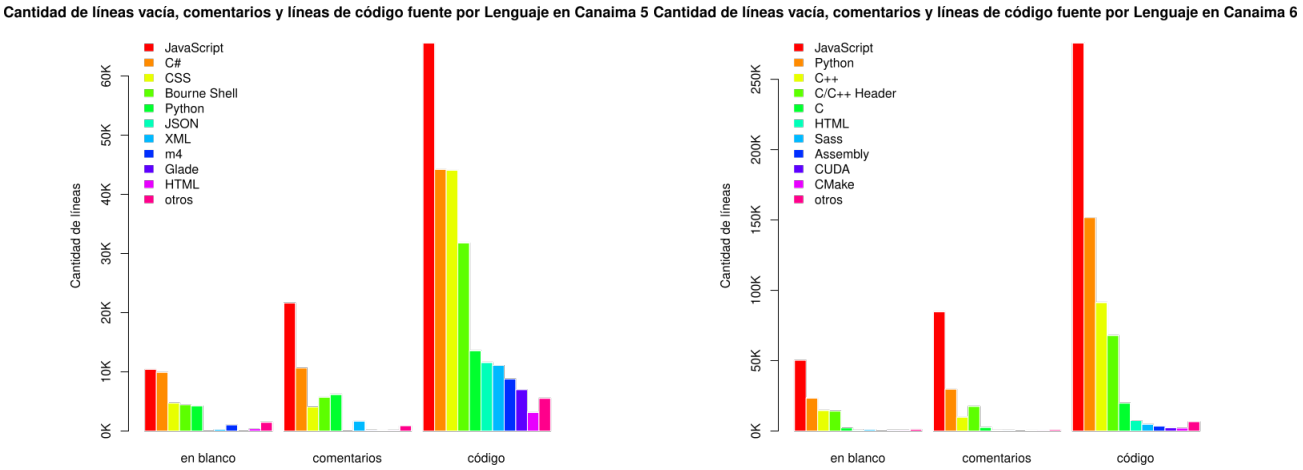
Número de paquetes analizados: 35
Cantidad de lenguajes reconocidos: 19

Tabla 5: SLOC agrupados por lenguaje para Canaima Kavac

Lenguaje	**Cantidad de archivos**	**Líneas en blanco**	**Líneas con comentarios**	**Líneas de código fuente**
JavaScript	163	**50.314**	**84.776**	**275.779**
Python	582	23.356	29.745	151.653
C++	340	14.556	9.801	91.252
Cabecera C/C++	**600**	14.129	17.471	67.921
C	69	2.231	2.558	19.770
HTML	61	360	288	7.390
Sass	24	957	314	4.782
Assembly	2	12	0	3.521
CUDA	13	477	95	2.311
CMake	43	470	618	2.074
OpenCL	2	326	313	1.442
CSS	18	267	75	1.263
JSON	5	0	0	903
make	9	143	65	486
MSBuild script	2	0	0	466
Perl	4	82	86	410
PHP	1	71	0	407
Markdown	6	157	0	320
Bourne Again Shell	21	64	282	278
Smarty	1	8	0	178
Bourne Shell	13	33	32	157
Java	1	5	4	88
Windows Resource File	1	19	23	62
Ruby	3	18	17	51
DOS Batch	9	6	1	40
Windows Module Definition	1	1	6	36
Protocol Buffers	1	4	0	13
XML	5	5	0	9
Total	**2.000**	**108.071**	**146.570**	**633.062**

Número de paquetes analizados: 13
Cantidad de lenguajes reconocidos: 28

Resalta que, a pesar de que Canaima Kavac posee menos paquetes en su repositorio comparado con Chimantá, 13 contra 35, la cantidad de archivos con código identificable por la herramienta de análisis es mayor en la última edición, arrojando a su vez mayor cantidad de líneas de código. Por ello, cabría pensar que el tamaño físico del conjunto de paquetes en Canaima 6 podría ser mayor, pero esto no pasa. Con el software desempaquetado, el tamaño total de los 13 directorios de Canaima 6 es de unos 169 MB, mientras que la suma del tamaño de los 35 proyectos de software de Canaima 5 es aproximadamente 381 MB. Esto se puede explicar



(a) Canaima Chimantá (b) Canaima Kavac

Figura 3: Cantidad de líneas vacías, comentarios y líneas de código fuente en los lenguajes del componente principal de Canaima.

sobre el hecho de que muchos de los archivos que componen los paquetes en Canaima 5 no son reconocidos como archivos contenedores de código dentro de los 241 lenguajes del programa de análisis, en esta categoría entran, por ejemplo, los archivos de texto. Adicionalmente, otra razón podría adjudicarse en la reutilización de código. Si por ejemplo, un script es utilizado y se localiza en varios proyectos, el código fuente de éste será contabilizado solamente la primera vez.

Modelo COCOMO

Las estimaciones obtenidas empleando COCOMO se muestran a continuación en la tabla 6.

Tabla 6: Estimaciones de esfuerzo y costos para los componentes principales de Canaima Chimantá y Kavac aplicando COCOMO Básico

Estimación	**Canaima 5**	**Canaima 6**
Líneas de código fuente:	246.255	633.062
Esfuerzo estimado de desarrollo (persona–mes):	778,33	2.097,63
Tiempo de desarrollo estimado (meses):	31,37	45,73
Personas requeridas estimadas (personas):	24,81	45,87
Costo total estimado del proyecto (US$):	1.873.361,59	3.463.948,65

Para el Costo Total estimado se toma el valor US$ 75.516,00 anual como salario de referencia para ingenieros de software, desarrolladores y programadores en septiembre de 2018.

Conclusiones

Canaima 6 provee menos paquetes que la versión 5 en su repositorio principal, pero por otro lado, estos paquetes ofrecen una mayor diversidad de lenguajes, archivos codificados y líneas de código fuente, llegando a duplicar y triplicar incluso sus valores previos. Se desea poder tener acceso a la totalidad de las fuentes del software distribuido en el proyecto Canaima para poder realizar un estudio más detallado de este comportamiento.

En este mismo sentido, es importante señalar también que un software que se considere Libre o Abierto, entre otras cosas, debe incluir el código fuente y posibilitar su distribución. Canaima Kavac está fallando en este aspecto al no incluir todas las fuentes en el repositorio, o al no mantener actualizadas públicamente las fuentes de los desarrollos en el sistema de control de versiones del proyecto.

Javascript se ubica como el lenguaje más popular en Canaima, según la cantidad de líneas de código fuente, al mantenerse en la punta durante las dos últimas versiones publicadas. Luego, Python está ganando terreno al situarse segundo en la última versión de Canaima, seguido de los lenguajes C/C++.

En cuanto a las estimaciones, partiendo del supuesto de que todos los paquetes fuesen codificados por el equipo de Canaima, el *Esfuerzo estimado de desarrollo* para Canaima 5.1 es de 778,33 persona–mes, lo que se traduce en que una persona necesitaría poco más de 778 meses (64.9 años) para codificar el software incluido en el repositorio principal de la distribución. Con Canaima 6.0 este período aumenta a 2.097,63 meses, o lo que es lo mismo, 174,8 años para programar el código dispuesto en el repositorio.

El tiempo estimado de desarrollo de Canaima 5.1 con un equipo de 25 personas tomaría alrededor de dos años y medio (31,37 meses) con un costo estimado de US$ 1.873.361,59. Por su lado, Canaima 6.0 tomaría algo más de 3 años y 9 meses (45,73 meses) con un equipo de 46 personas para completar su desarrollo, alcanzando un costo estimado de US$ 3.463.948,65.

Bibliografía

[1] 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.

[2] 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.

[3] 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).

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

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

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

[7] 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.

[8] González, J., 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).

[9] Hernández, D. (2018). *Ciclos de vida del software libre. Caso de estudio Distribución Canaima GNU/Linux* (Tesis de pregrado). Universidad de Los Andes, Venezuela.

[10] Muñoz, J. (2018). [Discusion] Sobre Canaima Popular. Recuperado de: http://listas.canaima.softwarelibre.gob.ve/pipermail/discusion/2018-April/012657.html

[11] 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)*.

[12] Robles, G., Amor, J. J., Gonzalez-Barahona, J. M. y Herraiz, I. (2005). *Evolution and Growth in Large Libre Software Projects*. In Proceedings of the Eighth International Workshop on Principles of Software Evolution (IWPSE '05). IEEE Computer Society, Washington, DC, USA, 165-174. DOI: https://doi.org/10.1109/IWPSE.2005.17

[13] Robles, G. y Gonzalez-Barahona, J. M. (2004). Toy Story: an analysis of the evolution of Debian GNU/Linux.

[14] Wheeler, D. (2000). Estimating Linux's Size. Recuperado de https://www.dwheeler.com/sloc/redhat62-v1/redhat62sloc.orig.html. Consultado en agosto 2018.

[15] Wheeler, D. (2001). More than a Gigabuck: Estimating GNU/Linux's Size. Recuperado de https://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html. Consultado en agosto 2018.