



Universidad
Rey Juan Carlos

INGENIERÍA EN SISTEMAS DE TELECOMUNICACIÓN

Curso Académico 2020/2021

Trabajo Fin de Grado

COMPARACIÓN DE ESTIMACIÓN DE COSTES
DE DESARROLLO ENTRE COCOMO Y
GIT2EFFORT

Autor : Víctor Miguel Torres Molina

Tutor : Dr. Gregorio Robles

*Dedicado a
mi familia y a Irene*

Agradecimientos

Familia y profesores, han sido indispensables durante todo este trayecto, pero sobre todo, mi más sincero y emotivo agradecimiento va dirigido a mi compañera de vida Irene. Gracias, de corazón, por impulsarme y apoyarme como lo haces cada día. Para toda la vida y después.

Resumen

Este proyecto consiste en una comparación y análisis de dos herramientas de estimación de costes de desarrollo software. En nuestro caso, hemos decidido centrarnos en COCOMO y Git2effort.

Para llevarlo a cabo, nos hemos apoyado fundamentalmente en estos dos modelos de estimación de esfuerzo. Ambos recursos se encargan de realizar una estimación del esfuerzo requerido para realizar un determinado proyecto software *a posteriori*, o sea, una vez ha sido realizado. Esto es especialmente interesante en proyectos de software libre, donde –a diferencia del desarrollo *in-house*– no es fácil conocer la participación que han tenido los desarrolladores en el proyecto, ya sea por la presencia de voluntarios, ya sea porque colaboran múltiples empresas en el proyecto.

El objetivo final de este trabajo es la implementación de un programa que nos permite cotejar dichos modelos, utilizando las diferentes métricas que nos proporciona, y de esta forma, poder calibrar las herramientas consiguiendo así unas estimaciones más precisas y realistas.

En el desarrollo de este programa, se han diferenciado varias fases de trabajo, en las cuales se han visto involucradas diferentes tecnologías, dependiendo de las necesidades en cada caso. La base principal sobre la que se sustenta la herramienta es el lenguaje de programación Python. En una primera fase, en la que obtenemos y procesamos toda la información se utilizaron las bibliotecas de Perceval y Pandas. En la siguiente fase, contamos con tecnologías relacionadas con el almacenamiento y la extracción de datos, empleando MySQL. Para realizar las estimaciones, obviamente, hemos utilizado las formulas y funciones procedentes de los modelos SLOCCount y Git2effort. Finalmente, a la hora de exportar resultados, nos hemos valido de dos formatos, HTML y PDF. En dichos ficheros se encuentra la información necesaria para comparar y estudiar la efectividad de dichos modelos.

Summary

This project consists of a comparison and analysis of two software development cost estimation tools. In our case, we decided to focus on SLOCCount and Git2effort.

To do this, we have relied mainly on these two cost estimation models. Both programs are responsible for forecasting the effort required to carry out a specific software project. This is really interesting on OSS projects where – unlike *in-house* development – it is not easy to know the participation that developers have had in the project, either due to the presence of volunteers or because multiple companies collaborate in the project.

The final objective of the project is the implementation of a program that allows us to compare these models, using the different metrics that it provides, and in this way, to be able to calibrate the tools, thus achieving more precise and realistic estimations.

In the development of this program, several work phases have been differentiated, in which different technologies have been involved, depending on the needs in each case. The main base of the tool is the Python programming language. In a first phase, in which we obtain and process all the information, the libraries we used were Perceval and Pandas. In the next phase, we have technologies related to data storage and extraction, we decided to use MySQL. To make the estimations, obviously, we have used the formulas and functions from the SLOCCount and Git2effort models. Finally, when exporting results, we have used two formats, HTML and PDF. In these files you will find the necessary information to compare and study the effectiveness of these models.

Índice general

1. Introducción	1
1.1. Contexto general	1
1.2. Motivación	2
1.3. Estructura de la memoria	2
2. Objetivos	5
2.1. Objetivo general	5
2.2. Objetivos específicos	5
2.3. Planificación temporal	6
3. Estado del arte	7
3.1. Modelos de estimación	7
3.1.1. COCOMO	7
3.1.2. Git2effort	9
3.2. Tecnologías	11
3.2.1. Python	11
3.2.2. MySQL	11
3.2.3. GitHub	12
3.2.4. SLOCCount	12
3.2.5. HTML	13
4. Diseño e implementación	15
4.1. Arquitectura general	15
4.2. Obtención de datos	15
4.3. Procesado de datos	17

4.4.	Almacenamiento de datos	18
4.5.	Extracción de datos	20
4.6.	Aplicación de los modelos de estimación	21
4.7.	Visualización de resultados	22
5.	Resultados	29
5.1.	Estimación inicial	29
5.1.1.	Ficheros en general	30
5.1.2.	Ficheros con código fuente	32
5.2.	Ajuste de Threshold en Git2effort	33
5.3.	Evolución del esfuerzo frente al threshold	41
5.4.	Saturación del threshold	44
5.5.	RoI	49
6.	Conclusiones	51
6.1.	Consecución de objetivos	51
6.2.	Aplicación de lo aprendido	52
6.3.	Lecciones aprendidas	52
6.4.	Trabajos futuros	53
A.	Manual de usuario	55
A.1.	Requisitos	55
A.2.	Conexión al servidor	56
A.3.	Ejecución	57
A.4.	Base de datos	58
A.5.	Resultados	60
	Bibliografía	61

Índice de figuras

2.1. Diagrama planificación temporal	6
4.1. Diagrama arquitectura general	15
4.2. Estructura diccionario de repositorios	16
4.3. Tablas almacenadas del repositorio Ceph	18
4.4. Datos Git2effort del repositorio Ceph	18
4.5. Datos SLOCCount del repositorio Ceph	19
4.6. Tablas SLOCCount por intervalo de tiempo del repositorio Ceph	20
4.7. Gráfica threshold frente al esfuerzo, modelo Git2effort, repositorio Ceph.	25
4.8. Gráfica saturación del threshold, modelo Git2effort	25
5.1. Análisis MediaWiki	36
5.2. Análisis MediaWiki	37
5.3. Tablas almacenadas repositorio Ceph	38
5.4. Tablas almacenadas repositorio Ceph	39
A.1. Sentencia SSH.	56
A.2. Sesión SSH iniciada.	56
A.3. Fichero MYSQL.	57
A.4. Fichero de repositorios.	57
A.5. Fichero de thresholds.	58
A.6. Ejecución del script.	58
A.7. Fin de la ejecución del script.	58
A.8. Inicio sesión MYSQL.	59
A.9. Selección de la BBDD.	59

A.10. Tablas de la BBDD.	59
A.11. Ficheros de salida.	60

Índice de cuadros

4.1. Datos SLOCCount.	16
4.2. Datos Git2effort.	17
4.3. Estructura del dataframe del modelo SLOCCount.	17
4.4. Estructura del dataframe del modelo Git2effort.	17
4.5. Tabla esfuerzo, modelos Git2effort y SLOCCount.	23
4.6. Tabla esfuerzo ficheros con código fuente, modelos Git2effort y SLOCCount.	23
4.7. Tabla del ajuste de threshold.	24
4.8. RoI, repositorio Mediawiki.	26
5.1. Threshold por repositorio.	30
5.2. Resultados de las estimaciones de los repositorios en el presente.	31
5.3. Resultados de las estimaciones de los repositorios en los tres primeros años.	31
5.4. Resultados de las estimaciones de los repositorios en los seis primeros años.	31
5.5. Resultados de las estimaciones de los repositorios en el presente de directorios con código fuente.	32
5.6. Resultados de las estimaciones de los repositorios en los tres primeros años de directorios con código fuente.	32
5.7. Resultados de las estimaciones de los repositorios en los seis primeros años de directorio con código fuente.	33
5.8. Evolución del esfuerzo, repositorio Ceph.	34
5.9. Evolución del esfuerzo, repositorio MediaWiki con ruido.	35
5.10. Evolución del esfuerzo, repositorio MediaWiki sin ruido.	40
5.11. RoI, repositorio Ceph	49
5.12. RoI, repositorio Mediawiki	49

5.13. RoI, repositorio Moodle 49

Capítulo 1

Introducción

Los modelos de estimación de costes de desarrollo software tratan de prever el esfuerzo requerido en la elaboración de un sistema software [4]. Son herramientas de un valor incalculable en la gestión de software. Tradicionalmente, la estimación de esfuerzo es utilizada en las primeras etapas de un proyecto software para determinar el número de desarrolladores y la cantidad de tiempo necesarios para llevar a cabo un proyecto software. Sin embargo, en el contexto de desarrollo de software libre (OSS, del inglés *Open Source Software*), este tipo de modelos apenas han proporcionado un seguimiento adecuado del esfuerzo.

El modelo de trabajo voluntario que presentan los proyectos OSS se ha visto influenciado por la incorporación de desarrolladores procedentes de empresas. Estas compañías, se ven interesadas en participar debido a sus estrategias de mercado, sin desplegar demasiados recursos. Como consecuencia, surge este modelo híbrido, que introduce una mayor complejidad en términos de estimación de costes.

1.1. Contexto general

Dada la naturaleza que presenta el desarrollo de proyectos OSS, se observa la carencia de dos puntos importantes en los modelos de estimación de costes. En primer lugar, la precisión de la medida del esfuerzo pasado de los desarrolladores. Los modelos de estimación tradicionales pueden prever el esfuerzo futuro gracias a información del pasado. Dado el aumento del número de empresas en el desarrollo de proyectos OSS, este aspecto es fundamental para permitir a las empresas evaluar dicho esfuerzo y sincronizar su contribución al proyecto adecuadamente. Por

último, el otro aspecto a destacar es la incorrecta categorización de los usuarios dentro del proyecto que provoca errores en la estimación [7].

1.2. Motivación

Debido a las dificultades que presenta la estimación en contextos de desarrollo de software libre, junto con la aportación que realizan las empresas en estos proyectos, generando nuevos modelos híbridos de contribución, se plantea si los modelos de estimación tradicionales se ajustan a estos cambios.

Por esta razón, la principal motivación de este proyecto es proporcionar una serie de métricas a la hora de predecir esfuerzos en el desarrollo de proyectos software, que nos permitan calibrar y configurar adecuadamente los nuevos modelos de estimación, que son más rigurosos y se adaptan mejor a estas necesidades.

1.3. Estructura de la memoria

En esta sección se muestra la estructura que sigue la memoria del proyecto con el fin de facilitar su comprensión. Está organizada en seis apartados, que se describen brevemente a continuación:

- **Introducción.** En este primer apartado se presenta y contextualiza la idea principal del proyecto. Se incluye la motivación que explica la finalidad del proyecto.
- **Objetivos.** Define el objetivo general y los objetivos específicos que lo componen. Se añade además la planificación seguida durante el progreso del trabajo.
- **Estado del arte.** Describe los modelos de estimación y todas las tecnologías implicadas en el desarrollo de la herramienta.
- **Diseño e implementación.** Esta sección contiene, en absoluto detalle, la distribución y la lógica que constituye el programa.
- **Resultados.** En este apartado se muestran y analizan los resultados obtenidos de todos los experimentos llevados a cabo durante la realización del proyecto.

- **Conclusiones.** Por último, se finaliza con las conclusiones y algunas ideas para posibles futuros estudios.

Capítulo 2

Objetivos

2.1. Objetivo general

El objetivo principal de este proyecto consiste en crear una herramienta de análisis y comparación de dos modelos de estimación de costes de desarrollo software, COCOMO y Git2effort, sobre repositorios de software libre alojados en la plataforma GitHub.

2.2. Objetivos específicos

En este apartado desglosamos las diferentes fases que han constituido el proyecto.

1. **Documentación.** Buscar y valorar información sobre el contexto de trabajo: Estimación de costes de desarrollo software.
2. **Programación del script.** Desarrollar un programa para exportar métricas que faciliten el estudio de un proyecto software.
3. **Experimentación.** Especificar y analizar los diferentes parámetros y recursos con el fin de ajustar y configurar los modelos de estimación.

2.3. Planificación temporal

En esta sección se muestra un diagrama de GANTT, que refleja el tiempo empleado en las diferentes fases del proyecto:

	2020		2021					
Actividad	Noviembre	Diciembre	Enero	Febrero	Marzo	Abril	Mayo	Junio
Documentación								
Especificación métricas								
Desarrollo								
Memoria								

Figura 2.1: Diagrama planificación temporal

Como puede observarse en la figura, en ocasiones se han solapado las fases de trabajo. Esto se debe a que a medida que se iban realizando avances en el proyecto, durante las validaciones, se planteaban nuevas métricas a incluir en el *script*, se iba documentando en la memoria o se realizaban otras tareas en paralelo.

Capítulo 3

Estado del arte

A continuación se realiza una breve descripción de los modelos y las diferentes tecnologías empleadas en el proyecto.

3.1. Modelos de estimación

3.1.1. COCOMO

COCOMO, *Constructive Cost Model*, es considerado uno de los modelos de estimación de costes de desarrollo software más utilizados y documentados. Fue desarrollado por Barry Boehm a finales de los 70 y comienzos de los 80, y lo expuso en su libro “Software Engineering Economics” [1] en el año 1981. Existen tres versiones del modelo:

- **Basic COCOMO.** Se trata de la versión más simple, ya que el cálculo del esfuerzo y coste lo lleva a cabo basándose en el tamaño del proyecto. El tamaño del programa viene determinado por las líneas de código que lo constituyen, expresadas como KSLOC (Kilo Source Line of Code).

Dependiendo del proyecto sobre el que queramos realizar la estimación, COCOMO distingue tres clases diferentes:

- **Organic.** Equipos de desarrollo software en entornos muy cercanos. La evolución del trabajo es normalmente estable, no prima la innovación en algoritmos y no tiene restricciones relevantes.

- **Semidetached.** Constituye un modo intermedio entre el Organic y el Embedded.
 - **Embedded.** En este caso, el proyecto está limitado por unas restricciones considerables, que pueden estar relacionadas con el procesador y el hardware.
- **Intermediate COCOMO.** A diferencia de Basic COCOMO este tiene en cuenta, además del tamaño del proyecto, unos coeficientes conocidos como cost drivers. Dichos factores incluyen la evaluación subjetiva del proyecto, hardware empleado y otros atributos. Se diferencian los siguientes:

1. Atributos del producto

- RELY: garantía de funcionamiento requerida al software.
- DATA: tamaño de la base de datos.
- CPLX: complejidad del producto.

2. Atributos del ordenador

- TIME: restricción de tiempo de ejecución.
- STOR: restricción del almacenamiento principal.
- VIRT: volatilidad de la máquina virtual.
- TURN: tiempo de respuesta del ordenador.

3. Atributos del personal

- ACAP: capacidad del analista.
- AEXP: experiencia en la aplicación.
- PCAP: capacidad del programador.
- VEXP: experiencia en máquina virtual.
- LEXP: experiencia en lenguaje de programación.

4. Atributos del proyecto

- MODP: prácticas de programación modernas.
- TOOL: utilización de herramientas software.
- SCED: plan de desarrollo requerido.

- **Detailed COCOMO.** Esta versión incorpora todas las características de la versión Intermediate, con un par de mejoras:
 - **Multiplicadores de esfuerzo sensitivos a la fase.** Los atributos no influyen en todas las fases de desarrollo de la misma forma. Se proporcionan multiplicadores de esfuerzo para cada atributo.
 - **Jerarquía del producto a tres niveles.** Se definen tres niveles de producto: módulo, subsistema y sistema. La cuantificación se realiza al nivel apropiado, es decir, al nivel al que es más susceptible la variación.

3.1.2. Git2effort

Git2effort implementa el modelo presentado en [7] y que se describe a continuación de manera resumida.

En el entorno de proyectos software de código abierto (FOSS) es difícil conocer el esfuerzo invertido, incluso una vez han finalizado. Estos datos están cobrando cada vez más importancia debido, entre otros, al crecimiento en el número de empresas que basan su estrategia de mercado en el desarrollo de proyectos de software libre. Git2effort se encarga de proporcionarnos dicha estimación, a partir de los datos de gestión de los repositorios alojados en GitHub. Para entender el modelo, previamente, debemos conocer los siguientes conceptos:

- **Commit.** Acción por la que un desarrollador atribuye una serie de modificaciones sobre un proyecto en un sistema de control de versiones.
- **Author.** Individuo que interviene en los cambios realizados sobre el desarrollo software. Puede no ser la persona que realice el commit.
- **Committer.** Desarrollador que realiza el commit sobre el repositorio, pero puede no ser el autor real de los cambios.
- **Full-Time developer.** Programador que dedica al menos 40 horas-semana al proyecto.
- **Person-Month (PM).** Magnitud del esfuerzo. Una persona al mes, se refiere a la dedicación de un Full-Time developer durante un mes sobre el proyecto.

- **Period of study n .** Intervalo de tiempo en el que se desarrolla la actividad del proyecto.
- **Threshold.** Umbral de actividad (en commits) durante un periodo de estudio que un desarrollador debe alcanzar para considerarlo Full-Time developer.
- **Esfuerzo E_d en un periodo de estudio (para un desarrollador d).** Cualquier desarrollador que supere el *threshold* t en un periodo de estudio de n meses, será considerado Full-time developer con un esfuerzo de n PMs. Un desarrollador, con a commits que no supere el umbral t , obtendrá un esfuerzo de $n * \frac{a}{t}$ PMs.
- **Esfuerzo E_n en un periodo de estudio (para el proyecto completo).** El esfuerzo total E_n se obtiene sumando todas las aportaciones de esfuerzo de todos los desarrolladores (Full-Time y Non-Full-Time). La formula que se emplea es la siguiente:

$$E_n = \sum_{d=1}^{d_{ft}} n + \sum_{d=1}^{d_{n.ft}} n * \frac{a_d}{t} \quad (3.1)$$

Los metadatos de los repositorios, contienen información relacionada a la actividad realizada sobre el proyecto. El modelo de estimación, se encarga de traducir dicha actividad en esfuerzo medido en personas/mes. Dicha conversión solo se realiza para determinados instantes en el tiempo, momentos, en los que se realiza el commit y este es almacenado. El tiempo dedicado para cada actualización del proyecto no está disponible. Se sabe que las contribuciones de proyectos FOSS no son uniformes, es decir, no todos los desarrolladores se implican de la misma forma. Hay desarrolladores que aportan más que otros. Esta característica dentro de los proyectos FOSS se convierte en uno de los principales problemas en la tarea de estimar esfuerzo. En este caso, Git2effort diferencia entre desarrolladores que dedican toda su jornada laboral al proyecto en cuestión (Full-time developers) y aquellos que se dedican de forma parcial (Non-full-time developers). Por lo tanto, la primera tarea que desempeña Git2effort se trata de identificar y catalogar a los desarrolladores. Para poder determinar el tipo de desarrollador, la herramienta evalúa dos parámetros:

- Número de commits realizado por un usuario durante un periodo de tiempo.
- Número de días en los que el usuario ha permanecido activo frente al proyecto (active days), es decir, aquellos días en los que ha realizado al menos un commit durante un periodo de tiempo.

Lo que se pretende con ambos parámetros es encontrar un valor umbral para el número de commits (o de active days) con el que poder identificar a los full-time developers con un mínimo error.

3.2. Tecnologías

3.2.1. Python

Python¹ es un lenguaje de alto nivel con semántica dinámica integrada, destinado principalmente al desarrollo web y de aplicaciones. Es comúnmente utilizado en diversos sectores tales como Internet scripting, administración de sistemas, data mining, desarrollo de videojuegos y garantía de calidad, entre otros. Se trata de uno de los lenguajes más utilizados en el mundo. El código del *script* se escribió en la versión 3.4 de Python. Estas son algunas de sus principales características:

- Lenguaje interpretado. A diferencia del compilado, es convertido a lenguaje máquina a medida que es ejecutado.
- Python es un lenguaje orientado a objetos.
- Multiplataforma. Puede ejecutarse en diferentes sistemas operativos como Windows, Linux, Macintosh, etc.
- Curva rápida de aprendizaje debido a su legibilidad.
- Gran cantidad de bibliotecas, tipos de datos y funciones incorporadas.
- Open source.

3.2.2. MySQL

MySQL² es uno de los sistemas de gestión de bases de datos relacionales más populares, basado en queries (SQL, Structured Query Language). Nos ayudamos de este software para

¹<https://python.org>

²<https://mysql.com>

almacenar los datos necesarios para estimar el esfuerzo correspondiente. A continuación se mencionan algunas de sus características fundamentales:

- Open source.
- Modelo cliente-servidor
- Es rápido, seguro, escalable y fácil de utilizar.
- Multiplataforma.

3.2.3. GitHub

GitHub³ es una plataforma de gestión de proyectos abiertos y control de versiones de código. Permite la colaboración con desarrolladores de todo el mundo, la planificación y seguimiento de proyectos. Se trata de uno de los espacios colaborativos más grandes en todo el mundo. Es la fuente de datos del modelo Git2effort.

3.2.4. SLOCCount

SLOCCount⁴ es un conjunto de programas encargados de la contabilización de líneas de código fuente (SLOC) procedentes de proyectos software de grandes dimensiones. Se trata de una herramienta de métricas de software. SLOCCount realiza la estimación del tiempo de desarrollo y esfuerzo a partir de las líneas de código utilizando el modelo de estimación COCOMO. Es uno de los modelos empleados en este proyecto. Algunas de sus principales características son:

- Open source.
- Manejo de un amplio abanico de lenguajes de programación como Ada, C, C++, COBOL, PHP,..etc
- Fácil utilización e interpretación de resultados

³<https://github.com>

⁴<https://dwheeler.com/sloccount/sloccount.html>

3.2.5. HTML

HTML, HyperText Markup Language, es un lenguaje destinado a la elaboración de páginas web. El principal pilar sobre el que se sostiene es la referenciación de recursos. Se considera el lenguaje web más importante. Se caracteriza por:

- Adaptabilidad.
- Fácil interpretación.
- Compatibilidad con cualquier versión.
- Todos los navegadores lo utilizan.

En nuestro caso, lo utilizamos como ventana para visualizar las conclusiones del análisis.

Capítulo 4

Diseño e implementación

En este capítulo se realiza una descripción detallada de la estructura y funcionalidad del programa *compara_estimadores*.

4.1. Arquitectura general

La arquitectura general del proyecto sigue una estructura modular. Se definieron todos los procesos que se debían llevar a cabo, y se programaron funciones independientes para cada uno:

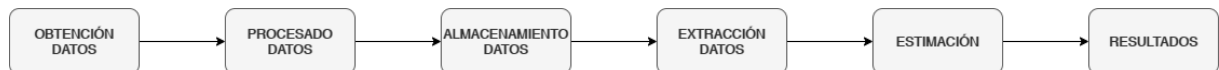


Figura 4.1: Diagrama arquitectura general

De acuerdo a las fases que se presentan en el diagrama, desglosamos el *script* de la misma forma. A continuación se describen cada una de ellas.

4.2. Obtención de datos

En esta fase, se consiguen los datos correspondientes a los modelos Git2effort y SLOC-Count, de los que algunos de ellos se emplean como parámetros a la hora de realizar las estimaciones. En cada caso se utiliza una técnica de extracción diferente, ya que cada modelo basa su cálculo sobre diferentes parámetros. Las funciones correspondientes a esta fase del *script* serían:

- `read_reps(list)`. Esta función lee la lista de repositorios. Toma el fichero de texto que contiene la URL de cada repositorio, y devuelve un array de URL's listas para procesar.
- `rep_dict(rep_list,ts_list)`. Esta función ofrece un diccionario de repositorios, en el cual podemos encontrar el nombre del repositorio, como key, y su correspondiente *threshold* y URL, como valor. Toma la salida de la función `read_reps`, y además un fichero de los valores umbrales de Git2effort de cada repositorio. La estructura que sigue el diccionario es la siguiente:

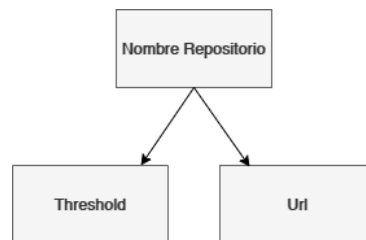


Figura 4.2: Estructura diccionario de repositorios

- `sloc_data(rep)`. Esta función obtiene los datos correspondientes al modelo SLOCCCount. Toma el nombre del repositorio y devuelve la salida de la ejecución del programa SLOCCCount. Los datos que se recogen son los siguientes:

File_Name	Language	Code.Lines
-----------	----------	------------

Cuadro 4.1: Datos SLOCCCount.

Se basa en la ejecución del programa SLOCCCount a través de la función `subprocess.run()` de la biblioteca de Python `subprocess` y el parseo de su salida.

- La extracción de datos del modelo Git2effort corresponde con unas líneas de código prestadas del propio programa Git2effort. Dichas líneas se incluyeron directamente en el procesado de datos de dicho modelo, por comodidad. En cualquier caso, los datos que establecimos relevantes dentro del estudio son:

Author	Committer	File_name	Date
--------	-----------	-----------	------

Cuadro 4.2: Datos Git2effort.

4.3. Procesado de datos

En este apartado se organiza la información para un adecuado almacenamiento en la base de datos. Las funciones que se emplean en este caso son:

- `sloc_dataframe(values)`. Con la ayuda de la biblioteca `pandas` esta función genera un dataframe, que contiene la información necesaria para la estimación y comprensión de cada repositorio. Toma como parámetro la salida de `sloc_data` y devuelve un dataframe que tiene la siguiente estructura:

Directory	Parent_Directory	File_name	Language	Code_Lines
-----------	------------------	-----------	----------	------------

Cuadro 4.3: Estructura del dataframe del modelo SLOCCount.

Los campos `Directory` y `Parent_Directory` se añadieron después. Esta información nos permite establecer un almacenamiento jerárquico dentro de la base de datos. El modelo empleado es el conocido como Adjency List. Es uno de los más utilizados, y permite que cada registro conozca inmediatamente su procedencia..

- `git2_dataframe(rep_url)`. Esta función tiene el mismo propósito que la que describíamos anteriormente, `sloc_dataframe`. El sistema de almacenamiento jerárquico es el mismo, solo varían algunos campos de acuerdo a los cálculos requeridos en este modelo. El dataframe correspondiente a `Git2effort` contiene la siguiente información:

Directory	Parent_Directory	Author	Committer	File_name	Date
-----------	------------------	--------	-----------	-----------	------

Cuadro 4.4: Estructura del dataframe del modelo Git2effort.

Como comentamos en el apartado de “Obtención de datos”, esta función contiene además unas líneas de código procedentes del propio script del modelo, con sus respectivas modificaciones acordes a la función. Se corresponden con la biblioteca de `Perceval`, que

nos permite la extracción de metadatos alojados en el histórico de commits de cada repositorio [2].

4.4. Almacenamiento de datos

En esta fase del *script*, una vez extraídos y procesados los datos, estos se almacenan en los servidores de la Universidad Rey Juan Carlos. Son servidores que cuentan con el sistema de almacenamiento MariaDB. En este caso, no ha sido necesario escribir un proceso en concreto como tal, ya que `pandas` proporciona una función destinada al almacenamiento de un dataframe. Con una única línea de código se realiza el almacenamiento:

```
dataframe.to_sql(name, con)
```

La conexión con la base de datos la establecemos gracias a la biblioteca `sqlalchemy`, utilizando la función `create_engine`. El parámetro que hay que pasarle es el que sigue:

```
dialect+driver://username:password@host:port/database
```

El formato de almacenamiento de los datos en MySQL sigue la forma de tabla. De acuerdo a los campos establecidos en los dataframes, se irán rellenando según el volumen del repositorio. El número de tablas por repositorio, en un principio, es de mínimo dos, una por cada modelo de estimación:

Tabla	Acción	Filas	Tipo	Cotejamiento	Tamaño
ceph_git2effort	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	~556,166	InnoDB	utf8mb4_general_ci	89.2 MB
ceph_sloccount	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	4,842	InnoDB	utf8mb4_general_ci	560.8 KB

Figura 4.3: Tablas almacenadas del repositorio Ceph

index	Directory	Parent_Directory	Author	Committer	File_Name	Date
0	top_dir	NULL	Sage Weil <sage@inktank.com>	Sage Weil <sage@inktank.com>	dcache.cc	1088119686
1	include	top_dir	Sage Weil <sage@inktank.com>	Sage Weil <sage@inktank.com>	MDS.h	1088119686
2	include	top_dir	Sage Weil <sage@inktank.com>	Sage Weil <sage@inktank.com>	dcache.h	1088119686
3	test	top_dir	Sage Weil <sage@inktank.com>	Sage Weil <sage@inktank.com>	fakemds.cc	1088119686
4	top_dir	NULL	Sage Weil <sage@inktank.com>	Sage Weil <sage@inktank.com>	dcache.cc	1088189798
5	include	top_dir	Sage Weil <sage@inktank.com>	Sage Weil <sage@inktank.com>	dcache.h	1088189798

Figura 4.4: Datos Git2effort del repositorio Ceph

index	Directory	Parent_Directory	File_Name	Language	Code_Lines
0	admin	top_dir	serve-doc	python	23
1	admin	top_dir	build-doc	sh	92
2	bin	top_dir	git-archive-all.sh	sh	203
3	ceph-menv	top_dir	build_links.sh	sh	10
4	ceph-menv	top_dir	mdo.sh	sh	11
5	ceph-menv	top_dir	mset.sh	sh	19

Figura 4.5: Datos SLOCCount del repositorio Ceph

Sí que es cierto, que debido al análisis que fuimos planteando a lo largo del desarrollo del proyecto, el número de tablas por repositorio varió. En nuestro caso, se añadieron tablas que contenían los mismo campos, pero correspondían a intervalos de tiempo concretos:

<input type="checkbox"/>	ceph_sloccount_before_2005	★	Examinar	Estructura	Buscar	Insertar	Vaciar	Eliminar	112	InnoDB	utf8mb4_general_ci	32.0 KB	.
<input type="checkbox"/>	ceph_sloccount_before_2006	★	Examinar	Estructura	Buscar	Insertar	Vaciar	Eliminar	267	InnoDB	utf8mb4_general_ci	64.0 KB	.
<input type="checkbox"/>	ceph_sloccount_before_2007	★	Examinar	Estructura	Buscar	Insertar	Vaciar	Eliminar	365	InnoDB	utf8mb4_general_ci	80.0 KB	.
<input type="checkbox"/>	ceph_sloccount_before_2008	★	Examinar	Estructura	Buscar	Insertar	Vaciar	Eliminar	799	InnoDB	utf8mb4_general_ci	144.0 KB	.

Figura 4.6: Tablas SLOCCount por intervalo de tiempo del repositorio Ceph

Este formato de tablas que guardan datos de una versión del repositorio en el pasado, están relacionadas únicamente a metadatos del programa SLOCCount. En git2effort, disponemos del campo Date como dimensión del tiempo. Para poder consultar los metadatos de versiones anteriores de los repositorios diseñamos la siguiente función:

- `repository_downgrade(rep_name, year)`. Toma el nombre del repositorio y el año que queremos consultar. Para ello, empleamos el comando `git checkout` de la biblioteca Git. Una vez obtenida la versión que buscamos del repositorio, volvemos a repetir el proceso de obtención de metadatos, procesado y almacenamiento en base de datos.

4.5. Extracción de datos

Para poder aplicar los modelos de Git2effort y SLOCCount a partir de la información recopilada en la base de datos, necesitamos extraerla. Para ello, nos valemos de *queries*. Estas solicitudes nos permiten manejar la información guardada en los servidores. Podemos efectuar multitud de operaciones sobre las tablas, pero nosotros nos centraremos en la función `SELECT`. Todas las funciones definidas en este apartado tienen el mismo objetivo: obtener los datos necesarios para la estimación de costes. En la query `SELECT` se han aplicado filtros, de acuerdo a los análisis efectuados y dependiendo de qué información se busca en cada caso. Las queries empleadas, en nuestro caso, tienen la siguiente forma:

```
SELECT author,timestamp FROM repository_name_git2effort
SELECT code_lines FROM repository_name_sloccount
```

Con este ejemplo obtendríamos todos los datos para poder realizar las estimaciones con cada modelo.

Las *queries* que se programaron en la herramienta son las siguientes:

- `git2_fetch(rep_name, timestamp)` y `sloc_fetch(rep_name, year)`. Estas funciones solicitan todos los datos almacenados de ambos modelos. Dependiendo del valor de los parámetros `timestamp` y `year` buscará información más o menos reciente. En el caso de `Git2effort` nos proporciona el nombre de los autores de los commits y sus fechas correspondientes. `SLOCCount` nos ofrece como salida las líneas de código totales.
- `git2_sc_query(rep_name, timestamp)` y `sloc_sc_query(rep_name, year)`. Estas funciones se encargan de extraer datos por cada modelo, filtrando por aquellos ficheros que contengan código fuente. Para ello, nos hemos valido del campo `File_Name`, ya que su formato nos proporciona información sobre el fichero. La salida es la misma que en las funciones anteriores.

4.6. Aplicación de los modelos de estimación

Las funciones desarrolladas en este apartado se encargan de realizar la estimación del proyecto software de acuerdo a la información recopilada del mismo:

- `sloc_estimation(code_lines)`. Toma como parámetro las líneas de código y devuelve como resultado la estimación del esfuerzo en personas-mes. El cálculo se basa en la siguiente fórmula:

$$persons_month = round((2,4 * (code_lines/1000) * *1,05)), 2) \quad (4.1)$$

- `git2_estimation(authordict, threshold)`. Toma como parámetro un diccionario de Python, que contiene los autores y fechas de cada commit realizado en el repositorio, y el *threshold*, que es valor umbral de commits. En este caso, a su vez, se usan algunas funciones empleadas en `Git2effort` dentro de `git2_estimation` como módulo. Al igual que `slocc_estimation`, devuelve el esfuerzo en personas-mes. Dentro de esta función podemos encontrar:

- `author_counting(authordict, period_length, active_days)`. Dado el diccionario que especificamos antes, un periodo de tiempo expresado en meses, devuelve una lista con el número de commits en cada periodo.
- `project_period_effort(author_count, threshold, period_length)`. Toma como parámetros la salida de la función `author_counting`, el umbral, y el periodo de tiempo, devolviendo:
 - Diccionario con el valor total del esfuerzo por cada periodo.
 - Diccionario con el número de desarrolladores full time por cada periodo.
 - Diccionario con el número de desarrolladores non-full time por cada periodo.
- `git2_authordict(authors, timestamps, roi_flag)`. Para poder realizar la estimación con el modelo `Git2effort`, como hemos observado, se necesita el parámetro `authordict`. Para ello, utilizamos esta función, que toma los autores y fechas de los commits, junto con el `roi_flag`, que nos permite evitar que se fusionen aquellas cuentas de usuario que contengan el mismo nombre. De esta forma, podemos disgregar entre las diferentes empresas o no, dependiendo de cada caso. La salida se ofrece como un diccionario de autores, con las respectivas fechas en las que han realizado commits en el repositorio de GitHub.

4.7. Visualización de resultados

Finalmente, en esta última fase se diseñaron una serie de funciones que presentan los resultados de esfuerzo obtenidos para cada modelo. En cada caso se utilizó una forma concreta de representar esos datos, tanto tablas como diagramas de barras. En este apartado podemos encontrar las siguientes funciones:

- `effort(rep_name, threshold)`. Esta función presenta el esfuerzo estimado para un repositorio de GitHub dado por ambos modelos, `Git2effort` y `SLOCCount`. Como puede observarse en la siguiente tabla, la estimación se realiza con todo el histórico de datos, para los tres primeros años y para los seis primeros años de desarrollo:

Modelo	Esfuerzo [personas-mes]
Git2effort	4941
SLOCCCount	2126
Git2effort [primeros 3 años]	283
SLOCCCount [primeros 3 años]	177
Git2effort [primeros 6 años]	883
SLOCCCount [primeros 6 años]	379

Cuadro 4.5: Tabla esfuerzo, modelos Git2effort y SLOCCCount.

- `source_code_effort(rep_name, threshold)`. El objetivo de esta función es similar a la anterior, a excepción de los datos con los que trabaja. En este caso, solo se tienen en cuenta aquellos ficheros con código fuente. Los intervalos de tiempo se mantienen:

Modelo	Esfuerzo [personas-mes](código fuente)
Git2effort	4462
SLOCCCount	2072
Git2effort [primeros 3 años]	276
SLOCCCount [primeros 3 años]	166
Git2effort [primeros 6 años]	864
SLOCCCount [primeros 6 años]	363

Cuadro 4.6: Tabla esfuerzo ficheros con código fuente, modelos Git2effort y SLOCCCount.

- `threshold_setting(rep_name)`. Esta función encuentra un valor para el *threshold* de Git2effort, de tal forma que el esfuerzo calculado para dicho modelo sea lo más aproximado al esfuerzo estimado por SLOCCount:

Año	Líneas de código	COCOMO	Threshold Git2effort	Commits	Git2effort
2003	22.840	64	21	654	62
2004	45.482	132	53	4.827	129
2005	60.084	176	77	9.034	175
2006	82.449	246	85	12.987	244
2007	100.796	304	111	17.578	303
2008	124.145	379	135	24.278	377
2009	147.749	455	147	29.585	451
2010	191.333	597	137	35.287	591
2011	241.771	763	131	43.107	755
2012	276.365	878	136	50.154	873
2013	298.648	953	145	56.433	947
2014	316.863	1.014	157	64.109	1.007
2015	353.385	1.137	156	71.300	1.129
2016	414.716	1.345	144	78.294	1.332
2017	465.295	1.518	137	84.265	1.507
2018	526.441	1.728	129	90.641	1.716
2019	553.939	1.823	132	98.096	1.811
2020	639.755	2.120	120	105.457	2.101

Cuadro 4.7: Tabla del ajuste de threshold.

- `threshold_effort(rep_name, path)`. Lo que se pretende con esta función es visualizar la evolución del valor del esfuerzo en relación al *threshold*, cuando este varía. El resultado se presenta en un diagrama de barras, como se puede apreciar a continuación:

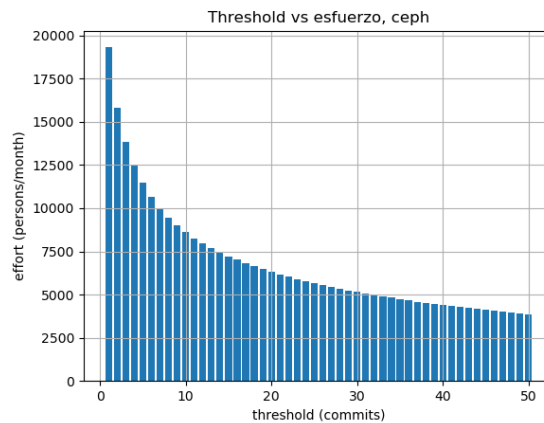


Figura 4.7: Gráfica threshold frente al esfuerzo, modelo Git2effort, repositorio Ceph.

- `threshold_saturation(rep_name, author_counter, path)`. Esta función también se centra en el estudio del modelo Git2effort, y nos muestra la cantidad de desarrolladores que sobrepasan el valor umbral de commits y aquellos que no lo hacen. Se tiene en cuenta todo el histórico, por año, y se utiliza un rango de valores impares de *threshold* del 1-49. Al igual que la anterior, esta función muestra los resultados en un diagrama de barras:

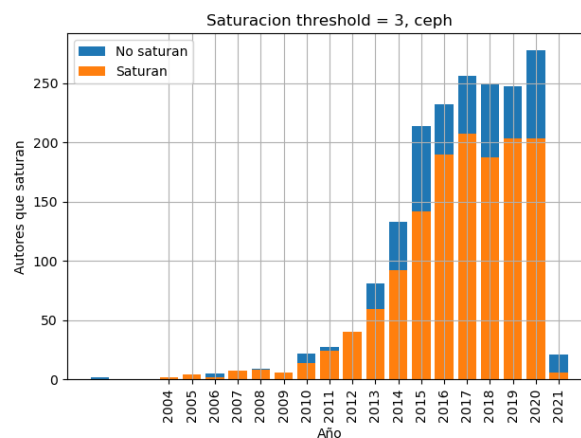


Figura 4.8: Gráfica saturación del threshold, modelo Git2effort

- `return_of_investment(rep_name, authorDict, total_effort, threshold)`.

Esta función se encarga de calcular y mostrar los resultados del Return of Investment (RoI). Esta métrica nos da una aproximación que compara el beneficio o la utilidad obtenida en relación a la inversión realizada por una empresa. Un valor de 1 indica que la empresa ha obtenido la misma utilidad que lo que ha invertido, mientras que un valor de 2 quiere decir que ha recibido el doble. Cuanto más alto el valor de RoI, mejor:

Dominio	RoI
web	45
translatewiki	44
mediawiki	38
gmail	2
wikimedia	1

Cuadro 4.8: RoI, repositorio Mediawiki.

- `html_report(rep_name, ef_df, sc_df, t_df, roi_df, threshold, path)`.

Esta función toma como parámetros el nombre del repositorio, junto con todos los dataframes generados en las funciones anteriores destinadas a la presentación de resultados mediante tablas, el *threshold* del repositorio y el path, donde se almacenan los ficheros de salida de la herramienta, y devuelve un archivo HTML que contiene todas las métricas obtenidas.

Hay que destacar, que todas las funciones que devuelven contenido gráfico, al igual que `html_report`, imprimen sus resultados sobre un fichero HTML. De esta forma, se exportan dos ficheros HTML:

- `Repositorio_tables.html`. En este fichero se pueden consultar los resultados de las tablas.
- `Repositorio_graphs.html`. En este otro fichero podemos examinar las gráficas.

Además de los archivos HTML, se generan dos PDF a partir del contenido web. No existe una función como tal, pero hay unas líneas de código dedicadas a este apartado. En este caso también se diferencian dos, uno tablas y otro de gráficas. Para llevarlo a cabo se emplea la

biblioteca `xhtml2pdf`, más concretamente la función `pisa.CreatePDF`, que toma como parámetros el fichero origen, HTML, y el fichero destino, PDF. De esta manera, tenemos dos alternativas de consulta.

Capítulo 5

Resultados

En este capítulo se incluyen los resultados obtenidos de los análisis llevados a cabo utilizando los modelos Git2effort y SLOCCount.

5.1. Estimación inicial

Para entrar en contacto con los modelos de estimación Git2effort y SLOCCount se realizó un primer experimento en el que calculamos y comparamos el esfuerzo de repositorios con una alta volumetría. En el caso de SLOCCount no se modificó ningún parámetro a la hora de estimar. Sin embargo, cuando ejecutamos Git2effort se fue ajustando el *threshold* dependiendo del repositorio. A la hora de realizar este análisis se tuvieron en cuenta una serie de condiciones que definían varios escenarios. Por un lado, se establecieron tres intervalos de tiempo: i) considerando todo el tiempo del desarrollo, ii) los tres primeros años y iii) los seis primeros años. Por otro lado, también se diferenció entre directorios: a) todos los directorios y b) solo aquellos que contenían ficheros con código fuente. Los casos de estudio que se definieron de acuerdo a esas restricciones fueron los siguientes:

- Git2effort frente SLOCCount teniendo en cuenta todo el tiempo de desarrollo y todos los directorios.
- Git2effort frente SLOCCount teniendo en cuenta únicamente los tres primeros años de desarrollo y todos los directorios.

- Git2effort frente SLOCCCount teniendo en cuenta únicamente los seis primeros años y todos los directorios.
- Git2effort frente SLOCCCount teniendo en cuenta todo el tiempo de desarrollo y los directorios con código fuente.
- Git2effort frente SLOCCCount teniendo en cuenta únicamente los tres primeros años de desarrollo y los directorios con código fuente.
- Git2effort frente SLOCCCount teniendo en cuenta únicamente los seis primeros años y los directorios con código fuente.

La presentación de los resultados de las estimaciones siguen el siguiente orden: primero analizaremos los casos que contenían todos los directorios y después aquellos que se limitaba a los directorios con código fuente.

5.1.1. Ficheros en general

En la siguiente tabla 5.1 se muestra el *threshold* empleado para cada repositorio con Git2effort en la primera fase de experimentación:

Repositorio	Ceph	MediaWiki	Moodle
Threshold [commits]	24	36	14

Cuadro 5.1: Threshold por repositorio.

El objetivo del *threshold* es el de diferenciar entre desarrolladores full-time y non-full-time. Este valor depende del proyecto. Los valores utilizados, son resultado del feedback generado por algunos desarrolladores de los diferentes proyectos [7].

Como puede observarse en la tabla 5.2, los valores de esfuerzo obtenidos por el modelo Git2effort son considerablemente superiores a los resultados del modelo SLOCCCount para el repositorio de MediaWiki. El resto presenta unos resultados más aproximados entre sí:

Repositorio	Ceph	MediaWiki	Moodle
Estimación Git2effort [persons-month]	5774	4941	7296
Estimación SLOCCCount [persons-month]	4534	2284	7496

Cuadro 5.2: Resultados de las estimaciones de los repositorios en el presente.

En la siguiente tabla 5.3 al considerar únicamente los tres primeros años de desarrollo, se aprecia, lógicamente, un descenso de los valores obtenidos para ambos modelos. En este caso los valores de MediaWiki no distan tanto:

Repositorio	Ceph	MediaWiki	Moodle
Estimación Git2effort [persons-month]	46	283	271
Estimación SLOCCCount [persons-month]	168	378	464

Cuadro 5.3: Resultados de las estimaciones de los repositorios en los tres primeros años.

Por último, si observamos la tabla 5.4 correspondiente al periodo de tiempo de los seis primeros años de desarrollo, el esfuerzo aumenta tanto para Git2effort, como para SLOCCCount, pero se mantienen la diferencia del repositorio de MediaWiki:

Repositorio	Ceph	MediaWiki	Moodle
Estimación Git2effort [persons-month]	148	882	1256
Estimación SLOCCCount [persons-month]	398	1580	1181

Cuadro 5.4: Resultados de las estimaciones de los repositorios en los seis primeros años.

Existe una diferencia que se puede apreciar a simple vista entre los resultados de los diferentes repositorios. Como hemos visto en las tablas, el valor de esfuerzo estimado con ambos modelos varía según que repositorio estemos analizando. El resultado obtenido para Ceph, por ejemplo, es menor que para Moodle. Esto se debe a que, obviamente, la actividad en el repositorio de Moodle es mucho mayor que en el repositorio de Ceph. Los resultados nos indican claramente que el número de líneas de código obtenidas por SLOCCCount y el número de commits recopilados por Git2effort es mucho mayor en un repositorio que en otro.

Tiene sentido que si tomamos mayores intervalos de tiempo el esfuerzo aumente, ya que la actividad de desarrollo también crece. Sin embargo, a la hora de tener en cuenta periodos más cortos, los resultados de las estimaciones nos indican que hay menos actividad y, como consecuencia, se requiere un menor esfuerzo.

Cabe destacar también el comportamiento de MediaWiki sobre históricos de datos de largos periodos de tiempo. Presenta una mayor diferencia en las estimaciones cuando aumenta la actividad.

5.1.2. Ficheros con código fuente

A continuación se muestran los resultados que involucran únicamente a los ficheros con código fuente.

En la tabla 5.5, podemos observar como, en general, los valores de esfuerzo experimentan un ligero descenso frente a los escenarios en los que se contemplaban todos los directorios. La diferencia en MediaWiki persiste:

Repositorio	Ceph	MediaWiki	Moodle
Estimación Git2effort [persons-month]	4676	4462	6960
Estimación SLOCCount [persons-month]	3446	2230	7384

Cuadro 5.5: Resultados de las estimaciones de los repositorios en el presente de directorios con código fuente.

El comportamiento que se observa para los diferentes periodos de tiempo se mantiene. En este caso, tabla 5.6, decrece si el análisis abarca solo los tres primeros años:

Repositorio	Ceph	MediaWiki	Moodle
Estimación Git2effort [persons-month]	45	276	193
Estimación SLOCCount [persons-month]	107	368	449

Cuadro 5.6: Resultados de las estimaciones de los repositorios en los tres primeros años de directorios con código fuente.

En el caso del último escenario 5.7 no se aprecia ninguna diferencia en cuanto a la tendencia. Considerar seis años de desarrollo en el repositorio provoca el mismo crecimiento en los valores de esfuerzo.

Repositorio	Ceph	MediaWiki	Moodle
Estimación Git2effort [persons-month]	144	864	1166
Estimación SLOCCount [persons-month]	269	1564	1140

Cuadro 5.7: Resultados de las estimaciones de los repositorios en los seis primeros años de directorio con código fuente.

Como hemos comentado previamente, el comportamiento no varía en cada escenario. Se observan los mismos aumentos y descensos atendiendo a los periodos de tiempo. La desigualdad entre repositorios también se mantiene.

Dependiendo si tenemos en cuenta todos los directorios, o solo aquellos que contengan código fuente, sí que se observa un ligero descenso en el esfuerzo. Es razonable pensar que si no estamos teniendo en cuenta la misma actividad esto provoque que el esfuerzo estimado cambie también.

5.2. Ajuste de Threshold en Git2effort

Tras este primer estudio de los repositorios, al darnos cuenta de la diferencia en los valores de estimación ofrecidos por ambos modelos decidimos realizar otro experimento. En este caso optamos por aproximar las estimaciones y que fueran lo más semejante posible.

Para ello se ajustó el *threshold* del modelo Git2effort de acuerdo a los datos de cada repositorio, de forma que el resultado se acercara lo máximo posible al de SLOCCount. Para obtener un visión más amplia se tuvo en cuenta toda el histórico disponible de los repositorios. Los resultados se muestran a continuación en la tabla 5.8:

Año	Líneas de código	COCOMO	Threshold Git2effort	Commits	Git2effort
2004	11.630	31	1	124	24
2005	39.877	115	1	556	72
2006	57.434	168	1	988	114
2007	145.415	447	1	2.280	168
2008	177.969	553	1	5.240	270
2009	130.209	398	1	8.980	306
2010	153.366	473	2	12.259	441
2011	219.865	691	2	17.819	681
2013	381.272	1.231	5	32.785	1.213
2014	475.777	1.554	11	42.021	1.548
2015	554.954	1.826	22	53.492	1.810
2016	750.604	2.508	24	68.603	2.495
2017	872.279	2.936	31	84.564	2.913
2018	964.957	3.265	37	98.000	3.271
2019	1.103.425	3.759	41	112.826	3.725
2020	1.313.625	4.514	38	126.394	4.516

Cuadro 5.8: Evolución del esfuerzo, repositorio Ceph.

Como se puede observar, la tabla contiene varios campos: el año hasta el que se recogen datos, las líneas de código, la estimación del modelo SLOCCount (COCOMO), el *threshold* de Git2effort, los commits realizados y, por último, el esfuerzo estimado por el modelo Git2effort. Con estos datos pretendemos visualizar la evolución del *threshold*.

En el repositorio de MediaWiki diferenciamos dos tablas, ya que nos encontramos con un comportamiento un tanto extraño en los datos correspondientes al modelo SLOCCount. La primera tabla 5.9 refleja dicho comportamiento:

Año	Líneas de código	COCOMO	Threshold Git2effort	Commits	Git2effort
2003	55.598	163	1	654	162
2004	92.971	279	10	4.827	279
2005	124.012	378	19	9.034	308
2006	184.387	574	17	12.987	580
2007	306.906	980	10	17.587	980
2008	483.584	1.580	6	24.278	1.591
2009	598.475	1.977	8	29.585	1.941
2010	696.437	2.318	10	35.287	2.293
2011	805.399	2.700	11	43.107	2.706
2012	903.176	3.046	14	50.154	3.076
2013	942.291	3.184	21	56.433	3.169
2014	356.409	1.147	136	64.109	1.148
2015	394.310	1.275	136	71.300	1.278
2016	507.590	1.486	128	78.294	1.483
2017	964.957	1.663	123	84.265	1.660
2018	569.740	1.877	116	90.641	1.875
2019	599.042	1.979	119	98.096	1.975
2020	684.947	2.278	109	105.457	2.284

Cuadro 5.9: Evolución del esfuerzo, repositorio MediaWiki con ruido.

Se puede apreciar como la tendencia de crecimiento de las líneas de código va incrementando desde el año 2003 hasta el 2014. En el año 2013 se recopilan un total de 942.291 líneas de código, y sin embargo, al año siguiente el total desciende a 394310. Este suceso ocurría nuevamente entre los años 2017 y 2018.

Comparando la información almacenada en la base de datos de los años 2013 y 2014, nos dimos cuenta por qué estaba ocurriendo esto. Filtrando en las tablas de la base de datos de la

siguiente forma nos hizo entender el suceso.

En primer lugar ordenamos los datos en orden ascendente por líneas de código para comprobar que ficheros eran los más pesados:

```
SELECT *
FROM `mediawiki_sloccount_before_2014`
ORDER BY Code_Lines DESC
```

index	Directory	Parent_Directory	File_Name	Language	Code_Lines	▼ 1
264	includes	top_dir	Installer.i18n.php	php	18751	
799	languages	top_dir	MessagesQqq.php	php	8718	
1650	resources	top_dir	jquery.js	javascript	6545	
754	languages	top_dir	MessagesEn.php	php	4463	
1573	maintenance	top_dir	messages.inc	pascal	4196	
975	languages	top_dir	MessagesJa.php	php	3948	
720	languages	top_dir	MessagesAr.php	php	3946	
918	languages	top_dir	MessagesMk.php	php	3946	
930	languages	top_dir	MessagesSr_ec.php	php	3910	
830	languages	top_dir	MessagesUk.php	php	3902	
953	languages	top_dir	MessagesDiq.php	php	3882	
743	languages	top_dir	MessagesFa.php	php	3879	
858	languages	top_dir	MessagesHe.php	php	3854	
981	languages	top_dir	MessagesNl.php	php	3845	
1072	languages	top_dir	MessagesRu.php	php	3842	
1052	languages	top_dir	MessagesKo.php	php	3835	
843	languages	top_dir	MessagesKsh.php	php	3830	
1030	languages	top_dir	MessagesId.php	php	3815	
1070	languages	top_dir	MessagesFr.php	php	3808	
1057	languages	top_dir	MessagesVi.php	php	3799	
964	languages	top_dir	MessagesDe.php	php	3763	
1016	languages	top_dir	MessagesSr_el.php	php	3753	
1042	languages	top_dir	MessagesPl.php	php	3749	
1041	languages	top_dir	MessagesCs.php	php	3741	
1069	languages	top_dir	MessagesGl.php	php	3741	
1071	languages	top_dir	MessagesTr.php	php	3724	
922	languages	top_dir	MessagesFrp.php	php	3719	
1076	languages	top_dir	MessagesPt.php	php	3710	
891	languages	top_dir	MessagesMl.php	php	3703	

Figura 5.1: Análisis MediaWiki

Realizando la misma operación para los datos del año posterior:

```
SELECT *
FROM `mediawiki_sloccount_before_2015`
ORDER BY Code_Lines DESC
```

index	Directory	Parent_Directory	File_Name	Language	Code_Lines
1855	resources	top_dir	jquery.js	javascript	6850
1944	resources	top_dir	oojs-ui.js	javascript	5823
381	includes	top_dir	Parser.php	php	3866
1943	resources	top_dir	sinon-1.10.3.js	javascript	3716
150	includes	top_dir	User.php	php	2781
1184	languages	top_dir	Language.php	php	2673
137	includes	top_dir	EditPage.php	php	2516
431	includes	top_dir	Title.php	php	2478
1862	resources	top_dir	moment.js	javascript	2203
310	includes	top_dir	GlobalFunctions.php	php	2101
2215	tests	top_dir	FileBackendTest.php	php	1935
172	includes	top_dir	OutputPage.php	php	1915
42	includes	top_dir	WikiPage.php	php	1878
588	includes	top_dir	Database.php	php	1831
240	includes	top_dir	LocalFile.php	php	1748
691	includes	top_dir	ApiBase.php	php	1746
1854	resources	top_dir	jquery.qunit.js	javascript	1672
748	includes	top_dir	jsminplus.php	php	1666
1954	resources	top_dir	jquery.ui.datepicker.js	javascript	1520
1724	resources	top_dir	Resources.php	php	1514
2413	tests	top_dir	LanguageTest.php	php	1418
382	includes	top_dir	Preprocessor_Hash.php	php	1387
322	includes	top_dir	Linker.php	php	1386
2503	top_dir	NULL	autoload.php	php	1321
358	includes	top_dir	FormatMetadata.php	php	1319
394	includes	top_dir	Preprocessor_DOM.php	php	1293
38	includes	top_dir	Article.php	php	1280
704	includes	top_dir	DefaultSettings.php	php	1251
288	includes	top_dir	SwiftFileBackend.php	php	1212

Figura 5.2: Análisis MediaWiki

Los archivos que contienen más líneas de código no coinciden de un año para otro. De hecho, para el año 2013 se observan una gran cantidad de scripts que comienzan con el nombre de Messages.

Comprobamos que ocurría con esos ficheros en el año 2014:

```
SELECT *
FROM `mediawiki_sloccount_before_2015`
WHERE File_Name LIKE "Messages*"
```

index	Directory	Parent_Directory	File_Name	Language	Code_Lines
902	languages	top_dir	MessagesEn.php	php	479
873	languages	top_dir	MessagesAr.php	php	364
893	languages	top_dir	MessagesFa.php	php	355
1053	languages	top_dir	MessagesSr_ec.php	php	333
1081	languages	top_dir	MessagesDe.php	php	328
996	languages	top_dir	MessagesHe.php	php	324
1182	languages	top_dir	MessagesKk_arab.php	php	323
1044	languages	top_dir	MessagesMk.php	php	316
1174	languages	top_dir	MessagesRu.php	php	315
1026	languages	top_dir	MessagesMI.php	php	313
1157	languages	top_dir	MessagesKo.php	php	311
1017	languages	top_dir	MessagesKk_cyrl.php	php	307
962	languages	top_dir	MessagesKk_latn.php	php	306
1095	languages	top_dir	MessagesNI.php	php	305
1072	languages	top_dir	MessagesDiq.php	php	302
1172	languages	top_dir	MessagesFr.php	php	301
972	languages	top_dir	MessagesUk.php	php	299
1089	languages	top_dir	MessagesJa.php	php	298
950	languages	top_dir	MessagesCe.php	php	296
1042	languages	top_dir	MessagesZh_hans.php	php	295
1137	languages	top_dir	MessagesId.php	php	295
1161	languages	top_dir	MessagesVi.php	php	293
1099	languages	top_dir	MessagesNds_nl.php	php	291
1152	languages	top_dir	MessagesEl.php	php	288
1173	languages	top_dir	MessagesTr.php	php	287
1047	languages	top_dir	MessagesFrp.php	php	284
1147	languages	top_dir	MessagesCs.php	php	283
1077	languages	top_dir	MessagesArz.php	php	277
1024	languages	top_dir	MessagesMr.php	php	272
1043	languages	top_dir	MessagesEo.php	php	272

Figura 5.3: Tablas almacenadas repositorio Ceph

La cantidad de líneas de código que se almacenan para estos ficheros desciende casi diez veces respecto al año anterior.

Para terminar de cerciorarnos si era esta la razón definitiva por la que las líneas de código descendían tan bruscamente de un año a otro, realizamos una última prueba. Nos pareció buena idea diseñar un script que comparara las líneas de código de todos los ficheros de cada periodo de tiempo. El programa está constituido por las siguientes funciones:

- `data_fetch(rep_name, year)`. Esta función recibe como parámetros el nombre del repositorio y el año de interés, devolviendo un diccionario que almacena el directorio en el que se encuentra el fichero, su nombre y el número de líneas que contiene. Los datos son extraídos de la base de datos.
- `dict_compare(dict_a, dict_b)`. Esta función recibe como parámetros dos diccionarios, a y b, retornando un diccionario que contiene aquellos ficheros que comparten, y un segundo diccionario que nos indica la diferencia de líneas de los ficheros.

Una vez finalizado el análisis de este comportamiento, decidimos no tener en cuenta esos registros. En lugar de eliminarlos preferimos mantenerlos almacenados y crear un flag que nos permitiera prescindir de ellos en cualquier momento utilizando filtros a la hora de lanzar queries contra la base de datos. Para ello, añadimos el campo binario *Exclude*:

index	Directory	Parent_Directory	File_Name	Language	Code_Lines	Exclude
713	languages	top_dir	MessagesLoz.php	php	845	1
714	languages	top_dir	MessagesGan_hant.php	php	1881	1
715	languages	top_dir	MessagesBjn.php	php	3261	1
716	languages	top_dir	MessagesTly.php	php	854	1
717	languages	top_dir	MessagesLez.php	php	1095	1
718	languages	top_dir	MessagesDe_formal.php	php	522	1
719	languages	top_dir	MessagesAst.php	php	3515	1

Figura 5.4: Tablas almacenadas repositorio Ceph

Esta actualización de las tablas se llevó a cabo con la siguiente sentencia:

```
UPDATE table_name SET Exclude =
CASE
WHEN File_Name like "Messages%" and Language like "php" THEN 1
WHEN File_Name like "Language%" and Language like "php" THEN 1
ELSE 0
END
```

En la exportación de datos es tan simple como asignar el valor que corresponda a este nuevo campo en el filtro de la query:

```
WHERE Exclude = 1
```

Los resultados tras estas modificaciones se pueden apreciar en la tabla 5.10:

Año	Líneas de código	COCOMO	Threshold Git2effort	Commits	Git2effort
2003	22.840	64	20	654	64
2004	45.482	132	51	4.827	132
2005	60.084	176	76	9.034	176
2006	82.449	246	84	12.987	246
2007	100.796	304	111	17.587	303
2008	124.145	379	134	24.278	379
2009	147.749	455	144	29.585	454
2010	191.333	597	136	35.287	595
2011	241.771	763	130	43.107	760
2012	276.365	878	135	50.154	879
2013	298.648	953	144	56.433	953
2014	316.863	1.014	156	64.109	1.014
2015	353.385	1.137	155	71.300	1.136
2016	414.716	1.345	143	78.294	1.341
2017	465.295	1.518	136	84.265	1.516
2018	526.441	1.728	128	90.641	1.727
2019	553.959	1.823	131	98.096	1.823
2020	639.755	2.120	119	105.457	2.114

Cuadro 5.10: Evolución del esfuerzo, repositorio MediaWiki sin ruido.

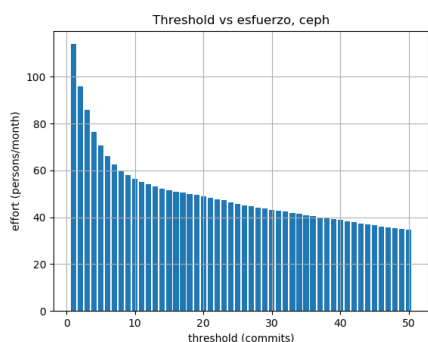
Ahora podemos ver como los datos ganan consistencia, centrando el estudio en los ficheros realmente relevantes.

5.3. Evolución del esfuerzo frente al threshold

En este apartado se describen los resultados de la evolución del esfuerzo dependiendo del valor umbral asignado.

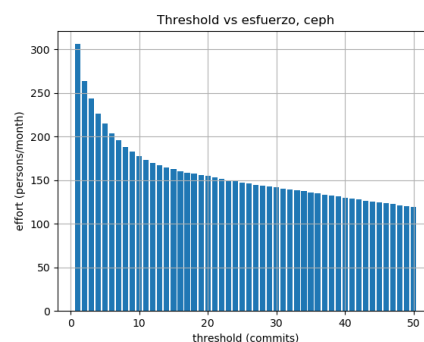
Al igual que en la sección 5.1 realizamos el análisis utilizando los mismos intervalos de tiempo.

Para el repositorio de Ceph:



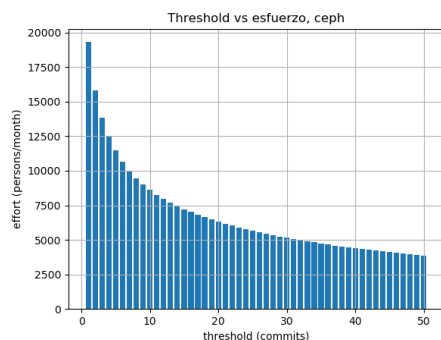
Esfuerzo

frente a *threshold*, Ceph, tres primeros años.



Esfuerzo

frente a *threshold*, Ceph, seis primeros años.

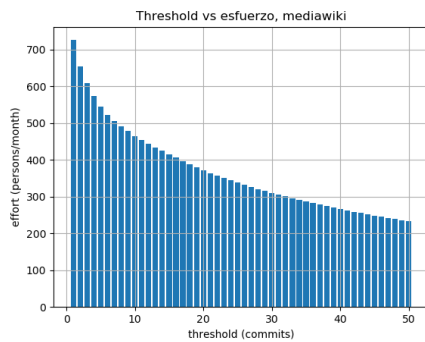


Esfuerzo

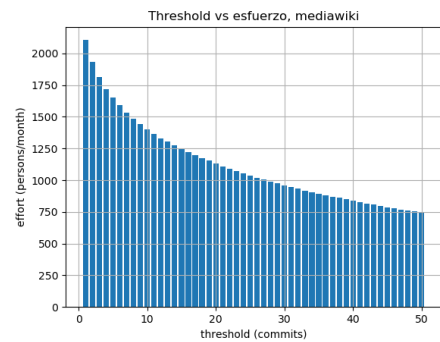
frente a *threshold*, Ceph, histórico completo.

Como podemos observar, para cualquiera de los históricos a medida que el umbral incrementa hace que el esfuerzo se reduzca. Cuando se alcanzan valores de *threshold* entre 40 y 50 commits el esfuerzo tiende a estabilizarse.

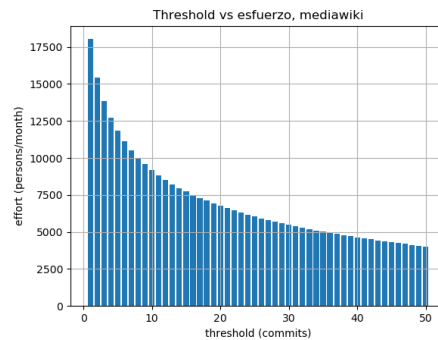
En el caso del repositorio MediaWiki se observa el mismo comportamiento:



Esfuerzo frente a *threshold*, Mediawiki, tres primeros años.

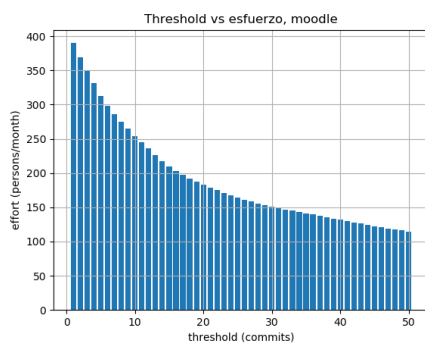


Esfuerzo frente a *threshold*, MediaWiki, seis primeros años.

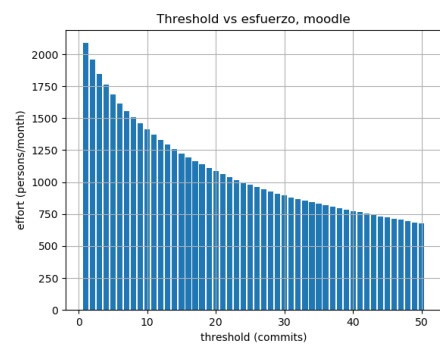


Esfuerzo frente a *threshold*, MediaWiki, histórico completo.

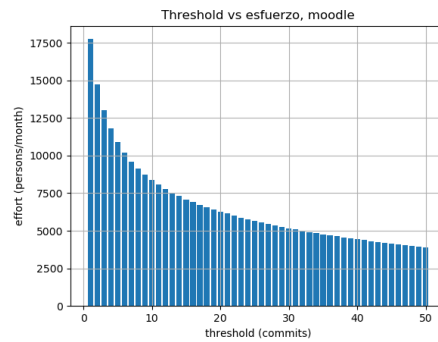
Para el repositorio de Moodle también se repite la misma tendencia:



Esfuerzo frente a *threshold*, Moodle, tres primeros años.



Esfuerzo frente a *threshold*, Moodle, seis primeros años.



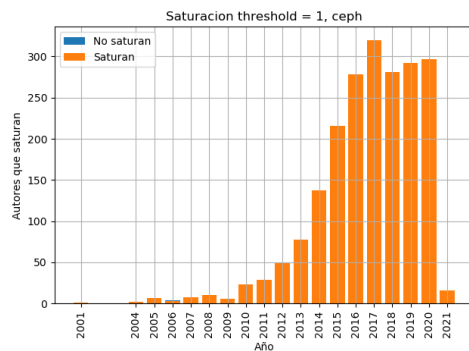
Esfuerzo frente a *threshold*, Moodle, histórico completo.

Para todos los repositorios se detecta el mismo patrón en la evolución del esfuerzo frente al *threshold*. Existe un primer intervalo en el que el esfuerzo desciende, a medida que el *threshold* aumenta. Una vez dicho umbral alcanza valores entre 40 y 50 commits, este provoca una estabilización en el esfuerzo. Obviamente, en esa primera fase en la que el *threshold* toma valores muy pequeños, la mayoría de usuarios se catalogan como Full-Time developers, obteniendo los valores de esfuerzo más altos. Cuando el *threshold* tiene valores más altos, la categorización de los desarrolladores se reparte más; como consecuencia, en el esfuerzo total intervienen tanto Full-Time developers como Non-Full-Time developers.

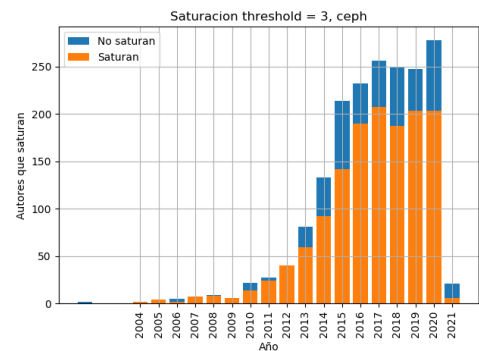
5.4. Saturación del threshold

En este apartado, se muestran los resultados obtenidos para la saturación del *threshold*. En las gráficas se puede apreciar cómo al aumentar de valor el umbral, va variando el número de autores que lo saturan.

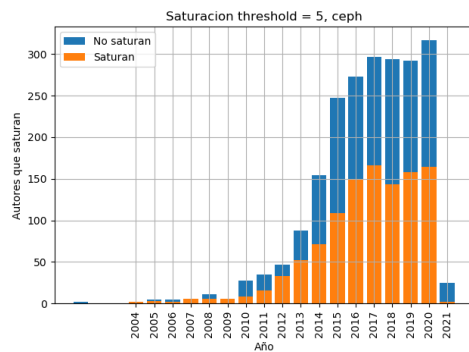
A continuación se muestran las gráficas correspondientes al repositorio Ceph:



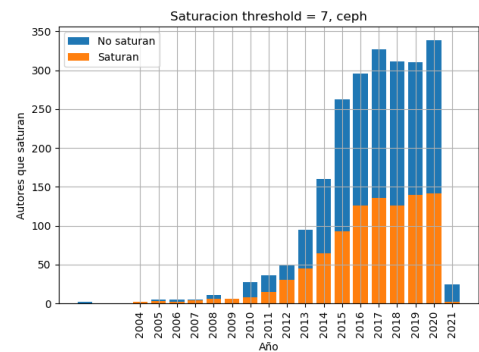
Saturación del threshold a lo largo de los años,
t = 1, repositorio Ceph.



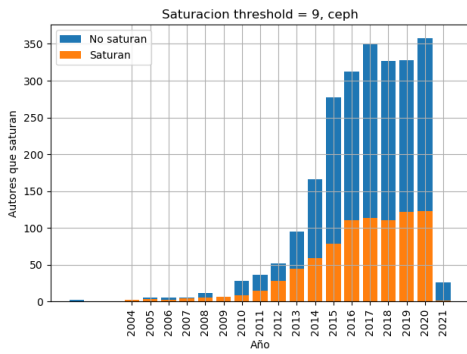
Saturación del threshold a lo largo de los años,
t = 3, repositorio Ceph.



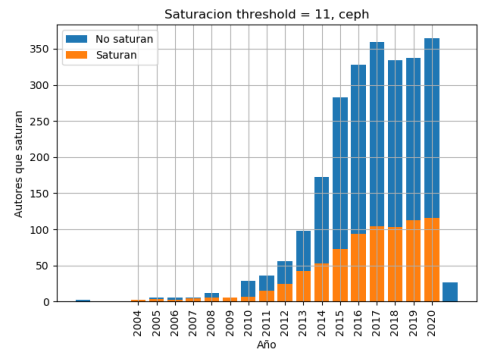
Saturación del threshold a lo largo de los años,
t = 5, repositorio Ceph.



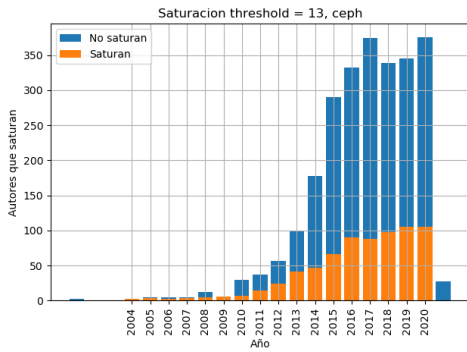
Saturación del threshold a lo largo de los años,
t = 7, repositorio Ceph.



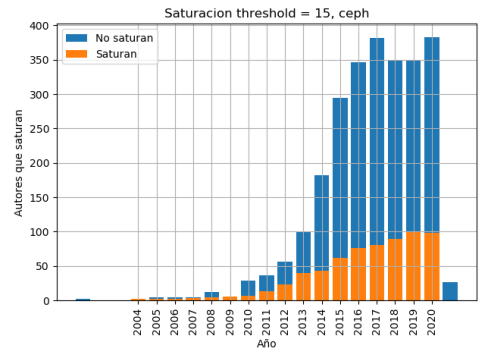
Saturación del threshold a lo largo de los años, $t = 9$, repositorio Ceph.



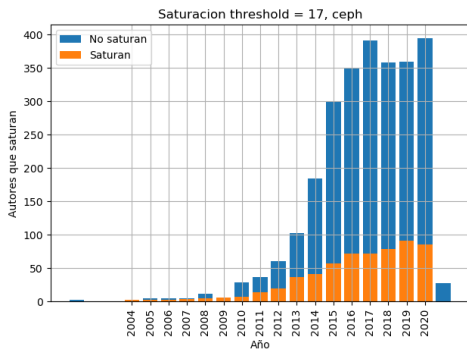
Saturación del threshold a lo largo de los años, $t = 11$, repositorio Ceph.



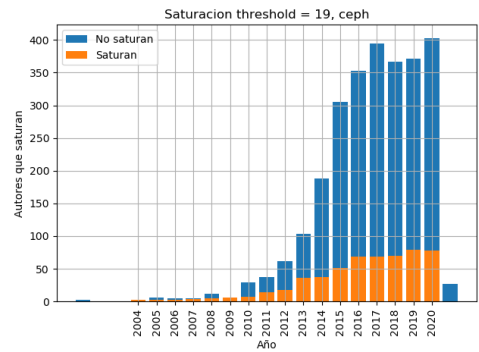
Saturación del threshold a lo largo de los años, $t = 13$, repositorio Ceph.



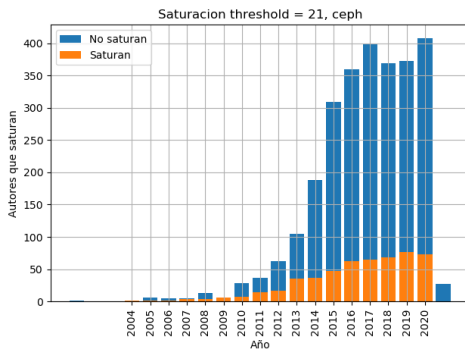
Saturación del threshold a lo largo de los años, $t = 15$, repositorio Ceph.



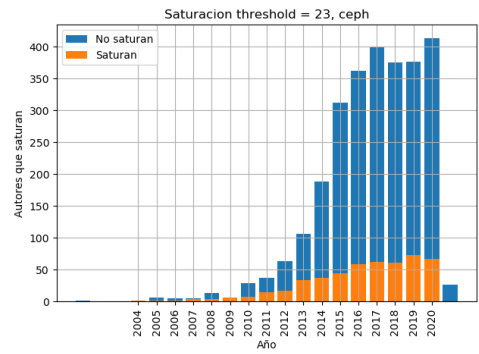
Saturación del threshold a lo largo de los años, $t = 17$, repositorio Ceph.



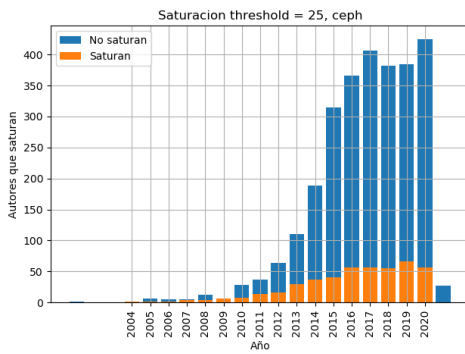
Saturación del threshold a lo largo de los años, $t = 19$, repositorio Ceph.



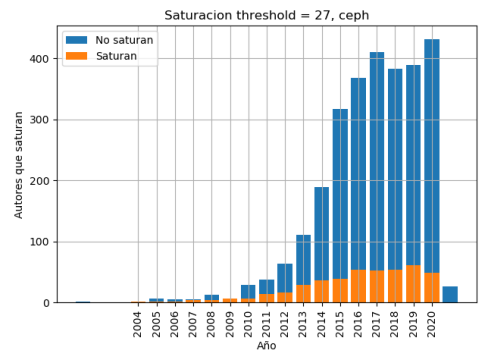
Saturación del threshold a lo largo de los años, $t = 21$, repositorio Ceph.



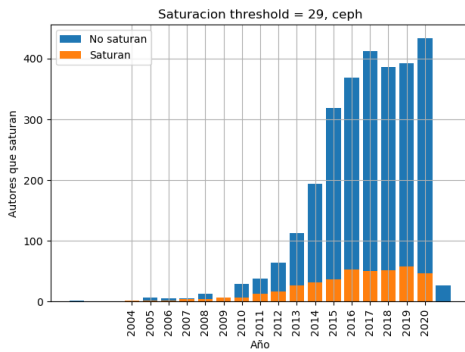
Saturación del threshold a lo largo de los años, $t = 23$, repositorio Ceph.



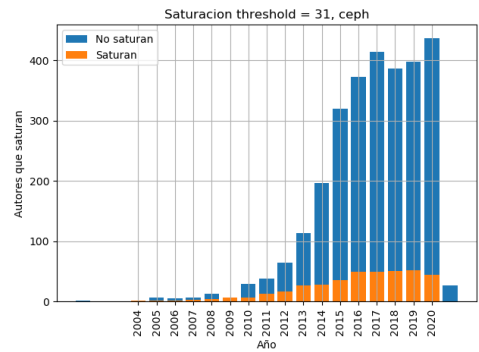
Saturación del threshold a lo largo de los años, $t = 25$, repositorio Ceph.



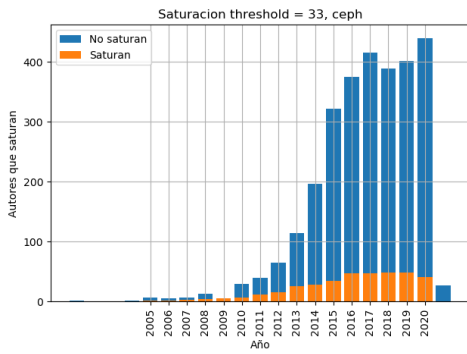
Saturación del threshold a lo largo de los años, $t = 27$, repositorio Ceph.



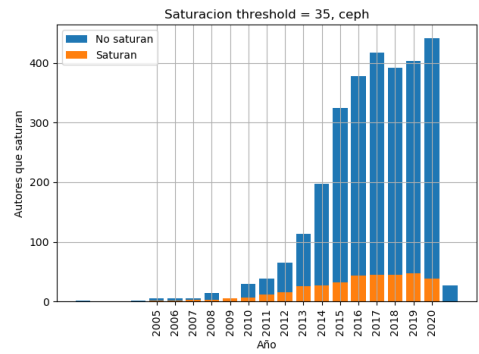
Saturación del threshold a lo largo de los años, $t = 29$, repositorio Ceph.



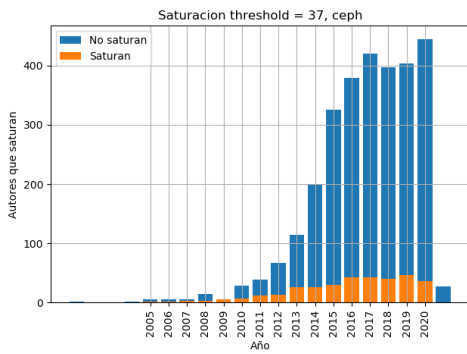
Saturación del threshold a lo largo de los años, $t = 31$, repositorio Ceph.



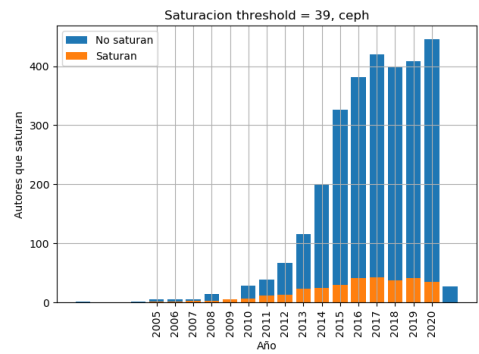
Saturación del threshold a lo largo de los años,
t = 33, repositorio Ceph.



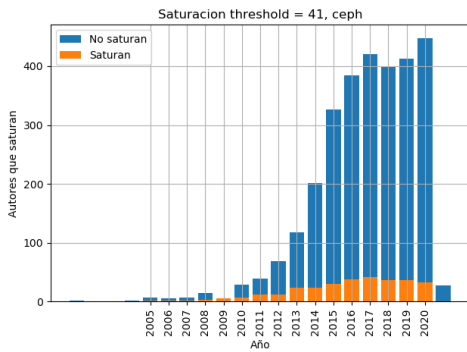
Saturación del threshold a lo largo de los años,
t = 35, repositorio Ceph.



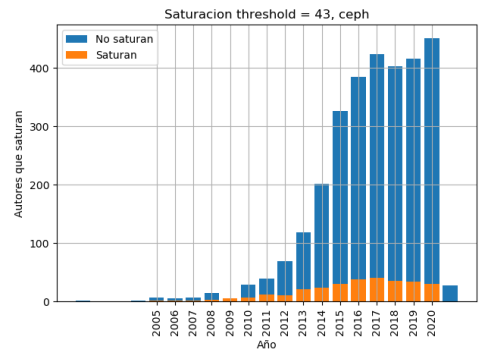
Saturación del threshold a lo largo de los años,
t = 37, repositorio Ceph.



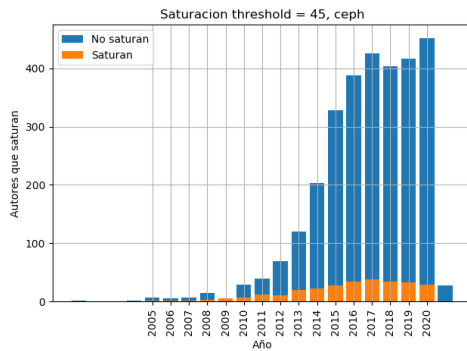
Saturación del threshold a lo largo de los años,
t = 39, repositorio Ceph.



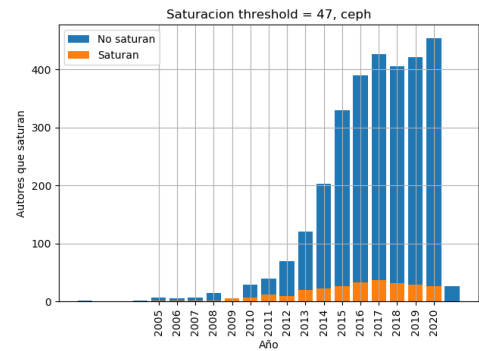
Saturación del threshold a lo largo de los años,
t = 41, repositorio Ceph.



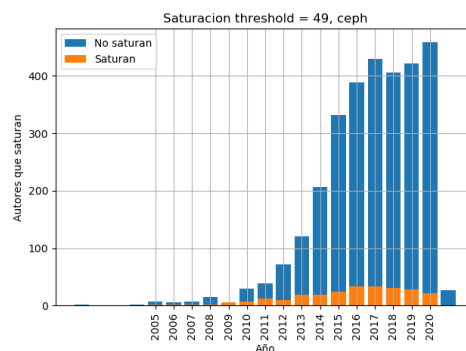
Saturación del threshold a lo largo de los años,
t = 43, repositorio Ceph.



Saturación del threshold a lo largo de los años,
t = 45, repositorio Ceph.



Saturación del threshold a lo largo de los años,
t = 47, repositorio Ceph.



Saturación del threshold a lo largo de los años,
t = 49, repositorio Ceph.

La asignación de diferentes valores de *threshold*, nos proporcionan una visión de cuantos usuarios superan dicho umbral. Este análisis nos puede ayudar a la hora de escoger el valor de *threshold* para la estimación de un repositorio. Dependiendo de los resultados que obtengamos en estas gráficas, podremos hacernos una idea aproximada de que umbral se ajusta mejor. De esta forma, se consigue una categorización de desarrolladores adecuada y como consecuencia unos resultados en la estimación más realistas.

5.5. RoI

En este apartado calculamos el RoI, Return of Investment. Este parámetro nos da información sobre cuan eficiente es la inversión realizada por una empresa. Viene dado por la siguiente formula:

$$RoI = \frac{E_{total} - E_{empresa}}{E_{empresa}} \quad (5.1)$$

Este cálculo solo se puede obtener gracias a los datos recopilados del modelo Git2effort, ya que en el modelo SLOCCount solo se emplean las líneas de código.

A continuación, en las tablas 5.11, 5.12 y 5.13 se muestran los resultados correspondientes a las empresas con mayor aportación, por repositorio:

Dominio	intel	gmail	inktank	suse	redhat
RoI	24,30	11,08	9,55	7,20	1,36

Cuadro 5.11: RoI, repositorio Ceph

Dominio	web	translatewiki	mediawiki	gmail	wikimedia
RoI	45,34	44,12	38,75	2,05	1,68

Cuadro 5.12: RoI, repositorio Mediawiki

Dominio	nicols	catalyst-au	open	gmail	moodle
RoI	72,38	54,99	20,30	10,23	2,13

Cuadro 5.13: RoI, repositorio Moodle

En aquellos casos que el campo dominio presente el valor gmail u otros que no sean reconocibles a primera vista, se refiere a desarrolladores independientes, es decir, individuos que aparentemente no trabajan para una empresa concreta.

La diferencia de resultados por repositorio se debe a la diferencia de volúmenes. Los valores de *RoI* cercanos a 1 indican que la empresa recibe lo mismo que aporta. Si este valor aumenta, lo hace a su vez el beneficio que reciben.

Esta métrica nos dice cuan eficiente es la aportación de cada empresa. Es un indicador más, que nos permite realizar el seguimiento del esfuerzo por empresa.

Capítulo 6

Conclusiones

En esta sección se repasan los objetivos que se han conseguido llevar a cabo y se presentan algunos problemas que han surgido. Además, se añaden otros apartados en los que se refleja que conocimientos específicos del grado se han aplicado y cuáles se han obtenido durante el proyecto. Se cierra la sección con algunas posibles ideas de futuro, con el fin de ampliar y mejorar todo este desarrollo.

6.1. Consecución de objetivos

Recordando el capítulo 2 podemos diferenciar entre el objetivo principal y los específicos.

Como objetivo principal describíamos que la meta es desarrollar una herramienta que nos permitiera comparar los modelos de estimación COCOMO y Git2effort. Este objetivo se ha conseguido a pesar de la problemática que presentaban algunos orígenes de datos que acabamos resolviendo en el capítulo 5. Los métricas que se exportan nos ayudan a entender mejor ambos modelos. Es importante comprender cómo funcionan para poder realizar estimaciones correctamente. Una vez se conocen en profundidad podemos comenzar a cotejar los resultados y extraer nuestras propias conclusiones.

Los objetivos específicos consistían en fases intermedias que nos iban conduciendo a alcanzar el propósito final. Para poder abordar toda esta tarea ha sido imprescindible el hecho de contextualizar y empaparse de la temática central. Esto nos ha ayudado a definir que elementos consideramos necesarios a la hora de comparar los modelos, y para saber que distribución y funcionalidad queremos que siga el programa.

6.2. Aplicación de lo aprendido

Durante el desarrollo de la herramienta he empleado todos lo aprendido durante el grado. Es cierto que la temática del proyecto está totalmente enfocada en la programación y, por lo tanto, tengo que destacar las siguientes asignaturas:

- **Fundamentos de la programación.** Fue mi primer contacto con este mundo. Establecí mis pilares en desarrollo con esta materia.
- **Arquitectura de redes de ordenadores.** El entorno de trabajo, en la mayor parte del tiempo, ha sido en remoto, por lo que he tenido que emplear conexiones SSH, tanto a la hora de programar, como por necesidades de red.
- **Programación de sistemas de telecomunicación.** Esta asignatura me hizo enfrentarme a los primeros desarrollos con más complejidad, diseñando funciones y estructurando adecuadamente un proyecto.
- **Servicios y aplicaciones en redes de ordenadores.** Fueron los primeros pasos con Python y HTML, entre otros. Ha sido la principal fuente de este TFG.
- **Software de sistemas.** Al igual que Programación de sistemas de telecomunicación, aprender nuevos lenguajes me ayudo a ser más versátil, ampliar conocimientos y saber más de cerca, cómo y por qué estaba haciendo lo que hacía.

6.3. Lecciones aprendidas

A pesar de aplicar todos los conocimientos adquiridos durante la carrera, en ocasiones, me he enfrentado a nuevos problemas y retos. Resolver estos obstáculos, me han proporcionado nuevas habilidades:

- Gestión de incidencias y solicitud de recursos a los administradores de los servidores de la universidad.
- Profundización en mis habilidades de MySQL.
- Incremento de habilidades programando en Python.

- Incremento de habilidades en el manejo y representación de datos con las bibliotecas de Pandas y Matplotlib.
- Incremento de habilidades en programación \LaTeX
- Mejora en el seguimiento y planificación de proyectos.

6.4. Trabajos futuros

Una vez finalizado este trabajo podemos visualizar con cierta perspectiva aspectos a mejorar e incluso ampliaciones del mismo:

- **Optimización del código.** Se podrían mejorar secciones del código consiguiendo así tiempos de ejecución más cortos y un script más limpio .
- **Introducción de nuevas métricas.** El trabajo proporciona una serie de recursos pero se podrían exportar más indicadores que ayuden en el estudio de los modelos.
- **Migración a Bigquery.** En el caso de que estemos trabajando con proyectos cuyas dimensiones sean muy grandes, cabe la posibilidad de emplear otras bases de datos más potentes como Bigquery.
- **Emplear nuevos modelos de estimación.** En este caso nos hemos centrado en Git2effort y COCOMO, pero se podrían involucrar más en la comparación.

Apéndice A

Manual de usuario

En este apartado se proporciona el manual de usuario. Se indica que necesitamos para utilizar la herramienta, y que pasos debemos seguir.

Podemos encontrar el *script* en el siguiente enlace https://github.com/victortm/compara_estimadores.

A.1. Requisitos

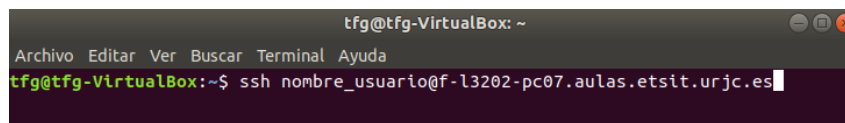
Para poder utilizar la herramienta, debemos tener instalados en el equipo los siguiente paquetes:

- Python - 3.4
- Perceval - 0.12
- Tabulate - 0.8.7
- Git2effort
- SLOCCount
- Pandas
- Sqlalchemy
- Matplotlib

- git
- build_essential
- xhtml2pdf

A.2. Conexión al servidor

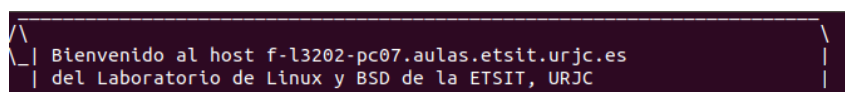
Debido a que las bases de datos se encuentran alojadas en los servidores de la universidad Rey Juan Carlos es importante que estemos conectados a su red, ya que el direccionamiento de las bases de datos es privado, y por lo tanto, si lo hacemos en local o desde otra red, no vamos a poder utilizar la herramienta. Para ello, será tan simple como activar una sesión SSH:



```
tfg@tfg-VirtualBox: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
tfg@tfg-VirtualBox:~$ ssh nombre_usuario@f-l3202-pc07.aulas.etsit.urjc.es
```

Figura A.1: Sentencia SSH.

Si hemos introducido correctamente las credenciales y el equipo al que nos queremos conectar, y todo ha ido bien, deberíamos visualizar el siguiente mensaje de bienvenida:



```
^  
| Bienvenido al host f-l3202-pc07.aulas.etsit.urjc.es  
| del Laboratorio de Linux y BSD de la ETSIT, URJC
```

Figura A.2: Sesión SSH iniciada.

Este mensaje en la pantalla de comandos nos indica que ya estamos conectados a los servidores de la universidad.

Si por el contrario, recibimos algún mensaje de error:

- Revisar que la sentencia SSH está correctamente escrita.
- Comprobar nuestra conexión a Internet.
- Consultar el parte de guerra de los laboratorios de la ETSIT. Podemos acceder a él desde el siguiente enlace <https://labs.etsit.urjc.es/index.php/parte-de-guerra/>.

- En última instancia, ponernos en contacto con el administrador de los servidores de la universidad por correo electrónico.

A.3. Ejecución

El modo de uso de la herramienta, se describe a continuación:

```
Usage: .py mysql_file.txt repos_list.txt threshold_file.txt
```

Como se puede apreciar en la sentencia de ejecución, se pasan tres ficheros como argumentos:

- Fichero de credenciales de la cuenta de MYSQL. Es necesario para realizar la conexión a la base de datos. Debemos escribirlo utilizando el siguiente formato:

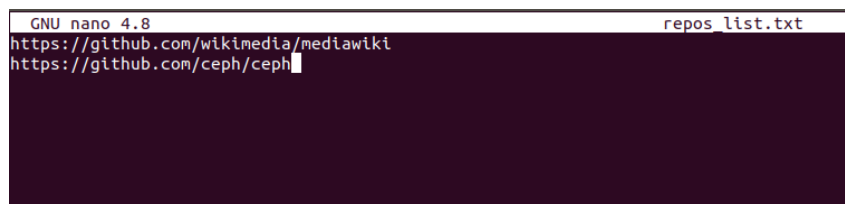


```
GNU nano 4.8 mysql_credentials.txt
uid=xxxx
pass=xxxx
ddb=xxxx
```

Figura A.3: Fichero MYSQL.

Como se muestra en el fichero `mysql_credentials.txt`, en las dos primeras líneas se definen el `uid`, que corresponde con el nombre de usuario, y el `pass`, que corresponde con la contraseña. En la tercera debe ir el nombre de la base de datos.

- Fichero de repositorios. Este archivo contiene las URLs de aquellos repositorios que queramos analizar. El formato que sigue se muestra a continuación:



```
GNU nano 4.8 repos_list.txt
https://github.com/wikimedia/mediawiki
https://github.com/ceph/ceph
```

Figura A.4: Fichero de repositorios.

Tiene que aparecer una URL por línea en el fichero.

- Fichero de thresholds. Este es el fichero que contiene los valores umbrales, por cada repositorio, que pasamos como parámetro a Git2effort para realizar la estimación:

```
GNU nano 4.8                               new_ts
moodle=14
webkit=17
ceph=24
mediawiki=36
```

Figura A.5: Fichero de thresholds.

Una vez tengamos los archivos mencionados, al lanzar el script, comenzaremos a observar cada uno de los pasos que va llevando a cabo:

```
Leyendo repositorios...
Lectura completada.
Creando diccionario de repostorios...
Diccionario listo.
Analizando repositorio: mediawiki...
Creando directorio para volcar los resultados...
Directorio creado: /tmp/mediawiki_results
Procesando tablas...
```

Figura A.6: Ejecución del script.

Cuando el programa finaliza, si todo ha salido bien, deberíamos visualizar el último mensaje de la ejecución del script:

```
Análisis del repositorio mediawiki completado.
vmtorres@f-l3202-pc07:~/Escritorio$
```

Figura A.7: Fin de la ejecución del script.

A.4. Base de datos

Adicionalmente, si queremos comprobar que todo ha ido bien durante la ejecución del script, a pesar de los mensajes de finalización, podemos consultar si se han almacenado los datos adecuadamente. Para poder verificar el almacenamiento, accedemos al siguiente enlace <https://labs.etsit.urjc.es/mysql/index.php>. Primero tenemos que introducir nuestras credenciales:



Figura A.8: Inicio sesión MYSQL.

Una vez hecho el logging, podemos verificar si se han almacenado las tablas de datos de los repositorios. Para ello, nos dirigimos a la base de datos:

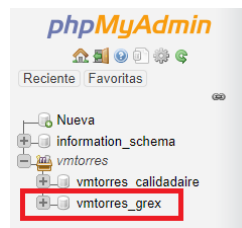


Figura A.9: Selección de la BBDD.

Desplegamos las tablas de la base de datos y comprobamos si se ha almacenado lo que esperábamos:

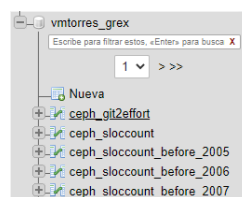


Figura A.10: Tablas de la BBDD.

A.5. Resultados

Finalmente, se exportan los resultados de los análisis realizados. El software devuelve cuatro ficheros: dos ficheros HTML y dos ficheros PDF. Estos archivos contienen todos los resultados que se muestran en el capítulo 5. Se encuentran alojados en el directorio:

```
/tmp/repositorio_results.
```



Figura A.11: Ficheros de salida.

Bibliografía

- [1] B. W. Boehm. *Software Engineering Economics*. Prentice Hall, Englewood Cliffs, NJ, 1981.
- [2] S. Dueñas, V. Cosentino, G. Robles, and J. M. Gonzalez-Barahona. Perceval: Software project data at your will. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings*, pages 1–4, 2018.
- [3] C. Kemerer. An empirical valuation of software cost estimation models. *Commun. ACM*, 30:416–429, 05 1987.
- [4] V. Kumari. Software development cost estimation methods and particle swarm optimization model. 04 2019.
- [5] D. K. Moulla. Cocomo model for software based on open source: Application to the adaptation of triade to the university system. *International Journal on Computer Science and Engineering*, 5, 06 2013.
- [6] S. Rajper and Z. Shaikh. Software development cost estimation: A survey. *Indian Journal of Science and Technology*, 9, 08 2016.
- [7] G. Robles, J. M. González-Barahona, C. Cervigón, A. Capiluppi, and D. Izquierdo-Cortázar. Estimating development effort in free/open source software projects by mining software repositories: a case study of openstack. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 222–231, 2014.
- [8] D. A. Wheeler. More than a gigabuck: Estimating GNU/Linux’s size, June 2001. <https://dwheeler.com/sloc/redhat71-v1/redhat71sloc.html>.