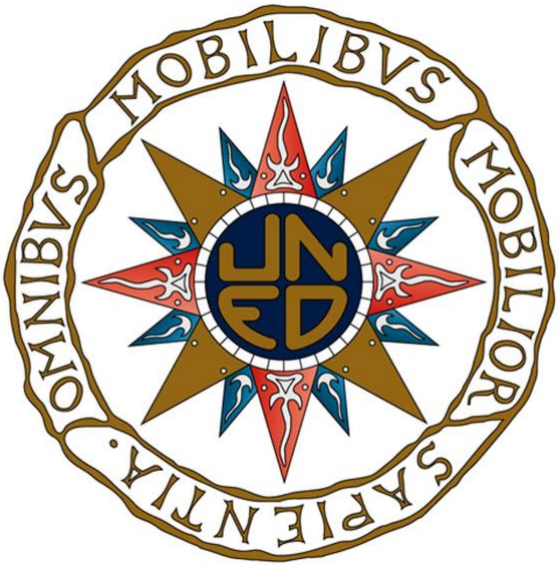
Máster en Investigación en Ingeniería de Software y Sistemas Informáticos
Itinerario de Ingeniería de Software
Código de la asignatura 31105151

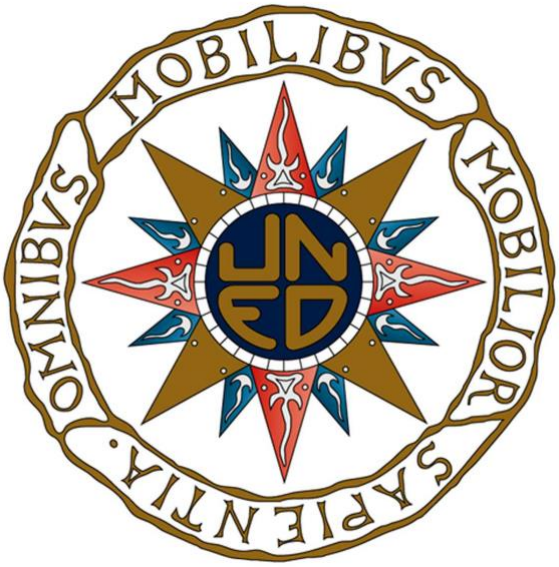
Estudio y mejora de un modelo de evaluación de calidad de proyectos de software libre



Autor: Samuel Iglesias Gonsálvez
Directora: Dra. Elena Ruiz Larrocha
Curso académico 2017-2018
Convocatoria de junio 2018

Máster en Investigación en Ingeniería de Software y Sistemas Informáticos
Itinerario de Ingeniería de Software
Código de la asignatura 31105151

Estudio y mejora de un modelo de evaluación de calidad de proyectos de software libre



Tipo de Trabajo B: Trabajo específico propuesto por el alumno
Autor: Samuel Iglesias Gonsálvez
Directora: Dra. Elena Ruiz Larrocha


UNED

UNED

DECLARACIÓN JURADA DE AUTORÍA DEL TRABAJO CIENTÍFICO, PARA LA DEFENSA DEL TRABAJO FIN DE MASTER

Fecha: 01/06/2018

Quién suscribe:

Autor(a): Samuel Iglesias Gonsálvez
D.N.I/N.I.E/Pasaporte.: 71650240-L

Hace constar que es la autor(a) del trabajo:

Estudio y mejora de un modelo de evaluación de calidad de proyectos de software libre

En tal sentido, manifiesto la originalidad de la conceptualización del trabajo, interpretación de datos y la elaboración de las conclusiones, dejando establecido que aquellos aportes intelectuales de otros autores, se han referenciado debidamente en el texto de dicho trabajo.

DECLARACIÓN:

- ✓ Garantizo que el trabajo que remito es un documento original y no ha sido publicado, total ni parcialmente por otros autores, en soporte papel ni en formato digital.
- ✓ Certifico que he contribuido directamente al contenido intelectual de este manuscrito, a la génesis y análisis de sus datos, por lo cual estoy en condiciones de hacerme públicamente responsable de él.
- ✓ No he incurrido en fraude científico, plagio o vicios de autoría; en caso contrario, aceptaré las medidas disciplinarias sancionadoras que correspondan.

Fdo.

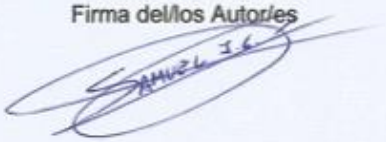
Impreso TFdM05_Autor. Autorización de publicación

y difusión del TFdM para fines académicos

Autorización

Autorizo/amos a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del/los Autor/es



Juan del Rosal, 16
28040, Madrid

Tel: 91 398 89 10
Fax: 91 398 89 09

Resumen

En prácticamente todas las empresas se usa software desarrollado por terceros como apoyo para realizar alguna tarea. Este software puede estar instalado tanto en forma de aplicación como en forma de librería, en el caso de que se use como una dependencia para desarrollar software propio.

La selección de qué software escoger para cada tarea es una tarea ardua y compleja, donde no sólo hay que centrarse en el coste en sí, sino tener en cuenta varios factores como el soporte, mantenibilidad, funcionalidad, usabilidad, disponibilidad de proveedores, etcétera. En general, se selecciona un software entre los más populares que se ofrecen la funcionalidad requerida, ya sean bajo licencia propietaria o bajo licencia libre. No obstante, hay una serie de preguntas que normalmente se hacen cuando se están valorando entre distintas opciones: ¿qué criterios usar? ¿se deben escoger en base a criterios económicos? Es decir, ¿escoger el que tenga el menor precio en su licencia de uso? ¿o el más barato de mantener? Si nos centramos en el proveedor, ¿el que realiza una empresa multinacional o el realizado por una comunidad de desarrolladores? Si nos fijamos en su licencia y soporte, ¿escogeríamos el que nos ofrece más flexibilidad debido a su licencia libre o al que se pueden contratar licencias de uso con un soporte de varios años? Dependiendo del caso, muchas de estas preguntas sólo pueden ser respondidas por un experto en la materia. Tanto si se dispone de esa persona experta en la empresa como si no, un modelo de evaluación de calidad del software puede ayudar a tomar decisiones, ya que están diseñados para obtener una calificación medible, reproducible y comparable entre distintas alternativas software.

Integración de modelos de madurez de capacidades o *Capability Maturity Model Integration* (CMMI) es un modelo para la mejora y evaluación de procesos para el desarrollo, mantenimiento y operación de sistemas de software, cuyo uso está ampliamente extendido en organizaciones y empresas de software. Sin embargo, este modelo se centra en los procesos más que en las características del software en sí. Hay otros modelos diseñados con este último objetivo como son ISO-9126, ISO-25000 y otros. Desgraciadamente, en el caso de software desarrollado bajo licencia libre, ninguno de estos modelos es totalmente aplicable directamente ya que un proyecto de software libre tiene una comunidad de usuarios y desarrolladores alrededor que afecta, de manera notable, en la calidad del mismo dependiendo de sus características. Por lo tanto, los modelos clásicos de evaluación de calidad tienen que ser adaptados de alguna manera para cubrir este hecho.

Este trabajo fin de máster se centrará en realizar un estudio de los principales modelos de evaluación de calidad de software libre, se buscarán sus fortalezas y debilidades y se propondrá un nuevo modelo que busque solucionar estas últimas. Finalmente, se aplicará este modelo a un proyecto de software libre popular para comprobar que puede ser utilizado en el mundo real.

Palabras clave: Métodos de evaluación de calidad de software, Software Libre, Desarrollo de Software, Ingeniería de Software.

Índice

Lista de Figuras

Lista de Tablas

1. Introducción

Dentro de la infraestructura IT de prácticamente cualquier empresa, ya sea desarrolladora de software o no, llega un momento que tiene que depender de librerías y herramientas software desarrolladas por terceros como parte de la solución realizada para un cliente o como parte de las aplicaciones usadas internamente en su actividad diaria. Entre todas las opciones disponibles en el mercado, ¿cómo escoger una u otra, si todas parecen iguales a primera vista?, ¿qué proveedor nos ofrece mayor confianza de que su software será una garantía de éxito si escogemos su software en vez de el de la competencia? Estas y otras muchas cuestiones no son fáciles de resolver en la mayoría de los casos sin la ayuda de un experto en la materia.

No obstante, poder tener una evaluación objetiva y que pueda ser comparable para distintas librerías y herramientas software, se ha trabajado desde hace años en la creación de modelos de evaluación de calidad de software, los cuales ofrecen un marco de trabajo adecuado mediante la definición de las métricas a obtener, qué se persigue obtener de ellas y cuál es un valor bueno con respecto a otro.

Uno de los modelos de evaluación más famosos es el de Integración de modelos de madurez de capacidades o *Capability Maturity Model Integration* (CMMI) que, pese a que está centrado en la madurez de procesos más que en las características del software, suele ser un requerimiento para realizar trabajos para la administración pública la obtención de la certificación de un nivel de madurez mínimo, ya que se considera un sello de calidad en la realización de los proyectos para las mismas y permite gestionar de manera adecuada y ordenada los procesos realizados. No obstante, si se busca evaluar las características y propiedades del software en sí, existen modelos basados como son ISO 9126 e ISO 25010 que cubren esta necesidad. Estos modelos evalúan de manera objetiva la funcionalidad del software, su mantenibilidad, usabilidad, eficiencia, portabilidad… entre otros.

Sin embargo, en los proyectos de software libre hay otros indicadores de calidad que no están relacionados directamente con el código fuente o con la gestión de procesos, sino con la calidad de la propia comunidad de usuarios y desarrolladores formada alrededor del proyecto de software libre: ¿es la comunidad heterogénea? ¿el conocimiento se concentra en unas pocas personas? ¿está desarrollado por una única empresa que puede dejar de hacerlo? ¿Hay herramientas que faciliten la comunicación y la coordinación? ¿La licencia permite reusar el código para cualquier actividad? Las respuestas a estas y otras cuestiones son indicadores de la salud de un proyecto de software libre, lo cual es muy importante para cualquier empresa que quiera seleccionar uno para utilizar en sus proyectos de desarrollo de software, en su propia infraestructura tecnológica o para utilizar como herramientas en su actividad diaria.

Como se verá en el capítulo 2, en investigaciones previas se ha estudiado cada una de las particularidades de un proyecto bajo licencia de software libre u *open-source* y se han propuesto modelos de evaluación de los mismos que satisfagan dichos requerimientos. Muchos de estos modelos empezaron basándose en los ya existentes, ya sean modelos genéricos como CMMI-DEV, ISO 9126 o ISO 25010, o en otros modelos centrados en

software libre anteriores, pero han ido evolucionando proporcionando nuevas características y métricas a analizar. No obstante, prácticamente todos los modelos centrados en el software libre analizados han sido abandonados tras una vida corta de varios años.

En la literatura hay un acuerdo que todavía no se ha desarrollado un modelo "perfecto" que sea ampliamente adoptado [1], que sea lo suficientemente sencillo para ser aplicado fácilmente y lo suficientemente robusto para no verse comprometido por la subjetividad de las personas evaluadoras. En varios artículos se han hecho comparaciones [1]–[6] y, en algunos casos, incluso se indican las características que un modelo debe tener para solucionar los problemas de los modelos existentes [1], [10]. Se va a emplear estos trabajos previos como punto de partida para este trabajo final de máster.

El objetivo de este trabajo fin de máster es proponer un nuevo modelo que intente cubrir las debilidades de los modelos ya existentes y se aplicará sobre un proyecto de software libre conocido como una prueba de concepto de su aplicación. Para ello, se realizará una comparativa en el capítulo 2 mediante el uso de la metodología FOCOSEM [7], donde se analizarán los modelos de evaluación de calidad para desarrollos de software libre existentes, se obtendrán como resultado las debilidades de los mismos y dónde hay margen de mejora, para aplicarlo en el modelo propuesto en este trabajo final de máster.

1.1 Objetivos del TFM

Este trabajo fin de máster se centrará en realizar un estudio de los distintos modelos de evaluación de calidad de proyectos de software libre y proponer un nuevo modelo con el objetivo de solucionar las debilidades encontradas en los mismos.

Los objetivos son:

- Analizar cada una de las metodologías de evaluación de proyectos de software libre más populares entre las existentes en la literatura.
- Realizar un estudio comparativo de dichas metodologías y buscar las cuestiones no resueltas en ellas.
- En base a los resultados del estudio anterior, proponer un nuevo modelo de evaluación de calidad de proyectos de software libre que cubra las debilidades de los ya existentes.
- Aplicar el modelo propuesto sobre un proyecto conocido de software libre y comprobar cómo se comportaría en un caso real.

1.2 Estructura del TFM

El trabajo fin de máster se divide en cuatro capítulos, describiéndose a continuación un breve resumen del contenido presente en cada uno de ellos.

En el primer capítulo se realiza la introducción del trabajo fin de máster, incluyéndose los objetivos que se persiguen en la realización del mismo y se describe la estructura del propio documento.

En el segundo capítulo se realiza un análisis del estado de la cuestión, donde se describen los distintos trabajos, artículos y estudios realizados hasta el día de hoy que sean relevantes en la definición de modelos de evaluación de calidad para el software libre.

En el tercer capítulo se describe la resolución del problema planteado donde se describe de manera detallada el nuevo modelo propuesto y se realizará un caso de uso sobre un proyecto de software libre popular.

En el cuarto capítulo se presentan las conclusiones resultantes de la realización del trabajo fin de máster y se plantean las futuras líneas de investigación.

Finalmente, se finaliza este documento con un par de capítulos al final del trabajo: el primero de ellos describe la bibliografía empleada y el segundo incluye el glosario de términos empleados durante la realización de este trabajo fin de máster.

2. Estado del arte

Se puede definir un producto software [8] como "un componente software empaquetado o un servicio basado en software que podría tener componentes auxiliares y que se publica y se intercambia en un específico mercado. De esta manera los componentes empaquetados se refieren a todo tipo de software. El producto software puede tomar la forma de: pequeño, componentes software comerciales *off-the-shelf*, software empaquetado, software comercial, *open-source* software y servicios.".

La investigación sobre la evaluación de calidad de productos software no es algo reciente, aunque sí ha ido avanzando de manera notable en los últimos años. Hace más de treinta años se empezó a estudiar cómo evaluar la calidad del software con modelos básicos como son el de Boehm [11] y el de Mc Call [12]. Desde entonces se ha ido evolucionando a modelos más complejos y que tengan en cuenta más aspectos del código, el proceso de desarrollo que se sigue y otros muchos aspectos que afectan a la calidad del software.

En este capítulo se revisarán brevemente los modelos de evaluación de calidad de software más famosos en la literatura del área, empezando por los modelos genéricos y aplicables a prácticamente cualquier tipo de software, para servir como retrospectiva y referencia a los modelos diseñados para evaluar productos de software libre, que serán descritos a continuación. Estos los modelos de evaluación de calidad de productos de software libre y *open source* tienen en cuenta aspectos y características específicos del software libre, siendo la mayoría relacionados con la comunidad de desarrolladores y usuarios que hay alrededor de estos productos software.

Finalmente, se realizará una comparativa entre los modelos de evaluación de calidad de proyectos de software libre, describiendo las características de cada uno, sus fortalezas y sus debilidades, para finalizar explicando las áreas donde hay margen de mejora.

Los modelos escogidos fueron a partir de realizar búsquedas por "Open Source Quality model" y "Open Source Evaluation model" en Google Scholar y Microsoft Academic. Seleccionando aquellos modelos que suelen ser descritos en comparativas y repetidamente citados en la literatura encontrada con estas palabras claves.

2.1 Modelos de evaluación de calidad de software

2.1.1 Modelos de Boehm y Mc Call

El modelo de Mc Call [12] se creó en el año 1977 como un informe para el ejército del aire de los Estados Unidos de América y accesible públicamente. Este modelo establece la calidad del producto software en base a una serie de factores agrupados en 3 actividades del producto software: *Project Review* (facilidad de mantenimiento, flexibilidad y testeo), *Product Operation* (fiabilidad, eficiencia, integridad, usabilidad y si es correcto), *Product Transition* (portabilidad, reusabilidad e interoperabilidad), como se puede observar en la Ilustración 1.

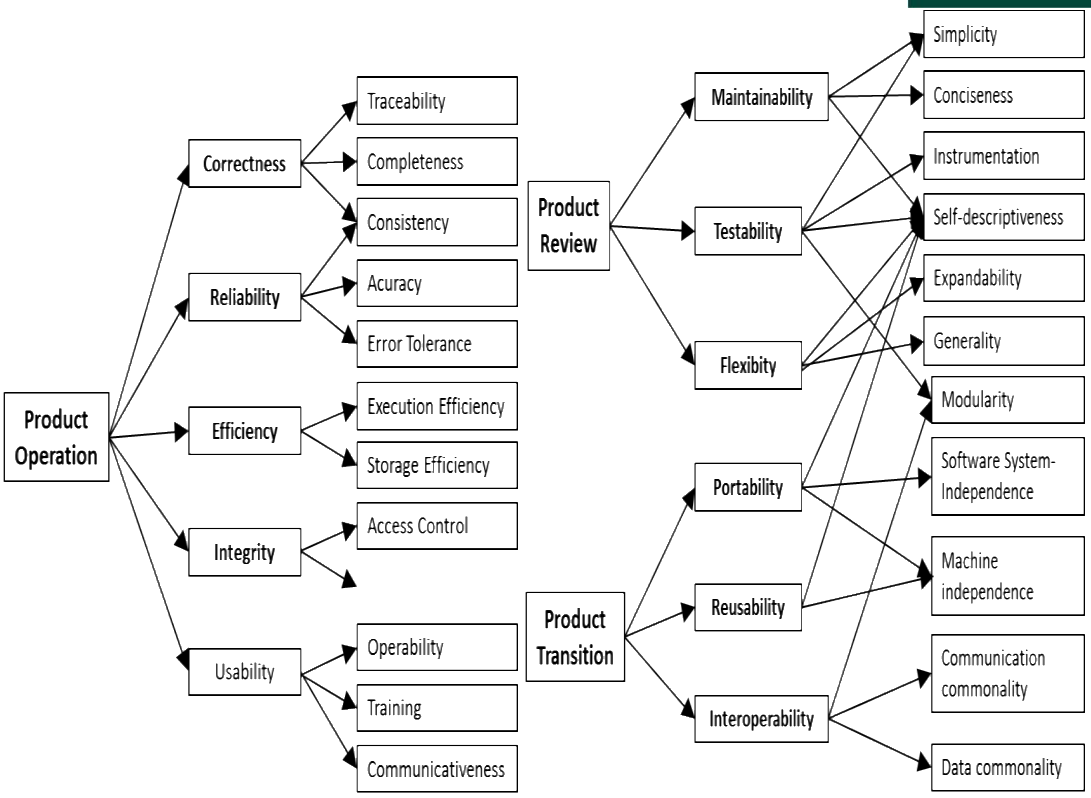


Ilustración 1: Modelo de calidad de Mc Call. Gráfica obtenida de Miguel et al [8]

Dependiendo de la aplicación, se dará más peso a algunos de los factores frente a otros. Por ejemplo, en una aplicación que se use en un avión, se requerirá que el factor de fiabilidad sea crítico y, por ello, tendrá muchísimo peso dentro de la evaluación. De la misma manera, se aplicarán pesos a cada uno de los factores para adecuarlo a las necesidades del software. Dentro del informe de Mc Call [12], se ponen ejemplos de qué factores son más críticos para cada uno de los perfiles típicos de las aplicaciones del ejército del aire de los Estados Unidos de América. Finalmente, se definen métricas para cada uno de los factores del modelo.

El modelo de Boehm [11] fue creado en 1976. Este modelo incorpora varios factores que añaden nuevos puntos de vista frente al modelo de Mc Call, interesantes a tener en cuenta en diferentes niveles. Boehm define una serie de factores de alto nivel como: Utilidad (facilidad, fiabilidad y eficiencia de uso en un producto software); facilidad de mantenimiento en forma de maneras de modificación, testeabilidad y los aspectos de entendimiento del código; portabilidad (facilidades para cambiar de entorno).

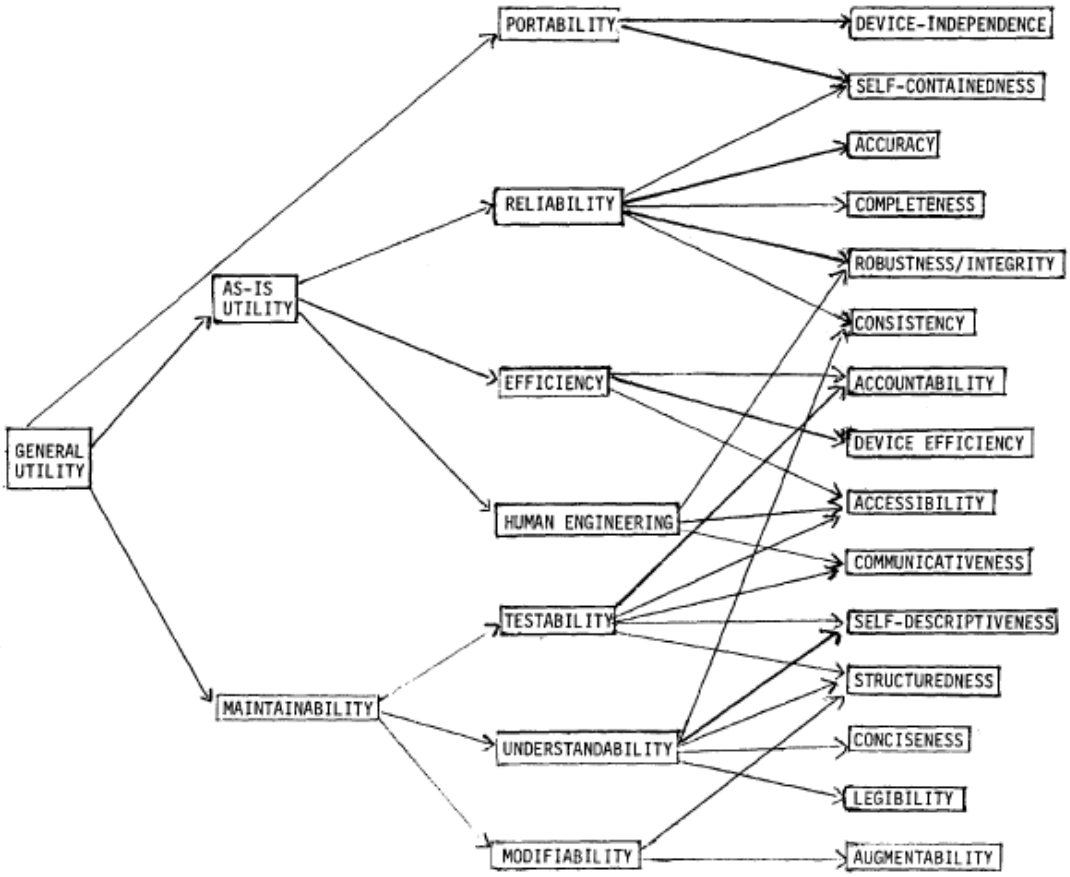


Ilustración 2: Modelo de Boehm. Gráfica sacada de Boehm [11].

Para cada uno de los factores, Boehm define métricas y las clasifica entre las que se pueden evaluar de manera automática y si al ser evaluadas con una herramienta obtienen resultados concluyentes o no.

2.1.2 Modelo CMMI

Tan importante como la evaluación de las características del software es la evaluación de la madurez de los procesos seguidos para crear dicho software. Uno de los modelos más utilizados hoy en día para cubrir esta necesidad es *Capability Maturity Model Integration* (CMMI) [13][14], que es un modelo creado por el Instituto de Ingeniería del Software (SEI) de Carnegie Mellon y financiado por la Oficina de Secretaría de Defensa y la Asociación Nacional de Defensa Industrial de Estados Unidos.

CMMI es una evolución de un modelo anterior llamado CMM (*Capability Maturity Model*). CMMI está formado por varios modelos que se centran en áreas específicas: CMMI para desarrollo (CMMI-DEV), CMMI para adquisición, CMMI para servicios, etc. CMMI se basa en niveles de capacidad y niveles de madurez donde se evalúan las actividades realizadas para cada uno de los procesos relevantes para el área analizada. Por ejemplo, para el caso de CMMI-DEV existen 22 áreas de proceso:

- Análisis Causal y Resolución (CAR).

- Gestión de Configuración (CM).
- Análisis de Decisiones y Resolución (DAR).
- Gestión Integrada del Proyecto (IPM).
- Medición y Análisis (MA).
- Definición de Procesos de la Organización (OPD).
- Enfoque en Procesos de la Organización (OPF).
- Gestión del Rendimiento de la Organización (OPM).
- Rendimiento de Procesos de la Organización (OPP).
- Formación en la Organización (OT).
- Integración del Producto (PI).
- Monitorización y Control del Proyecto (PMC).
- Planificación del Proyecto (PP).
- Aseguramiento de la Calidad del Proceso y del Producto (PPQA). • Gestión Cuantitativa del Proyecto (QPM).
- Desarrollo de Requisitos (RD).
- Gestión de Requisitos (REQM).
- Gestión de Riesgos (RSKM).
- Gestión de Acuerdos con Proveedores (SAM).
- Solución Técnica (TS).
- Validación (VAL).
- Verificación (VER).

Si se realizan estas actividades, se gestionan de manera adecuada, están definidas, administradas apropiadamente o son optimizadas podríamos alcanzar un determinado nivel de madurez o de capacidad dependiendo de hasta qué punto se realizan cada una de ellas.

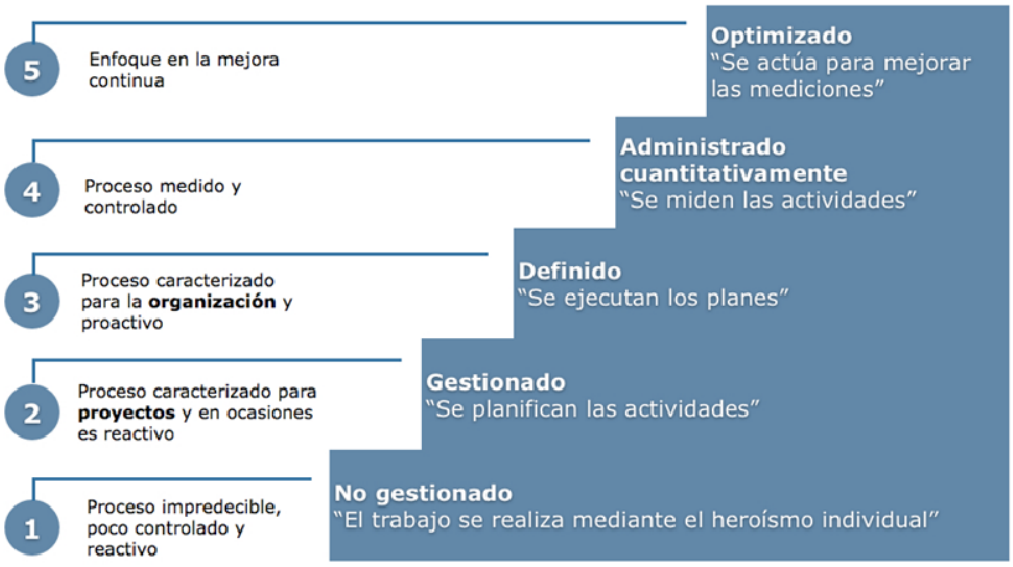


Ilustración 3: Esquema de los niveles de madurez CMMI.

Los niveles de capacidad se refieren a la consecución de la mejora de procesos de una organización en área de proceso individuales. Mientras que los niveles de madurez se refieren a la consecución de la mejora de procesos en múltiples áreas de proceso [14].

Nivel	*Representación continua* *Niveles de capacidad*	*Representación por etapas* *Niveles de madurez*
Nivel 0	Incompleto	
Nivel 1	Realizado	Inicial
Nivel 2	Gestionado	Gestionado
Nivel 3	Definido	Definido
Nivel 4		Gestionado cuantitativamente
Nivel 5		En optimización

Ilustración 4: Comparación de los niveles de capacidad y de madurez. Tabla obtenida de [14].

Para obtener un determinado nivel de madurez, se deben de realizar todas las actividades relacionadas en las áreas de proceso marcadas por la tabla en la Ilustración 5. No obstante, se puede obtener el mismo nivel de madurez mediante la consecución de niveles de capacidad en las actividades de cada una de las áreas de proceso que definen el perfil objetivo para dicho nivel de madurez.
En cualquiera de los dos casos, para conseguir la certificación CMMI-DEV para un determinado nivel de madurez, se debe de realizar una evaluación por parte de un experto certificador externo donde se definirán las áreas a mejorar (si las hubiera), se mejorarían las mismas y, tras varias iteraciones, se conseguiría finalmente la certificación CMMI-DEV para el nivel deseado.

No obstante, una empresa podría querer evaluar su nivel de CMMI-DEV de manera interna para mejorar sus propios procesos internos para desarrollar un software de calidad, incluso combinándolo con metodologías ágiles [15] si fuera necesario o, en el caso de

software desarrollado por terceros, se podría evaluar para comprobar la calidad del mismo.

Área de Proceso	*Categoría*	*Nivel de Madurez*
Análisis Causal y Resolución (CAR)	Soporte	5
Gestión de Configuración (CM)	Soporte	2
Análisis de Decisiones y Resolución (DAR)	Soporte	3
Gestión Integrada del Proyecto (IPM)	Gestión de proyectos	3
Medición y Análisis (MA)	Soporte	2
Definición de Procesos de la Organización (OPD)	Gestión de procesos	3
Enfoque en Procesos de la Organización (OPF)	Gestión de procesos	3
Gestión del Rendimiento de la Organización (OPM)	Gestión de procesos	5
Rendimiento de Procesos de la Organización (OPP)	Gestión de procesos	4
Formación en la Organización (OT)	Gestión de procesos	3
Integración del Producto (PI)	Ingeniería	3
Monitorización y Control del Proyecto (PMC)	Gestión de proyectos	2
Planificación del Proyecto (PP)	Gestión de proyectos	2
Aseguramiento de la Calidad del Proceso y del Producto (PPQA)	Soporte	2
Gestión Cuantitativa del Proyecto (QPM)	Gestión de proyectos	4
Desarrollo de Requisitos (RD)	Ingeniería	3
Gestión de Requisitos (REQM)	Gestión de proyectos	2
Gestión de Riesgos (RSKM)	Gestión de proyectos	3
Gestión de Acuerdos con Proveedores (SAM)	Gestión de proyectos	2
Solución Técnica (TS)	Ingeniería	3
Validación (VAL)	Ingeniería	3
Verificación (VER)	Ingeniería	3

Ilustración 5: Áreas de proceso, categorías y niveles de madurez. Tabla obtenida de [14].

2.1.3 Modelo ISO-9126

Sin embargo, CMMI-DEV no es el único modelo ampliamente utilizado. Existen otros estándares internacionales como ISO-9126 y, el más reciente, ISO-25000 que fueron definidos para evaluar la calidad del software basándose en sus propiedades y que complementan la evaluación realizada por CMMI-DEV, la cual se centra en la evaluación de madurez de procesos tal y como se ha visto en el apartado anterior.

El modelo ISO 9126 está basado en los modelos de Boehm [11] y McCall [12], descritos previamente. El modelo se divide en dos partes bien diferenciadas:

- Los atributos de calidad interna y externa.
- La calidad en atributos de uso.

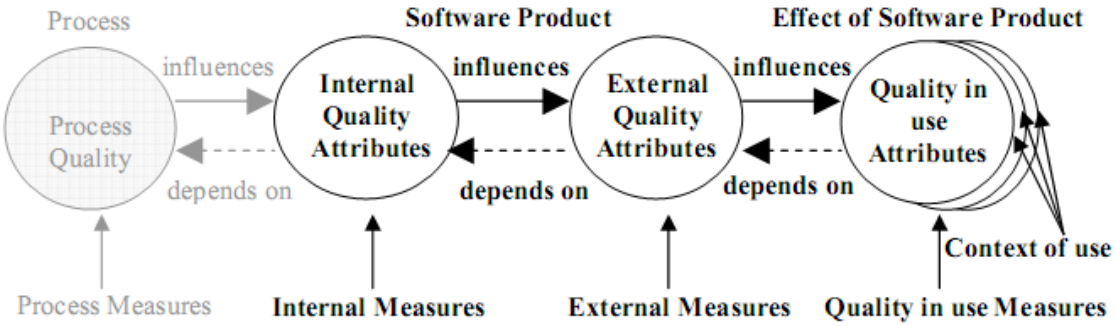


Ilustración 6: Calidad en el ciclo de vida de ISO 9126. Gráfica sacada de Miguel et al [8].

Los atributos de calidad internos se refieren a propiedades del sistema que deben ser evaluadas sin ejecutarse, mientras que los atributos externos se refieren a propiedades que deben evaluadas mientras se ejecuta el programa. Estas propiedades son las que experimentan los usuarios en la ejecución del sistema y durante su mantenimiento. A continuación se listan los atributos de calidad internos y externos junto con los sub-atributos que los componen (también mostrados en la Ilustración 7).

- Funcionalidad: capacidad del software de proveer los servicios necesarios para cumplir con los requisitos funcionales.
    - Idoneidad: se refiere si el software desempeña las tareas para las cuales fue desarrollado.
    - Exactitud: este atributo evalúa el resultado final que realiza la aplicación y si tiene consistencia a lo que se espera.
    - Interoperabilidad: se comprueba si el sistema puede interactuar con otro sistema independiente.
    - Seguridad: se verifica el control de acceso a usuarios.
- Fiabilidad: capacidad del software de mantener las prestaciones requeridas del sistema, durante un tiempo establecido y bajo un conjunto de condiciones definidas.
    - Madurez: se verifica los fallos y errores del sistema y si han sido corregidas en el tiempo de pruebas o uso del sistema.
    - Recuperabilidad: se comprueba si el software puede volver a funcionar correctamente y restaurar datos perdidos después de un fallo ocasional.
    - Tolerancia a fallos: se verifica si la aplicación desarrollada es capaz de manejar correctamente errores en el sistema.
- Usabilidad: esfuerzo requerido por el usuario para utilizar el producto satisfactoriamente.
    - Aprendizaje: se comprueba si el usuario puede aprender fácilmente a usar el sistema.
    - Comprensión: se verifica si el usuario puede comprender el funcionamiento del sistema
    - Operatividad: se resuelve si el usuario puede utilizar el sistema sin mucho esfuerzo.

- Atractivo: se comprueba que la interfaz de la aplicación sea atractiva para el usuario.

- Eficiencia: relación entre las prestaciones del software y los requisitos necesarios para su utilización.
  - Comportamiento en el tiempo: se verifica la rapidez con la que responde la aplicación.
  - Comportamiento de recurso: se determina si el sistema utiliza los recursos de manera eficiente o no.
- Mantenibilidad: esfuerzo necesario para adaptarse a las nuevas especificaciones y requisitos del software.
  - Estabilidad: se comprueba si el sistema puede mantener su funcionamiento a pesar de realizar cambios en el mismo.
  - Facilidad de análisis: se comprueba si el software puede ser analizable para poder diagnosticar fácilmente los errores.
  - Facilidad de cambio: se analiza si el sistema puede ser fácilmente modificado.
  - Facilidad de pruebas: se evalúa si el sistema puede ser probado fácilmente.
- Portabilidad: capacidad del software ser transferido de un entorno a otro.
  - Capacidad de instalación: se comprueba la facilidad de instalación del software.
  - Capacidad de reemplazamiento: se evalúa si el software puede remplazar otro software similar de manera sencilla.
  - Adaptabilidad: se verifica que el software se emplear en otros entornos.
  - Co-existencia: se comprueba si el software puede funcionar con otros sistemas.

Cada una de las características debe ser evaluada dentro del software basándonos en pruebas de funcionamiento, medición de rendimiento y pruebas con usuarios que harán uso del sistema.

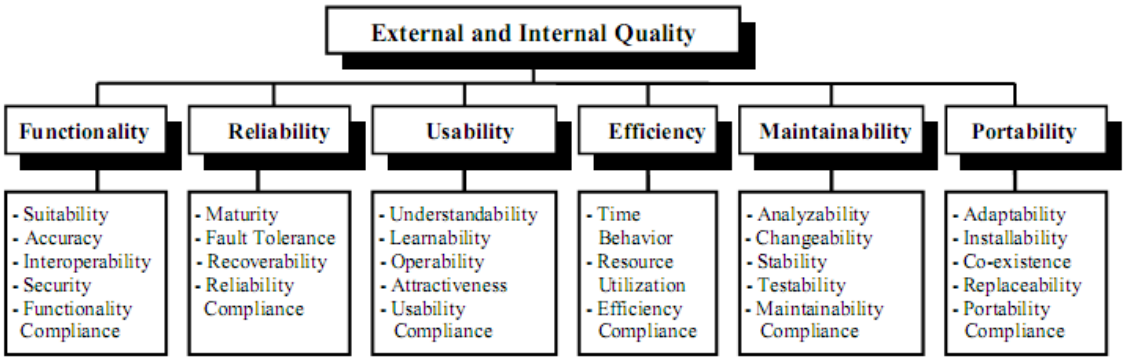


Ilustración 7: Atributos de calidad externa e interna de ISO 9126. Gráfica sacada de Miguel et al [8].

Por otra parte, los atributos de uso se refieren a la efectividad del producto, su productividad, su seguridad y la satisfacción de los usuarios.

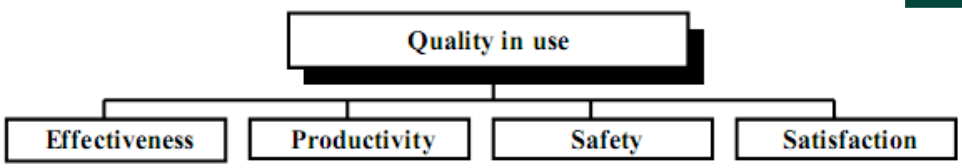


Ilustración 8: Atributos de uso del modelo ISO 9126. Gráfica sacada de Miguel et al [8].

2.1.4 Modelo ISO-25010

La norma ISO 25000 proporciona una especificación para poder usar los estándares internacionales llamados requisitos y Evaluación de Calidad de Productos Software (SQuaRE). La norma establece criterios para la especificación de requisitos de calidad de productos software, sus métricas y su evaluación. También incluye un modelo de calidad para unificar las definiciones de calidad de los clientes con los atributos en el proceso de desarrollo. El objetivo general de la creación del estándar ISO 25000 SQuaRE fue organizar, enriquecer y unificar las series que cubren dos procesos principales: especificación de requerimientos de calidad del software y evaluación de la calidad del software, soportada por el proceso de medición de calidad del software. A continuación, se muestran cada una de las divisiones que forman la norma ISO 25000.

- ISO 2500n. División de gestión de calidad. Los estándares que forman esta división definen todos los modelos comunes, términos y referencias que se usan en el resto de las divisiones de SQuaRE.
- ISO 2501n. División del modelo de calidad. El estándar que conforma esta división presenta un modelo de calidad detallado, incluyendo características para la calidad interna, externa y en uso.
- ISO 2502n. División de mediciones de calidad. Los estándares que pertenecen a esta división incluyen un modelo de referencia de calidad del producto software, definiciones matemáticas de las métricas de calidad y una guía práctica para su aplicación. También incorpora aplicaciones de métricas para la calidad de software interna, externa y en uso.
- ISO 25030. División de requisitos de calidad. Los estándares que forman parte de esta división ayudan a especificar los requisitos de calidad. Estos requisitos pueden ser usados en el proceso de especificación de requisitos de calidad para un producto software que va a ser desarrollado o como entrada para un proceso de evaluación. El proceso de definición de requisitos se guía por el establecido en la norma anterior llamada ISO 15288.
- ISO 25040. División de evaluación de la calidad. Estos estándares proporcionan requisitos, recomendaciones y guías para la evaluación de un producto software, tanto si la llevan a cabo evaluadores, como clientes o desarrolladores.
- ISO 25050–25099. Estándares de extensión SQuaRE. Incluyen requisitos para la calidad de productos de software “Off-The-Self” y para el formato común de la industria (CIF) para informes de usabilidad.

La división del modelo de calidad ISO 25010 fue creada como una actualización del modelo ISO 9126 para adecuarlo a las nuevas necesidades que fueron surgiendo y que éste no cubría. Este modelo considera como nuevas características la seguridad y la compatibilidad, la cual agrupa algunas características de portabilidad de ISO 9126 que no formaban parte de manera lógica en la transferencia de un entorno a otro. En este estándar, se usa el término transferibilidad como una extensión de la portabilidad.

Este modelo tiene varias características de calidad del producto software que luego se subdividen en sub-características (ver Ilustración 9):

- Adecuación funcional.
- Eficiencia de desempeño.
- Compatibilidad.
- Usabilidad.
- Fiabilidad.
- Seguridad.
- Mantenibilidad.
- Portabilidad

Ilustración 9: Modelo ISO 25010. Gráfica sacada de iso25000.com.

También hay características para medir la calidad en uso como son:

- Efectividad.
- Eficiencia.
- Satisfacción del usuario.

Mediante el uso de todos estos estándares, se define todo el proceso de evaluación de software, sus características, sub-características y las métricas asociadas a cada una de ellas.

2.1.5 Otros modelos

Hay muchos más modelos de evaluación de la calidad del software en la literatura, como por ejemplo el de Dromey [16]. No obstante, en este trabajo fin de máster nos centraremos en los más utilizados en la industria y los más populares en la literatura que trata modelos de evaluación de calidad de software.

2.2 Modelos de evaluación de software libre

Pese a su gran popularidad para evaluar la calidad del software, tanto CMMI-DEV como ISO 9126 e ISO 25010 no consiguen evaluar totalmente los proyectos de software libre pese a que pueden ser aplicados [10]. Tanto ISO 9126, como ISO 25010 pueden servir de modelo estándar para analizar software libre en términos de calidad de producto y calidad de uso. Sin embargo, estos modelos no tienen en cuenta las características únicas de los proyectos de software libre tales como la comunidad que hay alrededor de ellos.

Ya se ha analizado previamente en otros estudios que la calidad de la comunidad afecta a la calidad de un proyecto de software libre [17]. Las principales características relacionadas con la comunidad de un proyecto de software libre son capacidad de mantenimiento, sostenibilidad y madurez de proceso [18].

La capacidad de mantenimiento se refiere al número de contribuidores a un proyecto de software libre y durante cuánto tiempo contribuyen al esfuerzo de desarrollo, siendo sus métricas recogidas de los registros en los repositorios de control de versiones, listas de correo, foros de discusión y sistemas de reporte de errores. La sostenibilidad se refiere a la habilidad de la comunidad para crecer en términos de nuevos contribuidores y la regeneración de la comunidad mediante el reemplazo de contribuidores que abandonan el proyecto por nuevos desarrolladores. La madurez de proceso se refiere a la adopción y el uso de prácticas estándares en el proceso de desarrollo como el envío y la revisión de cambios, revisión de cambios entre pares, la disponibilidad de una suite de *testing* y realizar el lanzamiento de nuevas versiones del software de una manera planificada.

En este apartado, vamos a describir los modelos más populares en la literatura para luego realizar una comparativa entre ellos.

2.2.1 Capgemini Open Source Maturity Model

El modelo Open Source Maturity Model (OSMM) de Capgemini fue creado por esta consultora en el año 2003 para evaluar software libre, pero el método tiene una licencia propietaria y se requiere de una autorización expresa en caso de que se quiera utilizar por terceros.

Para la evaluación de un producto de software libre bajo este modelo, se emplean veintisiete indicadores, siendo doce de ellos los indicadores de productos usados para determinar la madurez de un producto de software libre. Estos indicadores se agrupan en cuatro diferentes grupos:

- El producto (Edad, licencias, las jerarquías humanas, puntos de venta, comunidad de desarrolladores).
- Integración (Modularidad, colaboración con otros productos, estándar).
- Utilización (Soporte, la facilidad de implementación).
- Aceptación (Comunidad de usuarios, penetración en el mercado).

Los quince indicadores restantes, son los Indicadores de Aplicaciones empleados para determinar la adaptación de las aplicaciones libres a las necesidades de los clientes: Usabilidad, Interfaz, Rendimiento, Fiabilidad, Seguridad, Tecnología probada,

Independencia de proveedor, Independencia de la plataforma, Soporte, Reporte, Administración, Asesoramiento, Capacitación, Dotación de personal, Implementación.

Se califica cada uno de los indicadores en una escala de 1 a 5 donde 5 es "extremadamente importante" y 1 "no importante". Todos estos datos de calificación se combinan en una calificación final que indica la idoneidad del producto para las demandas determinadas. Finalmente, determinar una única puntuación solo permite una fácil comparación entre productos de candidatos.

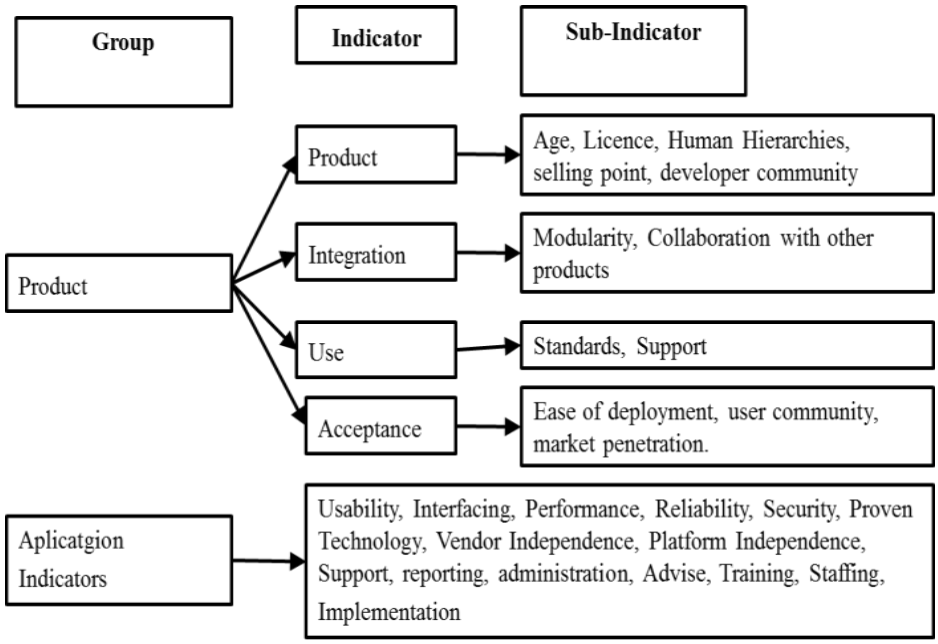


Ilustración 10: Modelo OSMM de Capgemini. Gráfica sacada de Miguel et al [8].

Tabla 1: Indicadores para el grupo de Producto en el modelo OSMM de Capgemini.

Grupo	Indicador	Inmaduro	Maduro	Puntuación
Producto	Edad	El proyecto acaba de empezar	El proyecto ha sido activo por algún tiempo	1 a 5
	Licencia	No clara o no descrita	Comercial y uso por Sistema Operativo	1 a 5
	Jerarquías humanas	Un solo desarrollador líder o unos pocos	Múltiples líderes	1 a 5
	Puntos atractivos	Sólo entusiasmo	Características comerciales como seguridad o mantenibilidad	1 a 5
	Comunidad de desarrolladores	Grupo pequeño	Comunidad activa de desarrolladores	1 a 5
Integración	Modularidad	No tiene módulos	Arquitectura d módulos o plugins	1 a 5
	Colaboración con otros productos	No es una prioridad	Atención para integrarse con otros productos	1 a 5

Uso	Estándares	No cumple con estándares	Cumple con estándares actuales	1 a 5
	Soporte	Limitado por la propia comunidad	Soporte altamente accesible	1 a 5
Aceptación	Facilidad de implementación	No hay formación ni cursos	Hay formación y cursos disponibles	1 a 5
	Comunidad de usuario	Pequeño grupo	Grande y bien organizada	1 a 5
	Penetración de mercado	Unas pocas referencias	Muchas referencias con casos de aplicación	1 a 5

Tabla 2: Indicadores de aplicación del modelo OSMM de Capgemini.

Indicador	Propósito	Prioridad (P)	Puntuación (S)	Puntuación ponderada
Usabilidad	Experiencia de usuario	1 a 5	1 a 5	P x S
Interfaz	Relación entre los estándares de producto y los requerimientos de conectividad de la organización	1 a 5	1 a 5	P x S
Rendimiento	Capacidad de procesado y carga de trabajo esperada por los usuarios	1 a 5	1 a 5	P x S
Fiabilidad	Nivel de disponibilidad	1 a 5	1 a 5	P x S
Seguridad	Evaluación de seguridad	1 a 5	1 a 5	P x S
Tecnología probada	La tecnología ha sido probada en entornos de producción	1 a 5	1 a 5	P x S
Independencia de vendedor	Nivel de compromiso entre cliente y vendedor	1 a 5	1 a 5	P x S
Independencia de plataforma	Independencia de plataforma	1 a 5	1 a 5	P x S
Soporte	Nivel de soporte requerido	1 a 5	1 a 5	P x S
Reporte	Reportes realizados por área funcional	1 a 5	1 a 5	P x S
Administración	Administración de operación, parches, cambios y mantenimiento	1 a 5	1 a 5	P x S
Consejo	Validación o recomendación por terceras partes independientes	1 a 5	1 a 5	P x S
Formación	Formación y cursos	1 a 5	1 a 5	P x S
Expertos	Expertos contratables, o que pueden enseñar.	1 a 5	1 a 5	P x S
Implementación	Escenarios de implementación	1 a 5	1 a 5	P x S

2.2.2 Navica Open Source Maturity Model

Por otro lado, existe el modelo OSMM de Navica, el cual no se debe de confundir con el de Capgemini pese a la similitud entre sus nombres. OSMM de Navica fue diseñado para ayudar a los managers del departamento IT de cualquier empresa a comparar y evaluar software bajo licencia libre. Este modelo fue creado en el año 2004. El modelo OSMM de Navica determina la madurez de un producto libre en tres fases bien diferenciadas:

1. Selección del software que se va a evaluar.

2. Definición de factores de ponderación. OSMM divide los principales aspectos de evaluación del software en seis categorías: Productos de Software, Soporte, Documentación, Formación, Integración de productos y Servicios profesionales. Se proporciona una lista de calificación con valores de ponderación propuestos por defecto por el propio modelo. Los valores por defecto pueden ser cambiados de acuerdo a las necesidades específicas, pero debe mantenerse la suma total de 10.
3. Cálculo de la calificación general de madurez del producto. OSMM asigna una ponderación a la calificación de madurez de cada elemento permitiendo que cada elemento refleje su importancia a la madurez global del producto. La calificación ponderada de cada elemento se suma a proporcionar una calificación general de la madurez del producto.

Productos de Software	4
Soporte	2
Documentación	1
Formación	1
Integración de productos	1
Servicios profesionales	1
Total	**10**

Tabla 3: Factores de ponderación por defecto para cada categoría de OSMM Navica.

El modelo se suministra bajo la licencia *Academic Free License*, lo cual permite ser utilizado sin necesidad comprar ninguna licencia.

2.2.3 OpenBRR

El modelo *Open Business Readiness Rating* (OpenBRR) [19] es una metodología de evaluación de software open-source creada en 2005 por Carnegie Mellon Silicon Valley junto con SpikeSource, O'Really e Intel. Esta metodología se centra en integrar las limitaciones de las compañías (principalmente tests y fiabilidad), y busca en compartir y reducir el coste total de propiedad del software libre.

En la definición de este modelo se ha tomado en cuenta el modelo OSMM de Capgemini y el modelo ISO 9126, gracias a los cuales, este modelo identifica varias categorías que son relevantes para evaluar software libre. El modelo tiene siete categorías, cada una con subcategorías que pueden ser retocadas para afinarse y para cubrir aspectos que no fueran considerados en un nivel superior.

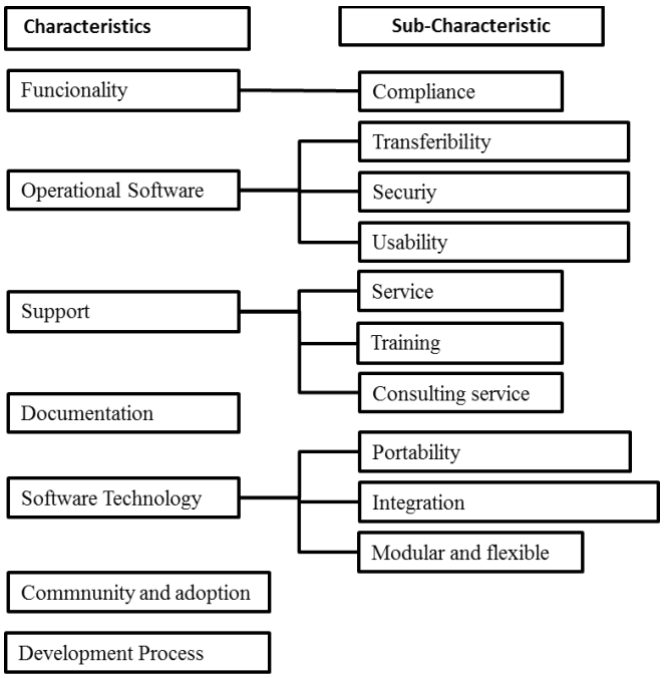


Ilustración 11: Modelo OpenBRR. Gráfica obtenida de Miguel et al [8].

La metodología OpenBRR hace los siguientes pasos:

i. Realizar una evaluación rápida. Este paso empieza con una lista de proyectos de software libre cuyo producto software encada en los requerimientos para, posteriormente, escoger los candidatos para la selección final. Este filtrado inicial está basado en la criticidad del producto o sistema que va a integrar el componente libre y tiene en cuenta su viabilidad en base al contexto.

ii. Se personaliza la plantilla de evaluación. Este paso consiste en revisar y seleccionar el criterio de evaluación a partir de la jerarquía propuesta por OpenBRR.

iii. Recolección de datos y procesado. Este paso se aplica sobre los proyectos de software libre escogidos en el primer punto y consiste en recoger los datos necesarios para puntuar el software con los baremos definidos en el segundo paso, aplicándose los factores de ponderación asociados a cada métrica que fueron determinados en dicho paso.

iv. Traducción de datos. Se agregan las puntuaciones de cada una de las métricas y se obtiene una puntuación para cada categoría. A continuación, se ajusta la puntuación de cada categoría mediante el uso de factores de ponderación determinados igualmente en el segundo punto.

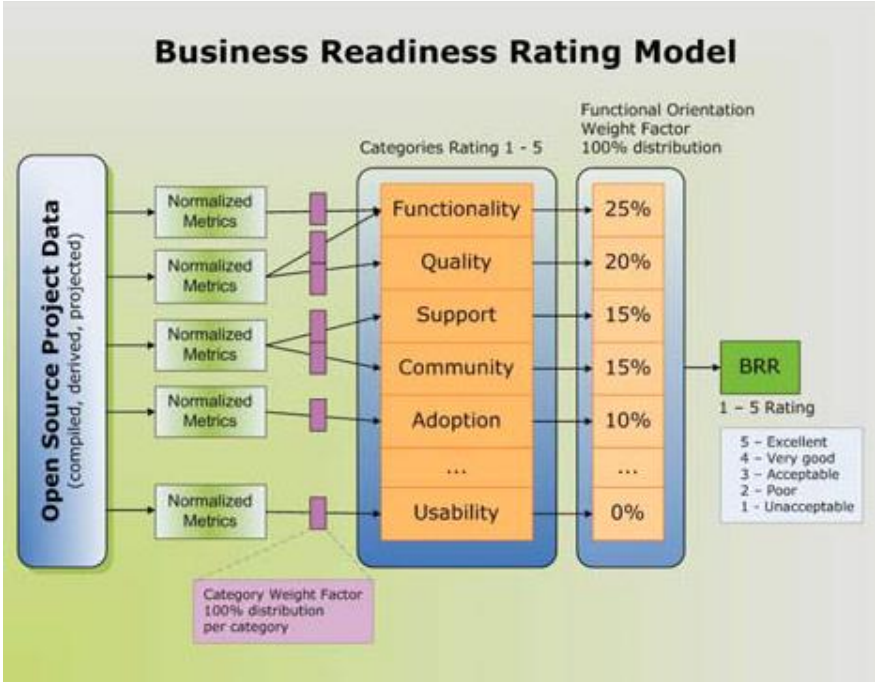


Ilustración 12: Modelo BRR, imagen sacada de [19].

El resultado final de realizar todos estos pasos será la valoración que puede usarse como comparación o para evaluar un software determinado. Normalmente, la evaluación se guarda en una hoja de cálcula proporcionada por el proyecto, pero la página web ha sido cerrada siendo sólo accesible mediante el sitio web archive.org.

2.2.4 QSOS

El modelo *Qualification and Selection of Open Source software* (QSOS) [20] es una metodología para evaluar software libre creada en 2004 por la compañía Atos Origin. Este modelo está licenciado bajo una licencia libre llamada *GNU Free Documentation License*.

Hay una serie de criterios genéricos establecidos por QSOS: Durabilidad Intrínseca, Solución Industrializada, Adaptabilidad Técnica, Estrategia. Cada uno de ellos tiene subcategorías, al igual que otros modelos.

QSOS define una serie de pasos a realizar para evaluar un conjunto de alternativas software:

- Obtener una lista de proyectos FLOSS cuyos productos software encajan en una serie de requerimientos.
- Evaluar cada uno de los proyectos bajo el criterio de evaluación proporcionado por QSOS. Este paso asignará una puntuación global para cada criterio de evaluación.
- El evaluador podrá ajustar la importancia de cada criterio basado en el contexto. Este paso se realiza mediante factores de ponderación y estableciendo umbrales para cada criterio.
- Las puntuaciones obtenidas en el segundo paso son ponderadas mediante los factores de ponderación establecidos por el punto anterior. La salida de este paso es una

clasificación que determinada si un software es válido o no para los requerimientos del cliente.

- QSOS sugiere que, como paso final, se prueben los proyectos FLOSS con mayor puntuación que cumplan el criterio de selección. El número de software escogidos puede variar entre 2-5 productos FLOSS dependiendo de lo crítica que sea esta decisión.

Además de la jerarquía que se muestra en la Ilustración 13, QSOS define un procedimiento para puntuar cada criterio. La puntuación válida del modelo tiene valores desde 0 a 2, siendo cero que no se realiza dicho criterio o que se realiza de manera muy pobre, a 2 que se realiza de manera excelente.

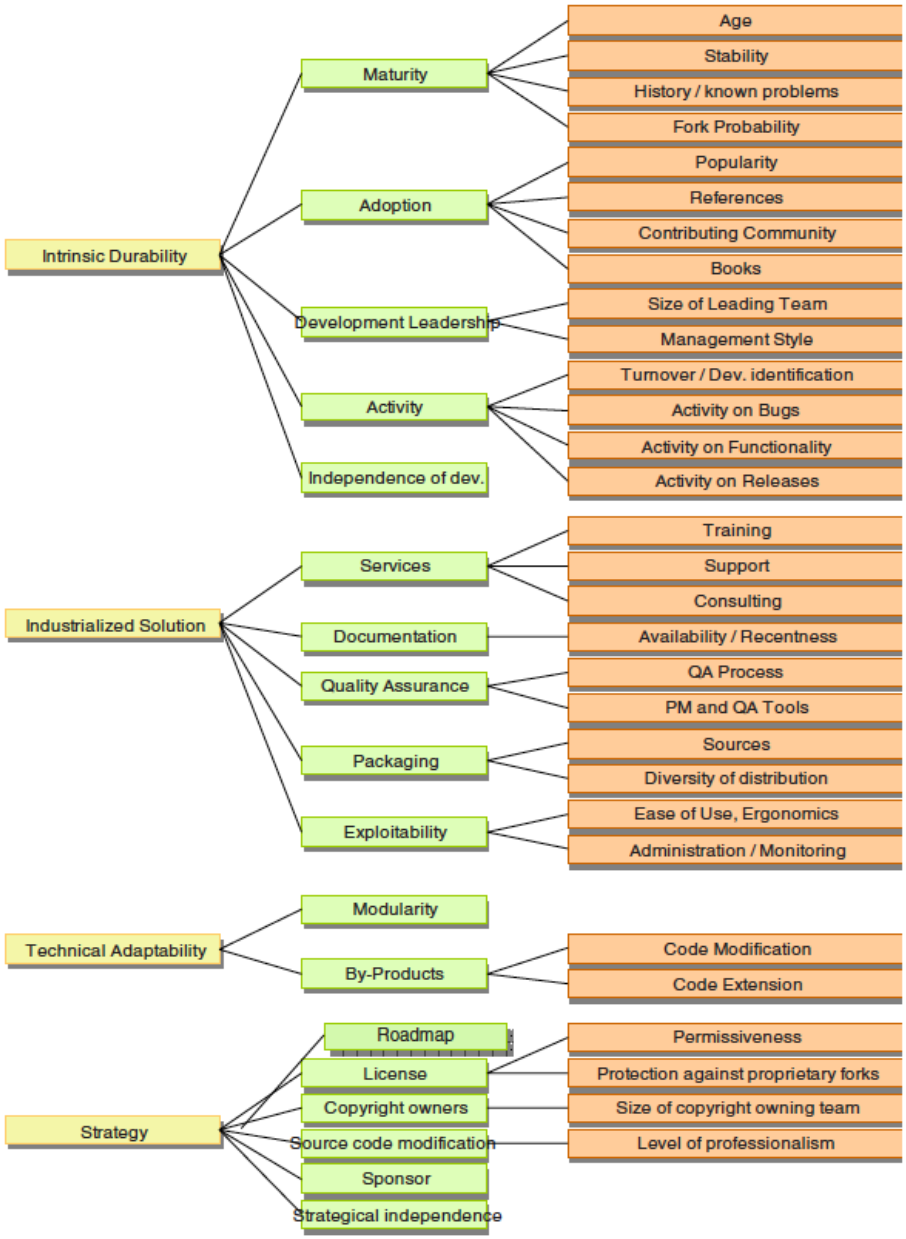


Ilustración 13: Criterios genéricos de QSOS. Imagen sacada de [2].

2.2.5 QualOSS

QualOSS es una metodología para medir la calidad del software libre cuyos criterios principales son la robustez y la facilidad de evolución de los proyectos de software libre [18], [21] creado como proyecto de la Unión Europea. Al contrario que otras metodologías, QualOSS considera que tanto las características del producto, las características de la comunidad como las características de los procesos software son igualmente importantes (ver Ilustración 14) para saber la calidad de un proyecto de software libre.

Cabe destacar que la realización de un análisis de cómo se realizan los procesos software dentro del proyecto de software libre es una de las características que aporta QualOSS y que lo diferencia con sus predecesores. No es una sorpresa que QualOSS se inspire en las métricas de CMMI-DEV para definir las suyas, ya que no hay diferencias notables con proyectos de código propietarios.

QualOSS está formado por tres tipos de elementos que están interrelacionados: características de calidad, métricas e indicadores. Las características de calidad son los atributos de un producto o comunidad que se considera relevante para su evaluación. Las métricas son los aspectos concretos que podemos medir de un producto o en la comunidad asociada al mismo; se espera que estas métricas tengan una correlación con las características de calidad que se van a evaluar. Finalmente, los indicadores interpretan una serie de valores de medida relacionados con las características de calidad, es decir, definen cómo agregar y evaluar los valores medidos para poder obtener un valor consolidado que permita ser comparable y que las personas que decidan en base a la evaluación tengan la información necesaria para hacerlo correctamente.

QualOSS asume que la calidad de un proyecto depende enormemente del contexto al que se vaya a usar y de los propósitos de la empresa que quiere seleccionarlo. Por tanto, se distingue entre escenarios de uso y escenarios de negocio, ya que hay atributos que cobran más importancia en un caso y otros en el contrario.

Para el caso de los escenarios de uso, se usa el enfoque tradicional de comparar entre varias opciones y escoger la más adecuada o, en el caso de una autoevaluación, se comprueba la robustez y adaptabilidad del proyecto software como escenario de introspección.

En el caso de los escenarios de negocio, éstos son determinados por la manera que la empresa que examina la dirección de un proyecto software, se enfrenta a dicha dirección. Algunos ejemplos de ello serían si la empresa está dispuesta a colaborar totalmente y mandar parches a la comunidad para adaptarlo a sus necesidades, si quiere crear una nueva versión del mismo (*fork*), si pretende construir una solución propietaria encima, vender servicios sobre ello; dependiendo del caso se dará más importancia a un aspecto u a otro. Gracias a que QualOSS depende del contexto que la empresa quiera usar un software, la evaluación se adapta a dicho contexto y ofrece una perspectiva distinta: el punto de vista del mánager de producto, el punto de vista del mánager del proyecto y el punto de vista del arquitecto, analista y desarrollador.

La evaluación de QualOSS se realiza de manera automatizada en su mayoría, minimizando cualquier subjetividad que pudiera ser introducida por un ser humano. Sin embargo, no es posible evitar realizar alguna evaluación manual para determinadas medidas, como por ejemplo las relacionadas con documentación. Los resultados finales se documentan en una hoja de cálculo que se rellena tanto automáticamente (por las herramientas) como manualmente, por lo que todos los valores obtenidos estarían recogidos en un mismo documento.

Los resultados se muestran en una escala de colores, basándose en las medidas calculadas mediante umbrales, las cuales distinguen los diferentes niveles de riesgo para un negocio que somete a escrutinio un determinado proyecto de software libre. La falta de información necesaria para un determinado punto se interpreta como un riesgo alto, ya que QualOSS interpreta que estamos tratando con una caja negra con un grado importante de incertidumbre con todo lo que ello implica. Los colores van desde el verde (riesgo despreciable), amarillo (riesgo pequeño), rojo (riesgo medio) y negro (riesgo alto).

Al igual que en otras metodologías, los umbrales para seleccionar un color de la escala se seleccionan mediante principalmente gracias a la experiencia del evaluador más que en un análisis estadístico. Pese a que puede considerarse un sistema arbitrario, esto permite ajustar los umbrales a los requerimientos específicos que se precisen para cada caso de uso.

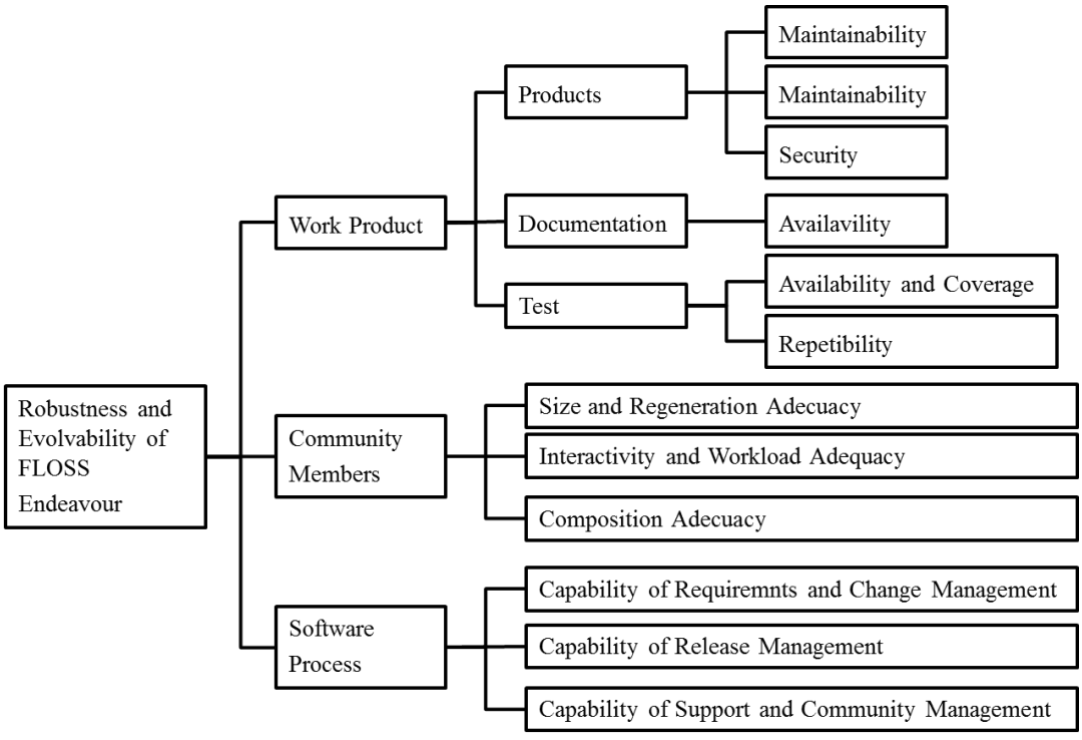


Ilustración 14: Modelo QualOSS. Gráfica obtenida de Miguel et al [8].

2.2.6 QualiSPo Open Source Maturity Model

El modelo QualiSPo fue creado como un proyecto financiado por la Unión Europea bajo el sexto programa marco para desarrollar un método de evaluación de código abierto centrado en la confianza y calidad de los sistemas de código abierto [22]. Esta metodología, basada en CMMI, está publicada bajo una licencia libre Creative Commons.

Como este modelo define todos los elementos que han de ser evaluados, así como el conjunto de normas y directrices que describe cómo llevar a cabo los procesos de evaluación, se puede considerar que es tanto un modelo como una metodología.

Al estar basado en CMMI, QualiSPo está organizado en niveles de madurez, cada nivel está basado en el nivel inferior, incluyendo sus elementos de confianza. La confianza en este modelo es una propiedad de un producto software que depende de la percepción que tienen los usuarios sobre las cualidades exhibidas por el propio producto (funcionalidad, interoperabilidad, fiabilidad, rendimiento, etcétera). Esta percepción depende de las cualidades intrínsecas del producto y depende además del tipo de uso que se haga del producto. En la Ilustración 15 se muestran algunos de los elementos de confianza que forman parte de QualiSPo.

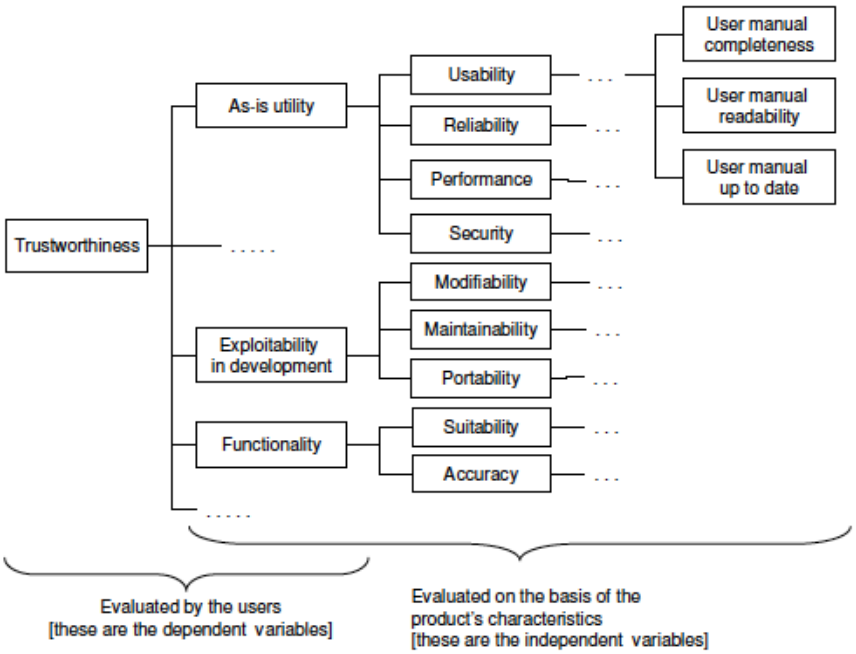


Ilustración 15: Elementos de confianza de QualiSPo. Gráfica sacada de [22].

Los usuarios tienen una percepción de las cualidades de un software basado en la confianza sobre el software. En cambio, en los modelos tradicionales como ISO 9126, se indica que la confianza en un software se basa en las propiedades intrínsecas del mismo. Por tanto, en QualiSPo se ha creado un modelo de confianza que tiene en cuenta ambos puntos de vista (ver Ilustración 16).

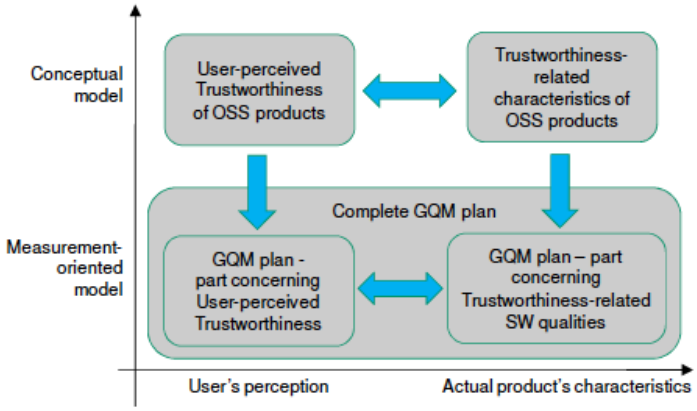


Ilustración 16: Modelos de confianza y puntos de vista de QualiSPo. Gráfica sacada de [22].

Las evaluaciones subjetivas se realizan mediante un cuestionario a los usuarios, mientras que la evaluación objetiva se realiza mediante la ejecución de herramientas software que analizan el código fuente.

A la hora de obtener la evaluación final, se tiene en cuenta los resultados de ambos métodos y se aplica el método GQM[1] ("Goal, Question, Metric") para intentar eliminar la subjetividad en la evaluación de las cualidades realizada por los usuarios y focalizarse en las medidas objetivas de las características del producto.

QualiSPo se apoya en varias herramientas que ayudan al evaluador a realizar la evaluación [22]:

- Spago4Q. Se trata de una herramienta que soporta la evaluación y la inspección de calidad de un producto software. Esta herramienta ofrece una visualización de los resultados obtenidos a través del resto de herramientas que se describen.
- MACXIM. MACXIM es una herramienta que realiza un análisis estático del código fuente para obtener sus propiedades estáticas. Se calcula hasta 70 métricas gracias a esta herramienta, incluyendo tamaño, complejidad, modularidad y otros.
- JaBUTi. JaBUTi es una herramienta que realiza tests para comprobar el flujo de control y el flujo de datos para un código fuente en lenguaje *bytecode* de Java.
- The GQM Tool. Esta herramienta implementa la metodología GQM.
- StatSVN and StatCVS. Ambas herramientas se usan para recoger las estadísticas referidas a un repositorio de software en SVN o CVS, respectivamente.
- FOSSology. Esta es una herramienta que analiza proyectos *open-source*. En este caso se usa para buscar incompatibilidades de licencias.
- JUnit. Se usa junto Spago4Q para recoger datos del estado de los informes realizados por herramientas Java como son ant y maven.
- PMD and Checkstyle. Son herramientas que analizan el código fuente escrito en Java y buscan posibles bugs, código muerto, variables no usadas, parámetros no usados o incluso métodos no usados, así como código subóptimo.

[1] https://en.wikipedia.org/wiki/GQM

Como se puede apreciar, las herramientas de análisis están muy centradas en código escrito en Java, por lo que proyectos de software libre escritos en otros lenguajes, necesitarían otro tipo de herramientas para realizar la evaluación.

2.2.7 QuESo

QuESO es una metodología para evaluar la calidad de un proyecto de software libre creado por investigadores de la Universidad Politécnica de Cataluña [23] en el año 2014. Actualmente se está en la versión 2.0 [24].

El modelo se basa en características de calidad y medidas, al igual que otros modelos anteriores. Las características de calidad están organizadas en jerarquía, las capas superiores son las dimensiones, éstas agrupan características y sub-características, algunas de éstas son basadas en las (sub)características ya definidas por QualOSS.

Hay tres dimensiones básicas donde las características se agrupan: calidad de plataforma, calidad de comunidad y calidad de la red de ecosistema. Después, éstas se subdividen en categorías y subcategorías que se muestra en la Ilustración 17.

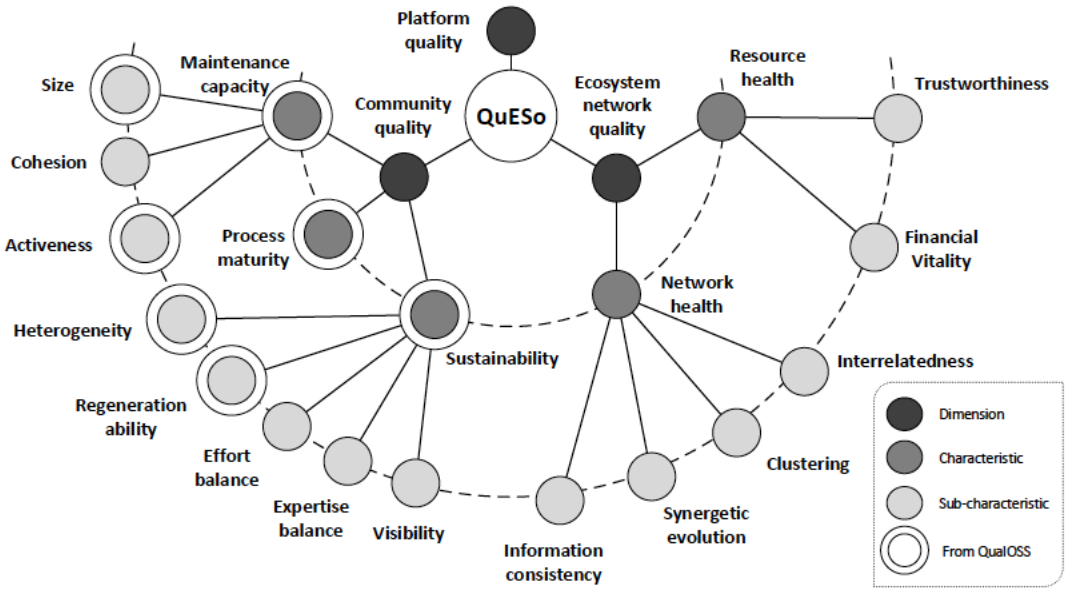


Ilustración 17: Modelo QuESO. Gráfica sacada de [23].

Las características que no están relacionadas con la calidad de la plataforma son: capacidad de mantenimiento, madurez del proceso, sostenibilidad, salud de la red y salud de recursos. Hay algunas características y subcategorías que proceden de los modelos QualOSS, ISO 25010 u otros, mientras que otras surgieron al analizar el ecosistema como un todo. Las características relacionadas con la calidad de plataforma se obtuvieron directamente de ISO 25010 donde se ofrece una buena cobertura de las mismas independiente de si el software es propietario o libre, ya que es algo que no depende de la comunidad ni de otras particularidades del software libre.

Para definir las métricas de cada una de las categorías, QuESO se basa en distintas métricas de la literatura hasta definir un total de 68 métricas. Hay métricas como son el

número de correos diarios en las listas de correo del proyecto a analizar que son fáciles de obtener de manera automatizada, mientras que otras son un poco más complejas de obtener ya sea por la dificultad de obtener los datos o porque son medidas cualitativas y no cuantitativas. No obstante, el propósito de QuESO es automatizar, en la medida de lo posible, los métodos de obtención de las métricas para evitar la subjetividad humana.

La gran virtud de QuESO es que tiene en cuenta aspectos de la comunidad que otros modelos no tienen en cuenta. Hay métricas definidas como la distribución geográfica de los desarrolladores, el número de desarrolladores principales y de *commiters* a los repositorios, si el número de contribuciones está distribuido proporcionalmente entre los desarrolladores, entre otras. Estas métricas permiten obtener una valoración más fidedigna a la realidad de un proyecto de software libre que otros modelos que sólo se centran en aspectos del código o de los procesos software.

Este modelo se describirá en detalle en el capítulo 3, ya que el modelo propuesto en este trabajo fin de máster se va a basar en QuESO.

2.2.8 SQO-OSS

El modelo SQO-OSS [17] [25] cuyas siglas significan "*Software Quality Observatory for Open Source Software*", es un modelo de calidad que se fundamenta en que la calidad y salud de un proyecto *open-source* depende de la calidad de su código fuente y de la comunidad que hay alrededor de este proyecto. Como inconveniente, SQO-OSS no mide funcionalidad, sino que se enfoca en las partes fundamentales de calidad del proyecto *open-source*, su mantenibilidad, fiabilidad y seguridad.

Para obtener un cálculo de la calidad del código fuente, se analiza el código para caracterizarlo con respecto a su mantenibilidad. La idea trata de simplificar la definición de los atributos a medir para que se puedan utilizar métricas directas, sin necesidad de combinarlas de alguna manera. Por ejemplo, para medir la mantenibilidad se utilizar la definición del modelo de calidad ISO 9126: que sea analizable, modificable, estable y que se pueda probar. Como hay varias maneras de escoger las métricas para cada uno de los atributos, se priorizaron las métricas que sean razonables y que hayan sido validadas ampliamente en la literatura del área.

A partir de estas métricas, se fue construyendo un árbol jerárquico donde la raíz es una vista aérea del modelo de calidad y los primeros dos nodos son la calidad del código fuente y la calidad de la comunidad, mientras que las hojas son las métricas propiamente dichas.

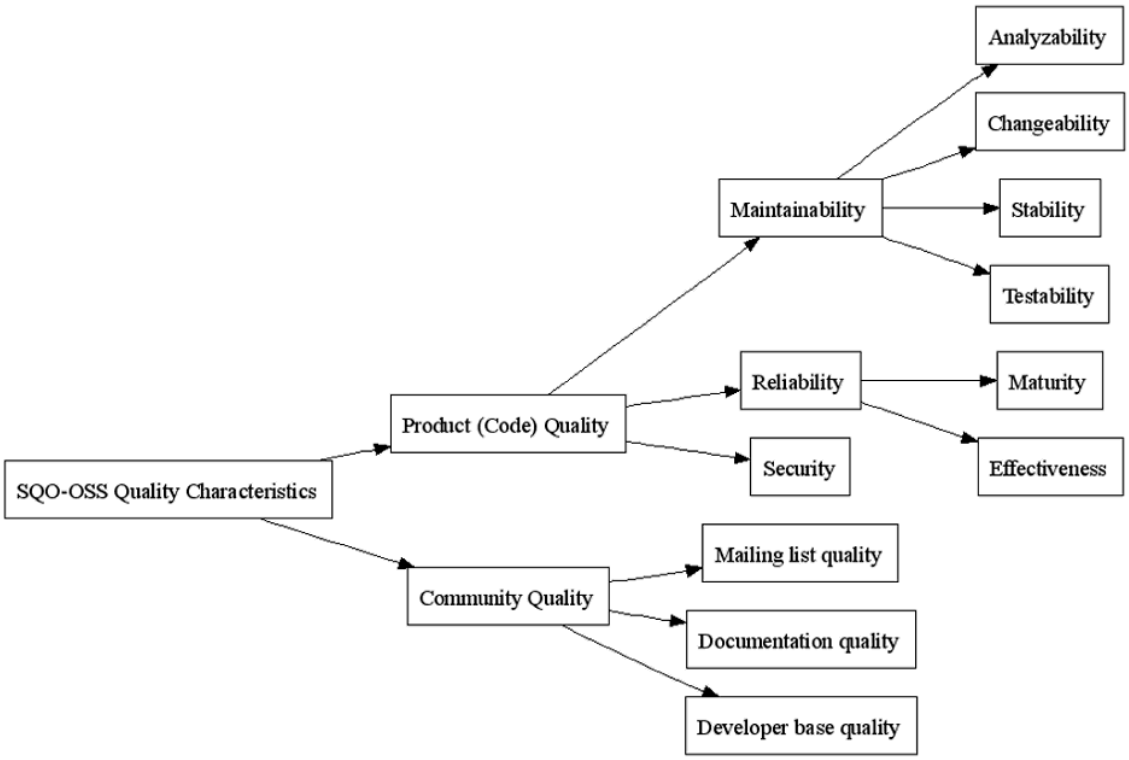


Ilustración 18: Modelo SQO-OSS. Gráfica sacada de [25].

El proceso de evaluación se basa en combinar todas las mediciones en una sola vista para obtener un resultado, es decir, se agregan los resultados de las mediciones. Para ello, se emplea un método de evaluación basado en perfiles, que permiten dar un mayor peso a ciertas métricas frente a otras dependiendo del caso que se quiera analizar.

La puntuación de cada categoría se basa en cuatro valores ordinales: Excelente, Bueno, Aceptable, Pobre; cuyos umbrales dependerán de cada perfil ya que cada categoría está interrelacionada. Estas puntuaciones facilitan el entendimiento del resultado de la evaluación al clasificarse cada uno de los valores obtenidos en ellas.

Este modelo fue diseñado desde el principio para ser usado en un sistema automático de soporte de decisiones basado en métricas. El software empresa se llama Alitheia, el cual, mediante el empleo de *plugins* para Eclipse y otras herramientas externas, se encarga de realizar estas mediciones y operar sobre ellas. De esta manera, se intenta evitar la subjetividad humana en las mediciones, siguiendo la metodología aplicada a otros modelos.

2.3 Comparación de los modelos de evaluación de software libre

Decidir qué modelo es el mejor modelo mediante una comparación, o realizar un análisis de los principales modelos para averiguar en qué se puede mejorar en este campo, es algo que se ha investigado en la literatura a lo largo de estos años [1]–[4], [8], [21].

Muchos modelos de evaluación de calidad de proyectos de software libre basan muchas de sus métricas en ISO 9126, ISO 25010 o CMMI-DEV. Este hecho se fundamenta en

que estos modelos sirven de base ya que analizan atributos y definen métricas para los mismos, que permiten obtener una estimación de la calidad de producto y calidad de uso de los proyectos de software libre. Como ISO 9126, ISO 25010 y CMMI-DEV no analizan las características propias del software libre, como es la comunidad, los modelos diseñados específicamente para proyectos de software libre analizan la comunidad que hay alrededor mediante nuevos atributos y métricas. Ya se ha analizado en otros trabajos que la calidad de la comunidad afecta a la calidad de un proyecto de software libre [17]. Las principales características relacionadas con la comunidad de un proyecto de software libre [18] son capacidad de mantenimiento, sostenibilidad y madurez de proceso.

La capacidad de mantenimiento se refiere al número de contribuidores a un proyecto de software libre y durante cuánto tiempo contribuyen al esfuerzo de desarrollo, siendo sus métricas los *logs* de los repositorios de control de versiones, listas de correo, foros de discusión y sistemas de reporte de errores. La sostenibilidad se refiere a la habilidad de la comunidad para crecer en términos de nuevos contribuidores y la regeneración de la comunidad mediante el reemplazo de contribuidores que abandonan el proyecto por nuevos desarrolladores. La madurez de proceso se refiere a la adopción y el uso de prácticas estándares en el proceso de desarrollo como el envío y la revisión de cambios, revisión de cambios entre pares, la disponibilidad de una suite de pruebas y realizar el lanzamiento de nuevas versiones del software de una manera planificada.

Para diferenciarse entre ellos, algunos modelos analizados se integran en herramientas software que realizan el análisis de una manera automatizada, mientras que otros modelos procuran analizar atributos y definir métricas que no fueron tenidas en cuenta por parte de sus competidores.

En la literatura se distinguen entre los modelos entre los llamados de primera generación y los modelos de segunda generación. Los modelos de primera generación son modelos básicos, con pocas métricas definidas para los atributos que se tienen en cuenta y que requieren que se realice la evaluación de manera manual por un experto del área, siendo éste el encargado de definir tanto los factores de ponderación como asignar los valores umbrales de las mediciones realizadas. Los inconvenientes principales de estos modelos de primera generación son que son muy sensibles a la subjetividad que pueda tener la persona evaluadora y que tienen pocas métricas, ello provoca que, si no se realizan correctamente o no se tienen suficientes datos, estas métricas afectarían de manera importante en los resultados de la evaluación. Para solucionar este problema, los modelos de segunda generación añaden muchas más métricas y aspectos a analizar, intentando automatizar, en la medida de lo posible, la obtención de valores para realizar la evaluación mediante el uso de herramientas software específicas. El inconveniente principal de los modelos de segunda generación es su opacidad en el nivel de detalle de las mediciones en las que se basa [1].

Tras analizar brevemente cada uno de los modelos, se pueden obtener una serie de características que los definen. Una de estas características serían qué modelo de evaluación de calidad de proyectos software utilizan como referencia. Los modelos que se basan en ISO 9126 son Capgemini Open Source Maturity Model, OpenBRR, QualOSS, QuESO, SQO-OSS. Por el contrario, QuESO también se basa en el modelo

ISO 25010. Por otra parte, QualiSPo está basado en CMMI-DEV, del que toma muchas métricas para realizar las evaluaciones.

A continuación, se muestra una tabla con las principales características de los modelos de calidad analizados que fueron diseñados específicamente para proyectos open-source. Esta comparativa sigue el framework de comparación de modelos de evaluación de calidad de productos de software libre llamado FOCOSEM [7].

Tabla 4: Comparación de los modelos de calidad para proyectos open-source analizados con el método FOCOSEM.

Componente	Elemento	Descripción	Capgemini OSMM	Navica OSMM	OpenBRR	QSOS	QualOSS	QualiSPo	QuESO	SQO OSS
Contexto del Método	Objetivo específico	¿Cuál es el objetivo particular del método?	Evaluar, seleccionar y comparar FLOSS de forma objetiva, trazable con argumento sostenido.	Evaluar, seleccionar y comparar FLOSS de forma objetiva, trazable con argumento sostenido.	Permitir a toda la comunidad (empresas, integradores y desarrolladores) ponderar el Software de manera abierta y estandarizada.	Evaluar, seleccionar y comparar FLOSS de forma objetiva, trazable con argumento sostenido.	Permitir la comparación de productos de software libre de forma semi-automática, sencilla y rápida.	Proporcionar una base para el desarrollo de productos de manera eficiente y confiable para las comunidades FLOSS y una base para evaluar los procesos empelados por las comunidades desarrolladoras para las empresas integradoras o integradores	Evaluar, seleccionar y comparar FLOSS de forma objetiva, trazable con argumento sostenido.	Evaluar, seleccionar y comparar FLOSS de forma objetiva, trazable con argumento sostenido.
	Evaluación de la funcionalidad	¿Es la política de funcionalidad parte del método?	No	No	Sí	No	Sí	No	Sí (calidad de plataforma basada en ISO 25010)	No
	Disponibilidad pública de resultados	¿Están las evaluaciones de los productos OSS almacenados en un repositorio público?	No	No	No	No, en repositorios. Si en la página web	Sí, en repositorios	Sí, en repositorios	No	Sí, en su momento estaban disponibles en la página web. Ahora no está disponible.
	Relación con otros modelos	¿Cómo se relaciona el modelo con otros modelos?	-	-	Capgemini OSMM y Navica OSMM	Capgemini OSMM y Navica OSMM	QSOS, OpenBRR	CMMI, OpenBRR, QSOS	QualOSS, ISO 25010	OpenBRR, QSOS
Usuario del Método	Habilidades requeridas	¿Qué habilidades necesita el usuario para usar el modelo?	Conocimiento profundo sobre cada uno de los softwares a utilizar y sobre cada detalle necesario para la evaluación, además del conocimiento general del	Conocimiento profundo sobre cada uno de los softwares a utilizar y sobre cada detalle necesario para la evaluación, además del conocimiento general del	Requiere conocimiento profundo sobre cada uno de los softwares a utilizar y sobre cada detalle necesario para la evaluación, además del conocimiento general del software	Conocimiento profundo sobre cada uno de los softwares a utilizar y sobre cada detalle necesario para la evaluación, además del	Requiere conocimientos básicos de instalación de software libre y manejo de la herramienta propia del método	Entendimiento básico de los procesos de desarrollo y la calidad del software. Conocimiento en manejo de herramientas libres y open source.	Requiere conocimientos básicos de instalación de software libre y manejo de la herramienta propia del método	Requiere conocimientos básicos de instalación de software libre y manejo de la herramienta propia del método

			software libre para generar una adecuada puntuación del mismo	software libre para generar una adecuada puntuación del mismo	libre para generar una adecuada puntuación del mismo	conocimiento general del software libre para generar una adecuada puntuación del mismo				
	Usuarios potenciales	¿Quiénes son los usuarios potenciales?	Usuarios, desarrolladores e ingenieros de IT	Usuarios, desarrolladores e ingenieros de IT	Usuarios, desarrolladores e ingenieros de IT	Usuarios, desarrolladores e ingenieros de IT	Usuarios, desarrolladores e ingenieros de IT	Usuarios, desarrolladores e ingenieros de IT	Usuarios, desarrolladores e ingenieros de IT	Usuarios, desarrolladores e ingenieros de IT
Proceso del Método	Actividades del Método	¿Cuáles son las actividades del método de evaluación?	Son 4 pasos 1 - Definición y organización de lo que será evaluado 2 - Evaluación del software 3 - Calificación de su evaluación 4 - Selección	Fase 1 - Evaluación Rápida Fase 2 – Asignar ponderación 3 – Calcular el resultado final teniendo en cuenta los datos obtenidos y sus ponderaciones	Fase 1 - Evaluación Rápida Fase 2 - Evaluación de uso objetivo Fase 3 - Recolección y procesamiento de datos Fase 4 - Traducción de datos	Son 4 pasos 1 - Definición y organización de lo que será evaluado 2 - Evaluación del software 3 - Calificación de su evaluación 4 - Selección	Son 5 pasos 1 - Inicio de la evaluación. 2 - Creación y planificación de la evaluación 3 - Recolección y procesamiento de datos 4 - Interpretación de resultados 5 - Supervisión de la evaluación	Pasos para escenario integración de los componentes de software libre: 1. Decisión a nivel de política corporativa/administración para incluir FLOSS en los productos de la empresa. 2. Proceso de selección. 3. Designación responsable para la evaluación y selección. 4.Documentacion de los requisitos/funcionalidad a ser satisfecha por el componente de FLOSS propuesto.	Son 4 pasos 1 - Definición y organización de lo que será evaluado 2 - Evaluación del software 3 - Calificación de su evaluación 4 - Selección	Fase 1 - Evaluación Rápida Fase 2 - Evaluación de uso objetivo Fase 3 - Recolección y procesamiento de datos Fase 4 - Traducción de datos
	Número de criterios	¿Cuántos criterios se usan en la evaluación?	27	-	29	41	-	12	19	53
	Categoría de Evaluación	¿Cuáles son las categorías de criterios del modelo en las que se evalúa el producto OSS?	Producto, Integración, Uso, Aceptación, Usabilidad, Interfaz, Fiabilidad, Seguridad, Tecnologías probadas, Independencia del proveedor y plataforma, soporte, reporte,	Producto, soporte y documentación, formación, servicios profesionales e integración de producto.	Funcionalidad, Usabilidad, Calidad, Seguridad, Rendimiento, Escalabilidad, Arquitectura, Soporte, Documentación, Adopción, Comunidad y Profesionalismo	Durabilidad, Solución Industrializada, Adaptabilidad Técnica, estrategia y proveedor del servicio.	Durabilidad, Solución Industrializada, Adaptabilidad Técnica, estrategia y proveedor del servicio.	Tres (3) Niveles de Madurez: Básico, Intermedio, Avanzado	Calidad de plataforma, calidad de ecosistema y red, calidad de comunidad.	Que sea analizable, modificable, testable, estabilidad, madurez, efectividad, seguridad, listas de correo, documentación, base de desarrollares

			administración, consejo, formación, implementación, y empleados							
	Resultados	¿Cuáles son los resultados del modelo?	Hoja electrónica	Ponderación en Hojas de cálculo electrónicas	Ponderación en Hojas de cálculo electrónicas	Hojas de evaluación	Ponderación en Hojas de cálculo electrónicas	Ponderación en Hojas de cálculo electrónicas	Ponderación en Hojas de cálculo electrónicas	-
	Herramientas de Apoyo	¿Se apoya en alguna herramienta?	-	-	Plantillas de evaluación	O3S (Open Source Selection Software)	Se pretende contar con una herramienta final que automatice el proceso de evaluación	Plantillas de evaluación online y offline. Herramienta de medición semiautomática está en la etapa de Prototipo.	SALMonOSS	Alitheia
Evaluación del Método	Validación	¿Ha sido validado el método de evaluación?	-	-	El modelo está probado y validado en varios proyectos de software libre.	El modelo está probado y validado en varios proyectos de software libre.	El modelo está probado y validado en varios proyectos de software libre.	El modelo está probado y validado en varios proyectos de software libre.	El modelo está probado y validado en varios proyectos de software libre.	El modelo está probado y validado en varios proyectos de software libre.
	Etapa de madurez	¿Cuál es el estado actual de madurez del modelo?	Llego a su madurez y fue abandonado.	Llego a su madurez y fue abandonado.	Llego a su madurez y fue abandonado.	Llego a su madurez y fue abandonado.	Llego a su madurez y fue abandonado.	Llego a su madurez y fue abandonado.	Maduro	Llego a su madurez y fue abandonado.

Como se puede observar, hay varios valores en la Tabla 4 que fueron omitidos por falta de información en el modelo correspondiente, principalmente por haber sido abandonado y la desaparición de los dominios web donde estaban alojados. Pese a ello, se ha recolectado gran parte de la información restante de los artículos de la bibliografía que hablan de los modelos descritos para poder completar la tabla.

Prácticamente todos los modelos de evaluación de calidad de productos de software libre y *open-source* están abandonados; con la excepción de QuESO, cuyo desarrollo sigue continuando al haber sido creado en los últimos años. Hay modelos cuyas páginas web han desaparecido y sus repositorios no tienen nuevas contribuciones desde hace años. Es precisamente este hecho el que muestra que no hay, ni hubo, un modelo de evaluación perfecto que pudiera ser usado ampliamente por la industria, algo ya pronosticado en la literatura hace años [1].

En los modelos de primera generación, se definen muchas menos métricas que los modelos de segunda generación, con lo que los resultados de sus evaluaciones están muy influidos en el caso que no se obtengan los suficientes datos. No obstante, que haya pocas métricas también implica una mayor simplicidad del modelo y que éste sea perfectamente entendible por una persona no-experta en el área. Por otra parte, como los modelos de primera generación requieren de expertos para realizar las mediciones en base a las métricas definidas, los resultados obtenidos suelen ser fiables, aunque haya una subjetividad inherente en el proceso.

En general, los valores de las mediciones y los resultados finales se presentan en forma de una hoja de cálculo electrónica, que facilita la comparación entre distintos proyectos de software. Prácticamente todos los modelos siguen unos pasos similares a la hora de realizar la evaluación: en algunos casos se define lo que se desea evaluar, posteriormente se realiza una evaluación rápida, se pondera bajo los criterios seleccionados (ya sea por el propio modelo o personalizando los pesos de ponderación por el propio evaluador o mediante el uso de perfiles de evaluación), se realiza una calificación final y se selecciona el proyecto software más adecuado para las necesidades del usuario.

Donde hay bastante diferencia entre los modelos son las características principales a evaluar. Dependiendo del modelo se evalúan aspectos como la formación, documentación, integración, fiabilidad, seguridad, comunidad… entre otros muchos aspectos.

La mayoría de los modelos analizados fueron validados con varios proyectos de software libre. Los dos modelos que no fueron validados (Capgemini y Navica OSMM), realmente es por falta de información sobre dicho modelo al haber desaparecido sus respectivas páginas web, con lo que no se descarta que hayan sido validados con proyectos de software libre.

La tendencia actual en los modelos es que éstos se basen en herramientas software para realizar sus mediciones, para así evitar la subjetividad de los evaluadores y evitar la necesidad que el evaluador sea un experto en el área. Los modelos de segunda generación (QSOS, QualOSS, SQO-OSS, QuESO, QualiSPo) se emplean herramientas software realizadas para ellos mismos que les permite obtener mediciones de manera automática sin requerir altos conocimientos en el área. No obstante, dentro de la literatura se recomienda que se tenga en cuenta el conocimiento de expertos para poder tener mayor flexibilidad en las evaluaciones y que éstas no sean basadas en medidas opacas y complejas, dando así validez y fiabilidad a los resultados obtenidos [9]. Además, las herramientas para realizar las mediciones en base a las métricas definidas en cada modelo tienen el problema que soportan un número limitado de lenguajes, como es el caso de QualiSPo que sólo soporta Java; por tanto, en algunas ocasiones no se puede realizar la evaluación de un modelo sobre un proyecto soportado, ya que su lenguaje de programación o servicios asociados (listas de correo, repositorios de software, segumiento de errores) no están soportados por las herramientas. Debido a que muchos de estos proyectos están abandonados en la actualidad, tampoco se tiene acceso a las herramientas que éstos utilizaban, provocando que se haya perdido la oportunidad de reutilizarlas para otros modelos de evaluación de calidad de proyectos de software libre.

Cabe destacar que, de todos los modelos analizados, sólo OSMM de Capgemini y QSOS tienen en cuenta las licencias de software. Además, no hay modelos que tengan en cuenta la salud de una comunidad en base a su apertura, ni diversidad, factores que podrían ser importantes como se verá posteriormente en este trabajo fin de máster. El modelo más completo en número de métricas y aspectos analizados es QuESO, lo cual no es de extrañar ya que es el modelo más moderno y tiene en cuenta todas las investigaciones realizadas en los últimos años para incorporarlas en el mismo, además de ser el único cuyo desarrollo está activo en la actualidad.

3. Resolución

Para realizar el diseño de un nuevo modelo de evaluación de calidad de proyectos de software libre, se necesita averiguar las características que tienen modelos anteriores y analizar las fortalezas y debilidades de los mismos. Ese trabajo se ha realizado en el capítulo 2 como parte de la lectura de la literatura para descubrir estado del arte en esta cuestión.

El modelo creado en este trabajo final de máster se basa en un modelo anterior llamado QuESO [23] como punto de partida. Este modelo se ha descrito en el capítulo de Estado del arte. Se ha escogido QuESO por ser un modelo moderno, desarrollado hace pocos años basándose en los modelos anteriores y cuyo desarrollo sigue estando activo en la actualidad [24]. Por tanto, el modelo propuesto en este trabajo fin de máster va a estar basado en QuESO, mientras que se van a plantear mejoras al mismo lo suficientemente importantes como para justificar la realización de este trabajo fin de máster.

A continuación, se realizará una definición formal del modelo planteado, detallando las características en las que se basan sus evaluaciones, las distintas métricas empleadas para obtener las mismas y el método de evaluación propuesto. Finalmente, se cerrará el capítulo con un ejemplo de aplicación del modelo propuesto en un proyecto popular de software libre.

3.1 Definición del modelo propuesto

El modelo QuESO [23], [24] fue diseñado a partir del modelo QualOSS como una manera de medir la calidad del ecosistema de un proyecto open source. Este modelo se basa en tres dimensiones principales:

- Calidad de plataforma.
- Calidad de comunidad.
- Calidad de red del ecosistema.

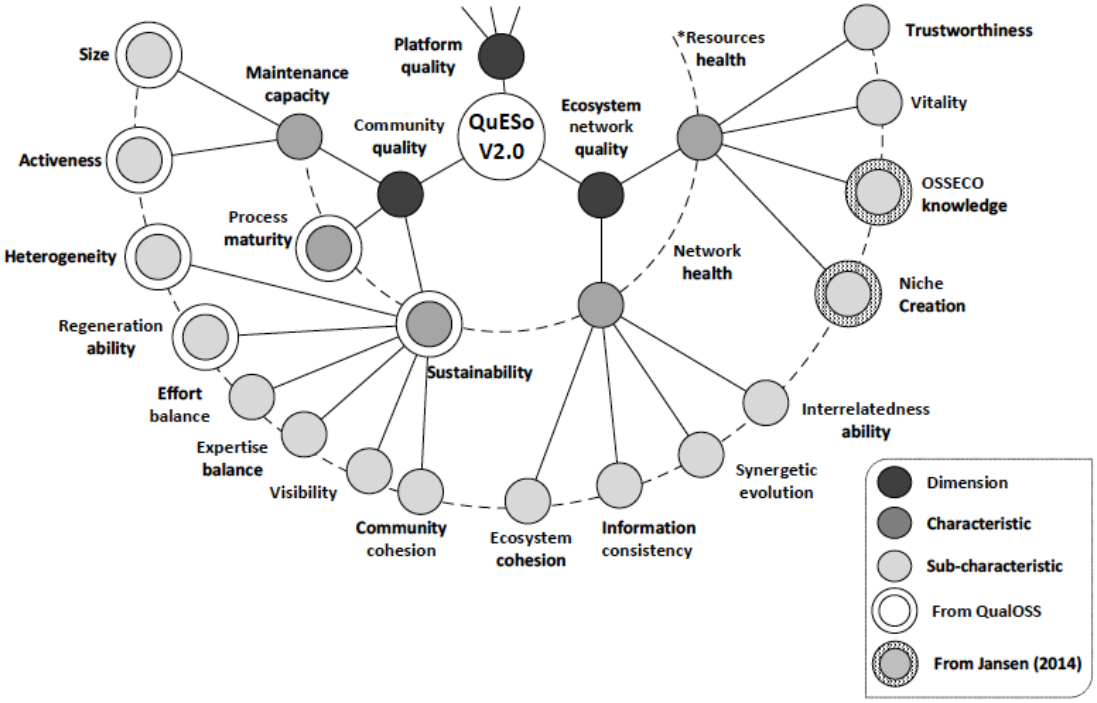


Ilustración 19: QuESO v2.0, modelo en que se basará este trabajo fin de máster. Diagrama sacado de [24].

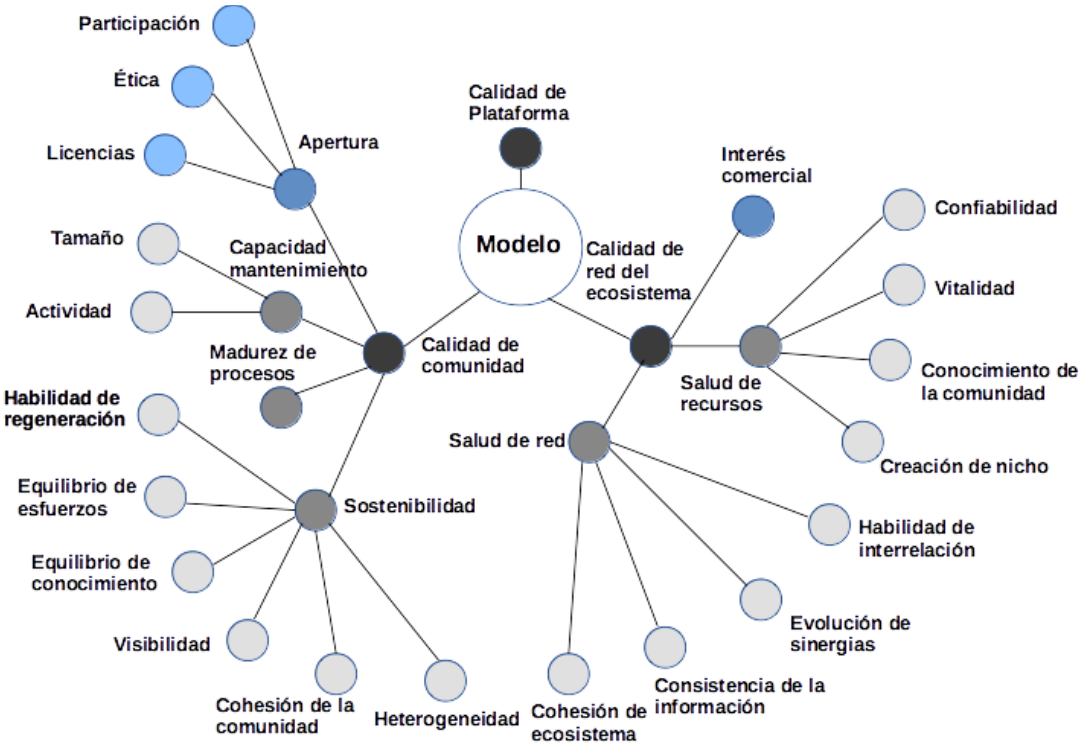


Ilustración 20: modelo final con los cambios propuestos en este trabajo fin de máster. En color azul se encuentran los cambios propuestos.

A continuación, se detallará cada una de ellas y las modificaciones propuestas al modelo QuESO y que forman el modelo planteado en este trabajo final de máster. En la Ilustración 20 se muestra un diagrama resumen de cómo sería el modelo final con las modificaciones planteadas en este trabajo final de máster, marcando las aportaciones realizadas en color azul.

Para explicar mejor el modelo, se van a proponer métricas para cada una de las sub-características. Estas métricas se van a listar en una tabla con los siguientes campos:

- **Métrica**. Indica el nombre de la métrica.
- **Descripción.** Breve descripción de la métrica.
- **Fuentes.** Dónde se puede obtener los valores para dicha métrica.
- **Método de obtención.** Cómo se puede obtener un valor de la métrica.
- **Interpretación.** Cómo interpretar el resultado de la métrica.

3.1.1 Calidad de plataforma

La calidad de la plataforma se refiere a la calidad del producto software en sí, sin tener en cuenta otros aspectos que son específicos de los proyectos bajo licencia libre. Tal y como se menciona en QuESO [23], las medidas relacionadas con esta dimensión no son particularmente distintas a las ya existentes en otros proyectos software que no tienen una licencia libre ni una comunidad detrás. Por tanto, no hay diferencias substanciales que afecten a proyectos bajo licencias libres y no se requiere el uso de mediciones de calidad del software diferentes a las ya existentes en otros modelos más genéricos.

Como QuESO está basado a su vez en QualOSS, y éste emplea un modelo ya establecido como ISO 25010 que se enfoca en la calidad genérica del software, este modelo sigue manteniendo la misma elección para esta dimensión sin requerir ningún cambio. Por tanto, se mantiene la misma elección para el modelo propuesto en este trabajo fin de máster.

3.1.2 Calidad de la comunidad

La calidad de la comunidad es una dimensión que persigue medir la calidad de la comunidad de usuarios y desarrolladores que hay alrededor de un proyecto de software libre. En esta dimensión, se tienen en cuenta las siguientes características:

- Apertura (específica del modelo propuesto y no presente en el modelo QuESO original).
- Capacidad de mantenimiento.
- Madurez de proceso.
- Sostenibilidad.

Las tres últimas características, presentes en QuESO, están basadas en características similares presentes en el modelo QualOSS, como así ocurre en parte de las sub-características que las componen.

Como aportación realizada por el modelo planteado en este trabajo final de máster, se presenta la característica Apertura que busca obtener una estimación de la calidad de la comunidad con referencia al aperturismo de la misma, tanto en forma de licencias de

software libre, como en aceptación de nuevos contribuidores, su participación en la comunidad y valores éticos de la propia comunidad.

3.1.2.1 Apertura

La característica "Apertura" es una de las aportaciones realizadas por este trabajo final de máster. La misión principal de esta característica es medir el grado de apertura de la comunidad del proyecto de software libre en referencia a varios aspectos relevantes.

Esta característica está basada en atributos de otros modelos como Capgemini OSMM y QSOS donde se tienen en cuenta atributos como licencia que tiene el software, sólo que busca ir más allá teniendo en cuenta una serie de sub-características que no fueron tomadas en cuenta anteriormente en otros modelos.

La razón principal para proponer esta característica de Apertura para la dimensión de calidad de la comunidad es que ésta afecta de manera primordial en cómo los usuarios y contribuidores al proyecto de software libre cómo ven al proyecto y, como consecuencia, a la cantidad de usuarios/desarrolladores, su diversidad y cómo contribuyen al proyecto software.

Por ejemplo, dentro de las motivaciones en las que una persona contribuye o se une a un proyecto de software libre, se encuentra la licencia bajo la que se encuentra el software. Este y otros factores, se han analizado en estudios previos [26]–[28]. Teniendo en cuenta estos factores, un proyecto de software libre tendrá más posibilidades de tener una comunidad de contribuidores y usuarios más dinámica y sana, aunque es cierto que habrá actores de la comunidad que no aprecien estos valores en igual medida y no afectarán a sus contribuciones de manera apreciable. Es por ello, que la primera sub-característica de Apertura tiene en cuenta, precisamente, tanto las licencias del software como los acuerdos de licencia de contribución (*Contributor License Agreement*, *CLA*).

La segunda sub-característica incluida en la característica de Apertura es la ética dentro del propio proyecto. En esta sub-característica se tiene en cuenta la diversidad de la comunidad y qué acciones realiza para atraer e integrar usuarios y contribuidores de minorías sociales en la comunidad. En general, suele haber un porcentaje bastante bajo de personas cuyo género no es el masculino, o cuya raza no sea caucásica, o que procedan de países del tercer mundo o cuyas culturas no son la mayoritaria presente en la comunidad del proyecto. En algunos proyectos, la realización de medidas para facilitar la participación de estos grupos sub-representados en la comunidad afecta de manera muy significativa a la hora potenciar la misma, ya sea por la aportación de ideas innovadoras, como para fomentar la heterogeneidad de la comunidad lo que la hará más robusta a cambios en el largo plazo. Otros factores que se tienen en cuenta serían los valores del proyecto en sí y cómo se comporta la comunidad en base a códigos de conducta, contratos sociales y otros.

La tercera sub-característica que forma parte de la característica de Apertura es la participación. Este ámbito cubre las facilidades que hay para participar en la comunidad, ya sea como usuario, como desarrollador o contribuidor no-desarrollador; así como el grado de participación en la toma de decisiones en base a la organización del proyecto: si

es asambleario, si hay comités de expertos o juntas directivas, si se tiene la figura de dictador benevolente[2] o si es una empresa la que dirige el proyecto sin dejar que otros actores tomen decisiones relevantes en el mismo. Esta sub-característica afecta a la calidad de la comunidad ya que dependiendo del modelo de participación que siga, los contribuidores actuales y los potenciales pueden decidir continuar contribuyendo al proyecto o decidir ir a otro que les dé más facilidades para hacerlo, lo cual afectaría a medio y largo plazo a la calidad de la comunidad y a su éxito.

Como resumen, las sub-características que componen la característica Apertura son:

- Licencias.
- Ética.
- Participación.

A continuación, se va a detallar cada una de ellas y se va a describir las métricas que las forman, así como unos indicativos de cómo se pueden obtener.

3.1.2.1.1 Licencias

La sub-característica de Licencia describe aquellas licencias que afectan a la comunidad de usuarios y desarrolladores. En esta sub-característica no estarán sólo las licencias del producto software en sí, sino aquellas que pueden afectar de igual manera como son los acuerdos de licencia de contribución entre otros.

Como ya se ha analizado en otros estudios, la ideología de software libre, donde se busca que las contribuciones sean compartidas y accesibles por otras personas, es algo que afecta de manera importante en la atracción de desarrolladores [29], [30], [31]. La ideología de software libre fue creada por Richard Stallman en el año 1985 cuando empezó con el proyecto GNU (*Gnu is Not Unix*) con el objetivo de desarrollar una alternativa libre al sistema operativo Unix de la época. Richard Stallman ha desarrollado la filosofía del software libre donde se busca que no se restrinjan las libertades de uso, modificación, copia y redistribución de versiones modificadas de un software. Como contrapartida a esta filosofía de software libre, surgió el movimiento de código abierto u *open-source* en el año 1998, mediante el rechazo de la parte ética de la misma, pero aceptando que el software libre tiene ventajas técnicas relevantes, principalmente por el acceso al código fuente para su estudio y modificación. El movimiento de código abierto fue creado por un grupo de personas donde se encontraban Eric S. Raymond y Bruce Perens, que fundaron poco después la *Open Source Iniciative*, que persigue promover esta nueva idea, sobretodo en entornos empresariales.

[2] Dictador benevolente es un líder de la comunidad que suele tener muy en cuenta la opinión de la comunidad pero, en ultima instancia, la decisión a tomar es suya únicamente.

Ilustración 21: Richard Stallman, fundador de Free Software Foundation, del proyecto GNU y promotor del software libre, es considerado por muchos como el padre del Software libre. Fotografia realizada por A. P.

Tabla 5: Descripción de las cuatro libertades del software libre.

Libertad	Descripción
0	Libertad de usar el programa, con cualquier propósito (uso).
1	Libertad de estudiar cómo funciona el programa y modificarlo, adaptándolo a las propias necesidades (estudio).
2	Libertad de distribuir copias del programa, con lo cual se puede ayudar a otros usuarios (distribución).
3	Libertad de mejorar el programa y hacer públicas esas mejoras a los demás, de modo que toda la comunidad se beneficie (mejora).

Una parte muy importante de la filosofía de software libre está en el tipo de licencia bajo la que está licenciado el software. La elección de la licencia afecta en la motivación de los desarrolladores que deseen contribuir en el proyecto ([31], [32]) y que, por tanto, afecta a la supervivencia del proyecto ya que sin desarrolladores estaría abocado al fracaso a largo plazo. Los usuarios y desarrolladores que anteponen la filosofía de software libre frente a otros aspectos se preocupan por temas como:

- El tipo de licencia es totalmente libre y cumple las 4 libertades del software libre (ver Tabla 5). Hay licencias que solamente permiten acceder al código fuente, pero no se permite realizar modificaciones al mismo ni redistribuir dichas modificaciones sin ningún tipo de restricción. La elección de licencias aprobadas por la *Free Software Foundation* (FSF) [33] [34] es normalmente un buen síntoma en este sentido. Por ejemplo, licencias como *GNU Public Licence* (GPL), *Berkely Software Distribution* (BSD) o licencia MIT son preferibles para obtener una comunidad de mayor calidad.
- La licencia sea de tipo *copyleft*. Que una licencia sea *copyleft* (o izquierdo de autor, en español) se refiere a exigir a las personas que redistribuyan el software con o sin modificaciones, que preserven las mismas libertades del software original.

- Hay algunas licencias de software libre que no cumplen dicha condición y ofrecen la libertad de re-licenciar el software y cerrarlo si así se desea por el redistribuidor. El problema con este tipo de licencias está en que una organización puede redistribuir el software libre bajo una licencia propietaria (también llamada privativa, porque priva de la libertad a los usuarios) y venderlo o incorporarlo como parte de otros productos sin necesidad de liberar los cambios. Es por ello, que muchos desarrolladores y contribuidores procuran evitar trabajar en proyectos software para así no ver que su trabajo es apropiado por terceros. Ejemplos de estas licencias serían la licencia BSD y la licencia MIT.

- La licencia es clara, sencilla, fácilmente accesible por los usuarios para su lectura y que no hay dudas de a qué ficheros afecta si es que hubiera varias licencias en el software. En algunas ocasiones, hay ficheros o librerías donde no está claro bajo qué licencia se encuentran y puede resultar problemático su uso. Por tanto, los usuarios y los desarrolladores que busquen contribuir en el proyecto esperan que la licencia sea clara, y que las partes donde aplica está perfectamente detallada. Las licencias presentes en un proyecto software no se delimitan únicamente al código fuente, sino que también afecta a la documentación[3] u otro tipo de obras artísticas, de entretenimiento o educativas como imágenes, sonidos, presentaciones u otros[4].

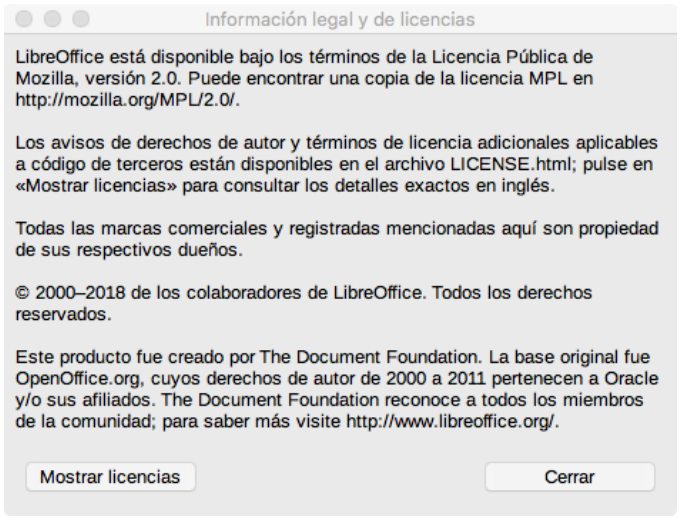


Ilustración 22: Captura de pantalla de la ventana de información legal y de licencias de LibreOffice, uno de los programas ofimáticos libres más populares.

Otro atributo que afecta al licenciamiento del software es el conocido como acuerdo de licencia de contribuidor o CLA en sus siglas en inglés. El acuerdo de licencia de contribuidor define los términos bajo los que está la propiedad intelectual de las contribuciones que se realicen a un proyecto de software, típicamente se emplea en proyectos bajo licencias de código abierto (*open-source*). La razón de la presencia de dicho acuerdo está en la necesidad de la organización que controla el proyecto de

[3] https://www.gnu.org/philosophy/free-doc.html
[4] https://www.gnu.org/licenses/license-list.html#OtherLicenses

protegerse en caso de disputas de copyright o, en el caso más común, permitir la posibilidad de re-licenciar el software bajo otras condiciones gracias a contribuciones de terceros.
La existencia de CLAs hace que la comunidad alrededor de un proyecto software considere si es prudente contribuir al proyecto, ya que podría encontrarse tiempo después con que este software esté bajo una licencia distinta a la que ellos accedieron a contribuir, incluyendo que el software sea ahora propietario. Se han encontrado múltiples ejemplos de estos casos.

Uno de los más recientes y más comentados en los últimos años, fue la exigencia de Canonical Ltd[5] a todos los contribuidores a su servidor gráfico Mir[6] de que tienen que aceptar y firmar un acuerdo de licencia de contribuidor [35], abriendo la posibilidad a que se pueda cambiar la licencia cuando la compañía desee. Este tipo de acuerdo fue muy criticado por la comunidad [36] ya podría cambiar las reglas del juego cuando Canonical lo desee, siendo rechazado por empresas como Intel [37]. En el año 2017, Canonical anunciaba que dejaba de desarrollar Mir para el escritorio y usaría Wayland[7] (competidor directo de Mir), gracias a su mayor apoyo en la comunidad y su mayor grado de desarrollo, y que sólo mantendría Mir para el sector de Internet de las cosas (IoT en inglés). Este ejemplo muestra que la calidad de una comunidad y, sobretodo, su supervivencia a largo plazo se ve afectada de manera significativa por decisiones de licenciamiento de la propiedad intelectual de las contribuciones de software y no solamente por motivos técnicos.

[5] Empresa desarrolladora de Ubuntu, una de las distribuciones de GNU/Linux más populares a nivel mundial.
[6] Servidor gráfico para GNU/Linix desarrollado por Canonical Ltd. El objetivo de Mir fue reemplazar el X Window System en Ubuntu en el año 2013.
[7] https://insights.ubuntu.com/2017/04/05/growing-ubuntu-for-cloud-and-iot-rather-than-phone-and-convergence/

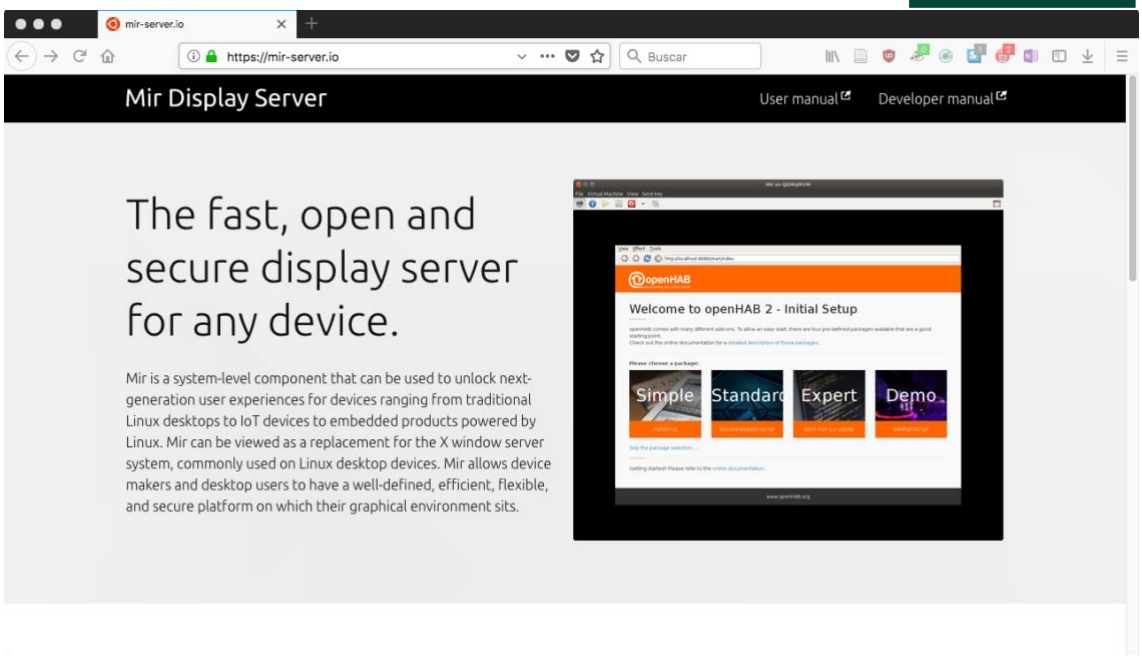


Ilustración 23: Página web de Mir, el servidor gráfico desarrollado por Canonical Ltd.

La necesidad de instalar software propietario para hacer funcionar el software o para desarrollar el mismo, pese a que el producto software final sea software libre, es otro aspecto que afectaría al licenciamiento. Hay casos en los que el producto software sería inútil sin este componente propietario; por ejemplo, los drivers para tarjetas de red inalámbricas necesitan de la carga de un firmware propietario para su funcionamiento.

En otros casos, el software es, simplemente, un *plugin* o una extensión a un software propietario y se necesita una licencia válida de éste para poder instalar el primero. Un ejemplo popular de este último caso, aunque sin estar bajo licencia libre, fue el origen del videojuego Counter Strike, que empezó siendo un *mod* del popular videojuego llamado Half-Life de la compañía Valve.

Al igual que en el caso de la licencia del software explicado previamente, este tipo de productos software pueden ver reducidas sus posibilidades de supervivencia a largo plazo debido a que la comunidad puede considerar que el software es tóxico, en el sentido de no seguir la filosofía de software libre.

Teniendo en cuenta todos los atributos mencionados en este apartado de licencias, se proponen las siguientes métricas.

Tabla 6: Lista de métricas propuestas para la sub-característica de licencia.

Métrica	Descripción	Fuentes	Método de obtención	Interpretación
Licencias de software	Licencias que afectan al software, documentación y otros recursos.	Fichero de licencias, asignación de licencia en cada fichero de código fuente, página web.	Obtener las licencias en las fuentes descritas,	Si las licencias son libres, mejor puntuación.

			comprobar si son libres.	
Acuerdo de licencia de contribuidor	Acuerdo de exigencia de transferencia de la propiedad intelectual de los contribuidores a la organización que dirige su desarrollo.	Página web.	Obtener el acuerdo de licencia de contribuidor en las fuentes descritas	Si no hay acuerdo de licencia de contribuidor o éste no limita ningún derecho, mejor.
Módulos propietarios	Módulos necesarios para el funcionamiento del software libre y que estén bajo licencia propietaria.	Documentación presente en la página web o en el propio software.	Obtener la información de las fuentes.	Menor número de módulos propietarios, mejor

3.1.2.1.2 Ética

Para fomentar el enriquecimiento de la calidad de una comunidad, una de las propuestas más interesantes es evitar la homogeneidad de la misma. La heterogeneidad de la comunidad es un aspecto ya cubierto en la característica de sostenibilidad que se detallará más adelante en este trabajo final de máster, pero en este caso se centra en otro ámbito.

Una comunidad será más vibrante, activa y de mayor calidad si las personas que la componen forman parte de diferentes culturas, distintas formaciones educativas, minorías sociales, de distintos géneros, razas y países de procedencia. Una comunidad que sea integradora y presente esta diversidad, tiende a ofrecer soluciones creativas a problemas tanto técnicos como sociales produciendo, en definitiva, una mayor solidez a largo plazo de la comunidad frente a posibles cambios. Una brecha muy importante se encuentra en el porcentaje de hombres y mujeres que contribuyen al software libre, pese a que hay estudios que sugieren que las mujeres son mejores programadoras [38].

En un estudio realizado por Linkedin [39], se muestra que el porcentaje de mujeres desarrolladoras de software está en el 16%. En otro estudio de la organización Women in Tech [40], el porcentaje de mujeres en las áreas de matemáticas e IT está en el 26%. En el caso específico de mujeres contribuyendo al software libre, el estudio realizado por Ghosh et at [41] indica que las mujeres forman solamente el 1% de los contribuidores, lo cual es un porcentaje extremadamente bajo, aún teniendo el cuenta el porcentaje de mujeres que desarrollan software según los estudios de Linkedin y Women in Tech.

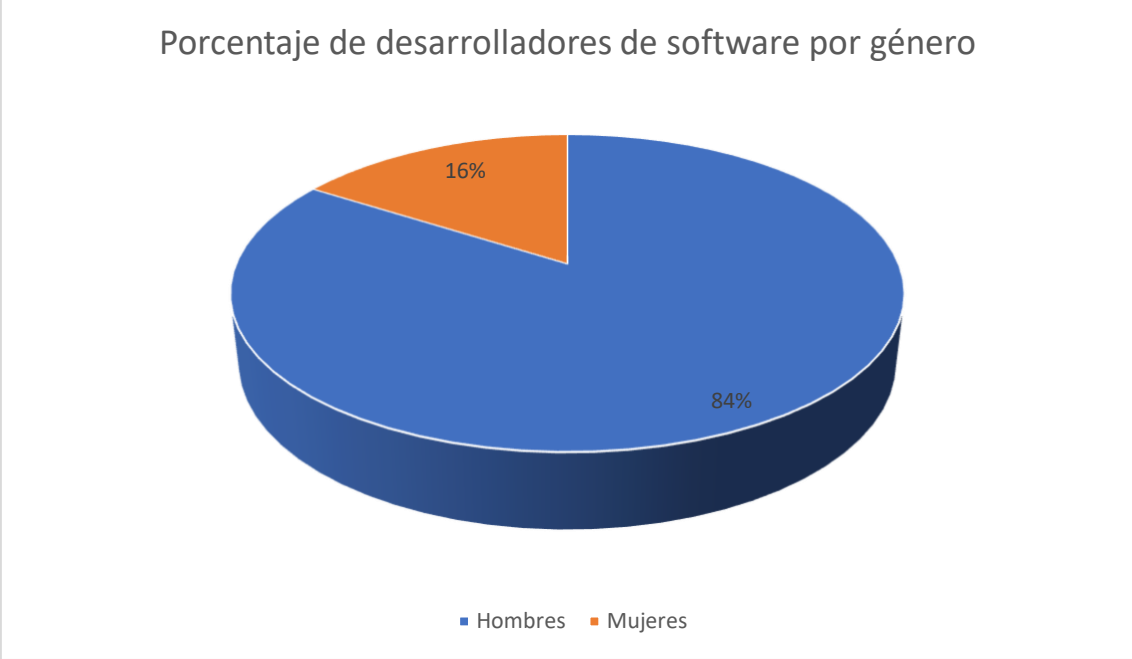


Ilustración 24: Porcentaje de hombres vs mujeres como desarrolladores en proyectos de código propietario. Fuente Linkedin [39].

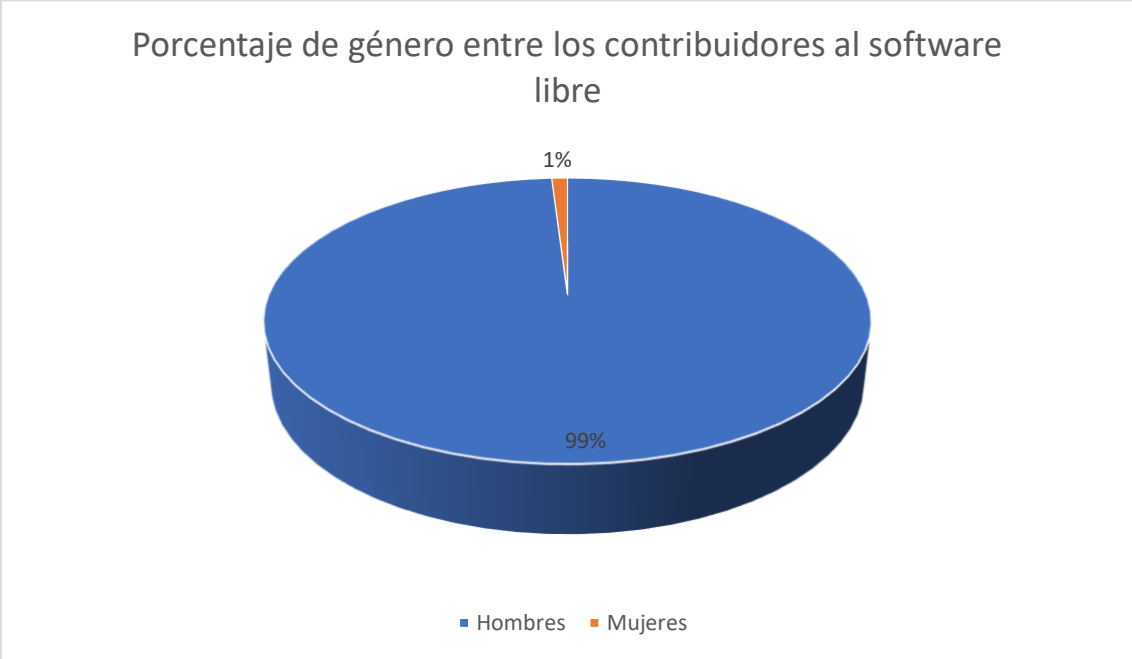


Ilustración 25: Gráfica que indica el porcentaje de hombres vs mujeres que contribuyen al software libre. Fuente [41].

El proyecto FLOSSPOLS [42] se encarga de realizar estudios para comprobar la evolución de la proporción de mujeres que son contribuidoras y las razones por las que dicha proporción no aumenta. En su estudio del período 2004-2006 [43], se indican varias razones de la tan acusada desigualdad de género. En dicho estudio, se propone que una de las razones sea la necesidad de realizar horas extras en el tiempo libre para tener los suficientes conocimientos del software como para realizar contribuciones; en el caso de

las mujeres, suele ser aceptado socialmente que las mujeres tengan más obligaciones domésticas (ya sea cuidado de personas dependientes, cuidados de hijos, tareas del hogar) que los hombres, provocando como consecuencia que no puedan dedicar el mismo número de horas que ellos. No obstante, en el mismo estudio se indican otras razones que pueden afectar igualmente a este porcentaje: normalmente las mujeres encuentran más dificultades para contribuir solamente por el hecho de ser mujeres, hay un porcentaje relevante de mujeres que son acosadas en conferencias e incluso se encuentran comportamientos discriminatorios en foros online o en los entornos de trabajo. También se encuentran diferencias de género en la edad de empezar a usar un ordenador y en qué edad se posee el primer ordenador, lo cual puede indicar un problema cultural en la sociedad.

Hay varias medidas que se pueden tomar para contrarrestar esta discriminación de género. La primera medida es establecer un código de conducta donde se listen los comportamientos que no se consideran adecuados en la comunidad, incluyendo ámbitos como la violencia física, violencia verbal, acoso y la discriminación entre otros, y cómo se deben de reportar las infracciones de dicho código, ya sean por las víctimas como por personas que son testigos del mismo. En los últimos años, se ha implantado satisfactoriamente códigos de conducta tanto en proyectos de software libre como Debian[8], Freedesktop.org[9] y Libreoffice [10], como en conferencias de software libre X.org Developer Conference [11] , FOSDEM [12] y los eventos organizados por la Linux Foundation[13]. En comunidades tradicionalmente conflictivas como Linux kernel, se está implantando un código de conducta[14] para evitar las agresiones verbales que hubo en el pasado y que han provocado que muchos desarrolladores hayan dejado de contribuir a este proyecto de software libre.

Otra medida que se puede tomar es la creación de un programa que fomente la incorporación de minorías como podría ser mujeres, personas de minorías culturales o de cualquier otro tipo, a la comunidad. Este tipo de iniciativas se centran normalmente en ofrecer alguna cantidad de dinero para incentivar las contribuciones de estas personas, así como facilitar su aprendizaje mediante la figura de un mentor que les guíe en todo el proceso y les revise su trabajo para que sea de valor para la comunidad. Ejemplos de estas iniciativas las encontramos en:

- Google Summer of Code[15], el cual es un programa de Google para que los estudiantes de todo el mundo puedan empezar su carrera contribuyendo al software libre, como una manera de ayudar a los proyectos y a los propios estudiantes, tanto mejorando su

[8] https://www.debian.org/code_of_conduct
[9] https://www.freedesktop.org/wiki/CodeOfConduct/
[10] https://www.documentfoundation.org/foundation/code-of-conduct/
[11] https://www.x.org/wiki/XorgFoundation/Policies/Harassment/
[12] https://fosdem.org/2018/practical/conduct/
[13] https://events.linuxfoundation.org/events/linux-security-summit-north-america-2018/attend/code-of-conduct/
[14] https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/CodeOfConflict?id=b0bc65729070b9cbdbb53ff042984a3c545a0e34
[15] https://summerofcode.withgoogle.com/

currículum como dándoles una pequeña ayuda económica. Google selecciona los proyectos de software libre seleccionables por los candidatos.

- Outreachy[16]. Es un programa que empezó estando bajo el paraguas de GNOME pero que ha crecido para ser una organización con entidad propia. Outreachy se centra en ofrecer ayuda económica y técnica a personas que forman parte de grupos sub-representados en la tecnología, principalmente en proyectos de software libre, desde haciendo tareas de programación, como de experiencia de usuario, documentación, ilustración y diseño gráfico. Un proyecto de software libre que ofrezca la posibilidad de trabajar en él mediante este programa se debe de ofrecer a la organización de Outreachy. Los patrocinadores del programa son normalmente empresas interesadas en buscar talento para sus plantillas y proyectos de software libre que buscan que una cierta característica sea implementada.

Ilustración 26: Cartel anunciando las condiciones del programa de Outreachy.

Hay más medidas dentro de las propias comunidades para ayudar a las personas que pertenezcan a grupos sub-representados. Una de ellas es la creación de grupos que las

[16] https://www.outreachy.org/

representen dentro de la comunidad. Por ejemplo, en Debian existe el grupo Debian Women[17] que busca la integración de las mujeres en la comunidad de Debian.
La finalidad de estas iniciativas no sólo que personas de estos grupos sub-representados contribuyan durante un tiempo limitado en la comunidad, sino que se unan como contribuidores activos durante mucho tiempo y puedan encontrar trabajo gracias a la experiencia ganada.

Por otra parte, si la comunidad tiene unos valores éticos marcados, éstos pueden ayudar a afianzar la comunidad y hacer más robusta en la resolución de conflictos y en el fomento de la colaboración. En el artículo de Margaret S. Elliot y Walt Scacchi [44] se analizó la comunidad GNU y los valores que tenía en relación al software libre para estudiar cómo resolviendo conflictos, la comunidad se hacía más fuerte. En proyectos como Debian, existe un contrato social[18] donde se declaran las intenciones y valores del proyecto, donde se indica que Debian permanecerá 100% libre, contribuirá a la comunidad de software libre, no ocultará los problemas, la prioridad son los usuarios y los desarrolladores, y que acepta que sus usuarios utilicen programas que tengan una licencia no libre en Debian, incluso al punto de dar soporte oficial al uso de dicho software. Otras distribuciones de GNU/Linux tienen contratos sociales similares, como por ejemplo Mageia [19] (ver Ilustración 27).

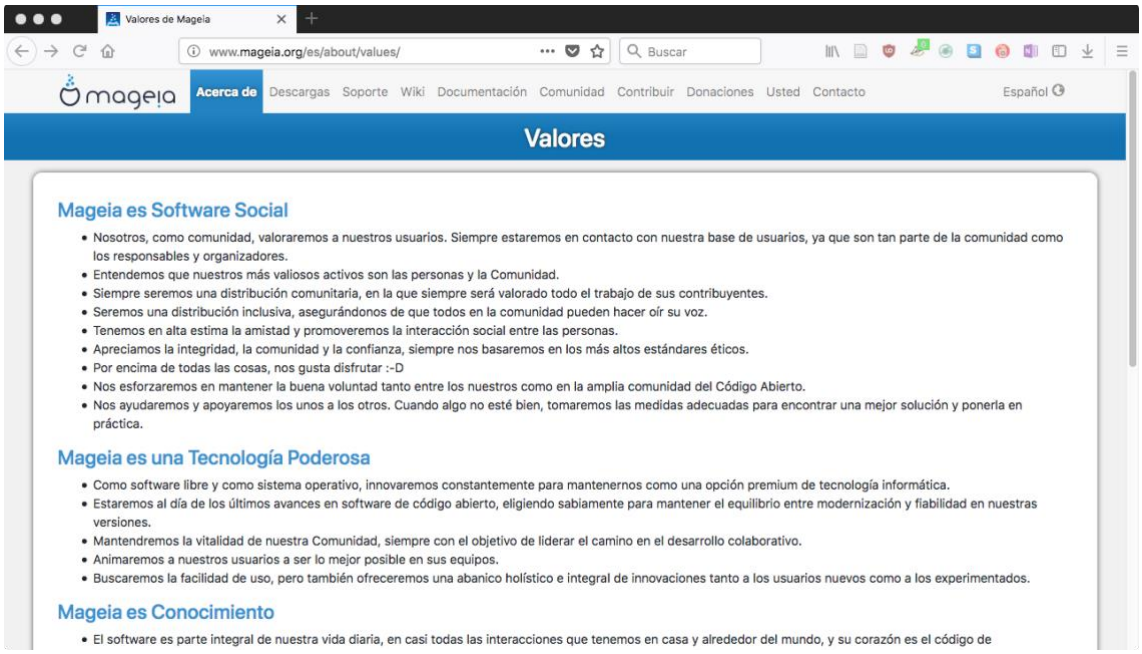


Ilustración 27: Captura de pantalla de la página web de Mageia que muestra los valores que comparte la comunidad.

Las métricas propuestas en este trabajo final de máster para la sub-característica Ética son las mostradas en la siguiente tabla.

[17] https://www.debian.org/women/
[18] https://www.debian.org/social_contract.es.html
[19] http://www.mageia.org/es/about/values/

Tabla 7: Lista de métricas propuestas para la sub-característica de ética.

Métrica	Descripción	Fuente	Método de obtención	Interpretación
Código de conducta	Acuerdo de la comunidad para establecer qué comportamientos son inadecuados, cómo notificarlos y sus castigos.	Página web.	Presentcia del código de conducta	Mejor puntuación si hay código de conducta y éste es claro.
Medidas de integración de minorías	Catálogo de medidas que toma el proyecto para integrar personas de grupos sub-representados en la comunidad: grupos específicos en la comunidad como Debian Women, iniciativas tipo Outreachy, etc.	Página web.	Cuántas medidas hay.	Mejor puntuación si hay más medidas de integración.
Valores	Lista de valores que comparte la comunidad: libertad de software, diversidad, entre otros. Suele estar recogido en un contrato social o similar.	Página web.	Número de valores de la comunidad	Mejor puntuación cuantos más valores haya y más importantes sean para la comunidad.

3.1.2.1.3 Participación

Las comunidades alrededor de proyectos de software libre también se caracterizan por la forma en la que se organizan. Los desarrolladores de software libre suelen contribuir a proyectos donde han recibido ayuda antes [32], discutir las soluciones propuestas en foros públicos [27] sin importar si trabajan para una empresa o son voluntarios y, sobretodo en el caso de voluntarios, no se les puede imponer tareas a realizar [27].

Este último punto es muy importante en el caso de voluntariado. Los voluntarios suelen trabajar en el proyecto durante su tiempo libre por diversas razones, desde altruistas (aportar algo útil a la comunidad, ideología, motivación para aprender algo novedoso) a mejorar su empleabilidad laboral futura, obtener reputación, entre otras [28]–[30], [31]. Al trabajar de manera voluntaria, suelen ser ellos los que escojan las tareas que quieren realizar, siendo la imposición de las tareas algo que suele dar problemas dentro la comunidad.

Es por ello, que los proyectos de software libre que no dependan de una empresa suelen centrar sus esfuerzos en intentar captar voluntarios para distintas tareas, pero sin obligar a nadie a realizar una tarea concreta, ya que realiza sus contribuciones generalmente durante su tiempo libre. Por ejemplo, en las distribuciones de GNU/Linux se suele buscar voluntarios para realizar tareas como empaquetado de aplicaciones, *triaging*[20] de bugs, diseño gráfico, traducciones… pero sin forzar a un contribuidor a escoger un área en concreto. En el caso de proyectos dirigidos por empresa, este hecho es distinto ya que

[20] Triaging de bugs significa todas las acciones necesarias para clasificar un bug en base a su importancia, relevancia, número de usuarios afectados entre otros motivos, si es un duplicado de otro existente, etc. De esta manera, otros desarrolladores podrán priorizar dicho bug frente a otros menos urgentes o importantes.

serían los propios empleados de la empresa los que asumirían las tareas como parte de su trabajo diario.

Otra parte importante sería cómo se toman las decisiones en la comunidad. Tradicionalmente, en el software libre se ha seguido el modelo bazar en contraste del modelo catedral [45]. Como parte de este modelo, los centros de decisión se han focalizado en alguno de estos tipos de jerarquía, muchos de ellos basados en la meritocracia [46]:

- Comités. Suele haber un comité técnico y/o un comité ejecutivo que toma decisiones técnicas/ejecutivas sobre el devenir del proyecto software. Este comité puede ser escogido entre contribuidores elegidos por la comunidad o escogidos directamente si fuera un proyecto de una empresa. Ejemplos de este caso los encontramos en distribuciones GNU/Linux como Mageia y OpenSUSE.
- Dictador benevolente. Es una figura bastante particular, se centra el poder de decisión en una sola persona que hace el rol de dictador a la hora de tomar decisiones, pero que procura buscar el bien común en la comunidad (benevolente). Un ejemplo de este caso sería la figura del fundador de Ubuntu, Mark Shuttleworth, donde él decide dónde se va a dirigir el proyecto, en parte porque es el dueño de Canonical, la empresa que controla Ubuntu y lo finanza económicamente. Otro ejemplo sería la figura de Linus Torvalds en el kernel de Linux cuando hay un conflicto técnico donde se necesite tomar una decisión.
- Modelo asambleario. No hay un comité que tome las decisiones importantes del proyecto, sino que éstas se deciden por medio de votaciones por los contribuidores de la comunidad. En este caso estaría la comunidad de Debian como ejemplo más representativo.
- Empresa. Es una empresa la fundadora del proyecto y la mayor parte de los contribuidores son trabajadores de dicha empresa, con lo que controlan todo lo que se hace en el proyecto y deciden dónde dirigir el proyecto sin que se consulte, en muchas ocasiones, al resto de la comunidad. Ejemplos de este caso sería Android (controlado por Google), WebKit (controlado por Apple) y Chromium (controlado por Google).

Hay más modelos, pero, en general, los modelos mencionados suelen ser los que se encuentran más a menudo en las comunidades de software libre. No obstante, hay casos como la librería de gráficos 3D llamada Mesa, donde hay una colaboración entre voluntarios de los drivers libres y las empresas (Intel, AMD, Igalia, Collabora, Valve, entre otras). En Mesa no hay un órgano que gobierne todo el proyecto, pero sí se podría decir que los sub-proyectos en los que se dividen son autogobernados por los contribuidores de estos sub-proyectos.

Otro aspecto muy importante es la jerarquía de la comunidad. En general, las comunidades del software libre se organizan en una estructura jerárquica como: usuarios, contribuidores puntuales (sin derechos de escritura en el repositorio de software), *commiters* (desarrolladores con permisos de escritura en los repositorios de software), *reviewers* (commiters que, además, revisan los commits de otros) y, en algunos casos, *mantainers* (personas que son responsables de una parte determinada del código y llevan

mucho tiempo contribuyendo). La existencia de esta jerarquía facilita que el trabajo se divida entre todos los miembros de la comunidad, siendo de mayor responsabilidad a las personas que hayan demostrado su valía.

Gracias a las propias particularidades del software libre y sus comunidades, que los proyectos de software libre creados y/o dirigidos por empresas tienen que adaptarse a los voluntarios de las comunidades de software [31]. Un proyecto de software libre que cumpla estas premisas, tendrás bastantes más posibilidades de perdurar en el futuro.

Por tanto, las métricas propuestas para esta sub-característica se muestran a continuación.

Tabla 8: Lista de métricas propuestas para la sub-característica Participación.

Métrica	Descripción	Fuentes	Método de obtención	Interpretación
Programas de captación de voluntariado	Programas de captación de voluntariado. Documentación sobre cómo contribuir en cada una de las áreas disponibles.	Página web.	Número de programas de captación de voluntariado	Cuantos más programas haya, mejor puntuación.
Toma de decisiones en la comunidad	La organización de la comunidad en la toma de decisiones importantes sobre el proyecto: comités, asambleas, un líder, empresa, etc.	Página web, listas de correo.	Puntuar la libertad entre 0 y 10, siendo 0 que la comunidad no puede decidir nada y 10 que se hace de manera democrática.	Mejor cuánto más poder se proporcione a toda la comunidad.
Organización jerárquica de la comunidad	Conocer cómo se organiza jerárquicamente una comunidad de desarrolladores	Página web, listas de correo	Obtener información sobre la jerarquía y los pasos que hay.	Si hay una estructura definida, abierta e inclusiva, mejor.

3.1.2.2 Capacidad de mantenimiento

Este sub-característica es la misma la existente en QuESO. En el artículo de Soto et al sobre QualOSS [18], se define la capacidad de mantenimiento como la habilidad de la comunidad para proveer los recursos necesarios para mantener sus productos y menciona que los aspectos relevantes para ello son el número de contribuidores de un proyecto y el tiempo que cada uno ellos contribuye al esfuerzo de desarrollo. Por tanto, las métricas asociadas a esta sub-característica serán el propio tamaño de la comunidad y lo activa que sea. El tamaño de la comunidad influye en la capacidad de mantenimiento, ya que mayor número de contribuidores hace que sea más fácil mantener el software, y puede ser medida en base al número de partners y el número de usuarios pasivos. Para la segunda sub-característica, se puede evaluar con mediciones como la actividad en la herramienta de seguimiento de errores, la fecha del último *commit* en el repositorio u otras similares que permitan saber si la comunidad es activa o no.

3.1.2.2.1 Tamaño

Esta sub-característica indica la evolución general del tamaño de la comunidad de un proyecto de software libre a lo largo del tiempo. Si se calculan estos valores en diferentes

momentos temporales, se puede estudiar la propia dinámica de la comunidad para conocer si está creciendo, decreciendo o su tamaño es estable a lo largo del tiempo. El tamaño de la comunidad afecta a la capacidad de mantenimiento, ya que se mide el número de contribuidores que son los que mantienen el software, a mayor tamaño se entiende que es más fácil mantener el software.

Tabla 9: Lista de métricas propuestas para la sub-característica Tamaño, obtenida de [24].

Métrica	Descripción	Fuentes	Método de obtención	Interpretación
Número de partners	Organizaciones que proporcionan distinto tipo de apoyo al proyecto.	Página web.	Contar las compañías, instituciones y organizaciones que apoyen a la comunidad.	Mayor número y mayor apoyo mejor.
Número de usuarios pasivos	Cuántos usuarios descargan y usan el software.	Repositorios de software.	Consulta a la base de datos.	Más usuarios, mejor.
Número de contribuidores	Cuántas personas colaboran de alguna manera en el proyecto.	Repositorios de software.	Consulta a la base de datos.	Más contribuidores, mejor.
Tamaño de la red de comunidad	Calcular el tamaño de la red social de la comunidad	Repositorios de software	Consulta a la base de datos.	Cuántos más nodos en la red, mejor.
Número de miembros de la comunidad del proyecto	Número de contribuidores, usuarios pasivos, partners de la comunidad	Repositorios de software	Es la suma de las tres primeras métricas.	Cuanto mayor sea el número, mejor.

3.1.2.2.2 Actividad

La actividad como sub-característica de calidad se refiere a lo activo que es la propia comunidad, en el sentido de resolver bugs, contestar emails, mensajes en los foros, etcétera.

A continuación, se listan las métricas asociadas con esta sub-característica que define QuESO.

Tabla 10: Lista de métricas propuestas para la sub-característica Actividad, basado en [24].

Métrica	Descripción	Fuentes	Método de obtención	Interpretación
Actividad en el bug tracker	Actividad de la comunidad dentro del bug tracker.	Bug tracker	Contar el número de actividades (bugs abiertos, cerrados, cambios de estado, tiempo medio para cerrar un bug, número de comentarios por bug, etc). Cualquier medida de ellas sirve.	Mayor número en cada una de ellas mejor.

Correlación entre comunicación y resolución de bugs.	Calcular la correlación entre la comunicación entre desarrolladores y la calidad del producto.	Listas de correo y bug tracker.	Calcular el número de comunicaciones entre desarrolladores en la lista de correo y el número de bugs resueltos.	Si la correlación es negativa, mejor.
Fecha del último commit	Calcular a qué ritmo se desarrolla el software.	Repositorios de software.	Consulta a la base de datos.	Más reciente, mejor.
Historial de versiones	Calcular el ritmo de evolución del software.	Página web.	Calcular el número de versiones mayores lanzadas al año	Cuántas más versiones mejor.
Fecha de la última *release*.	Cómo es la comunidad actualmente.	Repositorios de software	Consulta a los repositorios de software.	Cuanto más reciente, mejor.
Número de ficheros cambiados	Calcular la media de ficheros cambiados por commit.	Repositorios de software.	Consulta a los repositorios de software.	Más cambios mejor.
Número de ficheros por versión	Número de ficheros que componen una cierta versión del software para ver su evolución.	Repositorios de software	Consulta a los repositorios de software.	Más ficheros, mejor.
Actividad de comunicaciones	Número de correos y respuestas a un correo para comprobar la fluidez de las comunicaciones.	Listas de correo	Calcular la media de correos y respuestas en hilos de correos.	Más actividad, mejor.
Actividad de los contribuidores	Número de commits y el número de contribuidores.	Repositorios de software	Consulta a los repositorios de software.	Más commits y más contribuidores da mayor puntuación.
Ratio de commits	Calcular el tiempo medio entre primer y el último commit entre todos los proyectos de la comunidad.	Repositorios de software	Consulta a los repositorios de software. $CCR = \frac{\sum_1^n PCT(p_i)}{n}$ PCT = intervalo de tiempo de un proyecto. n = número de proyectos	Si el ratio es menor, más activo se considera un proyecto.
Punto de declive	En qué momento el número de correos empezó a decrecer.	Listas de correo.	$ADP = x < \max(n) * 0{,}8$	-
Punto cúspide	En qué momento, el número de correos fue el máximo.	Listas de correo.	$ACP = \max(n)$	-
Periodo de actividad de la comunidad	Conocer si la comunidad está activa todo el rato o sólo durante	Listas de correo.	$[ACP, ADP]$	Cuanto mayor sea el periodo, mejor.

	determinados periodos de tiempo.			
Tiempos de la comunidad	Calcular el tiempo medio entre una pregunta o requerimiento y una primera respuesta.	Listas de correo, repositorios de software y bug trackers.	$ET = \frac{\sum_1^n TFR(r_i)}{n}$ TFR = intervalo de tiempo de la primera respuesta a un requerimiento i. n = número de requerimientos en el proyecto.	Cuanto más bajo sea, mejor.
Número de eventos	Conocer la actividad social de la comunidad.	Página web.	Contar el número de eventos y personas que participan en dichos eventos.	Cuantas más personas atiendan los eventos y más eventos haya mejor.

3.1.2.3 Madurez de procesos

En el caso de las comunidades de proyectos de software libre, el proceso de desarrollo de software no puede ser estandarizado en modelos de procesos, herramientas y maneras de trabajar, ya que la manera de trabajar varía mucho entre los propios voluntarios. Es decir, la aplicación de procesos de madurez tradicionales, como sería CMMI-DEV, se hace mucho más difícil de justificar en este caso, pese a que haya muchas medidas que sí serían perfectamente aplicables. A la pregunta si la madurez de procesos puede ser evaluada, se debe responder primero a las preguntas de ¿está documentado el proceso para realizar una tarea? ¿Hay un proceso establecido y que sea ejecutado consistentemente?
En este trabajo final de máster no se va a tratar este punto que está igualmente pendiente en QuESO [24]. En todo caso, se puede emplear el trabajo realizado por Soto et al [18] como punto de partida futura para tratar de ofrecer un conjunto de sub-características y métricas aplicables a este punto y que permitan realizar una valoración adecuadas de esta característica.

3.1.2.4 Sostenibilidad

La sostenibilidad se refiere a los métodos empleados por la comunidad por mantener los productos software que desarrolla en un espacio de tiempo prolongado. De acuerdo con Soto et al [18], se ve afectado por la heterogeneidad y la habilidad de regeneración, no obstante en QuESO [24] se añadieron sub-características como equilibrio de esfuerzos, equilibrio de conocimiento, visibilidad y cohesión de la comunidad.

3.1.2.4.1 Heterogeneidad

Una de las sub-características que afectan a la sostenibilidad de la comunidad es la heterogeneidad. Por ejemplo, si una comunidad está principalmente formada por empleados de una particular compañía, hay un riesgo significativo en la sostenibilidad de proyecto si la compañía decide recortar gastos en el mismo.

Hay aspectos medibles de la heterogeneidad como son la distribución geográfica de los miembros de la comunidad. En la tabla a continuación, se describen las métricas de este apartado.

Tabla 11: Lista de métricas asociadas a la heterogeneidad. Lista obtenida de QuESO [24].

Métrica	Descripción	Fuentes	Método de obtención	Interpretación
Distribución geográfica de miembros	Identificar la localización geográfica de los miembros de la comunidad.	Listas de correo.	Obtenerlo de las listas de correo.	Mayor número de localizaciones y más distribuido mejor.
Tipos de actividad de miembros	Cómo se distribuyen los miembros de la comunidad con respecto a los tipos de tareas a realizar.	Repositorio de software.	Identificar los tipos de actividad y contar los miembros que participan en cada una. Calcular el índice Gini [47].	Cerca de cero es mejor, ya que significa una distribución uniforme.
Variedad de proyectos	Cuántos proyectos tiene la comunidad.	Consulta en la base de datos.	Buscar información de proyectos	Más proyectos mejor.
Variedad de partners de la comunidad	Cuántos partners tiene la comunidad.	Encuestas a partners.	Primeros los partners son clasificacos por sus características (publico, privado). Segundo, calcular las proporciones de cada tipo en el mercado como punto de referencia. Calcular para cada partner en qué categorías encaja cada partner. Se calcula la covarianza de la variedad de partners con respecto al mercado.	Cuánta mayor covarianza mejor, ya que muestra la variedad de los partners.
Distribución en organizaciones de los miembros de la comunidad	Cuáles son las afiliaciones de los miembros de la comunidad.	Encuestas a los partners.	Contar el número de organizaciones a las que pertenece cada miembro.	Cuanto más hayan, mejor.

3.1.2.4.2 Habilidad de regeneración

Para mantener una comunidad a lo largo del tiempo, es importante que haya un flujo de contribuidores que reemplacen a los que dejan de contribuir, lo que permite que la comunidad no se vea resentida y se mantenga en tamaño o incluso crezca. Podemos definir, la regeneración como el grado de evolución a lo largo del tiempo del tamaño de la comunidad.

En la tabla que se muestra a continuación, se describen las métricas relacionadas con la sub-característica de habilidad de regeneración.

Tabla 12: Lista de métricas para la habilidad de regeneración. Lista obtenida de QuESO [24].

Métrica	Descripción	Fuentes	Método de obtención	Interpretación
Ratio de supervivencia de contribuidores	Cuál es el número de contribuidores supervivientes en la comunidad.	Repositorios de software, listas de	Siendo X el número de contribuidores que sobreviven, Y el número de	Mayor número mejor.

		correo y bug trackers.	contribuidores activos desde el principio, RA-CSR = X/Y, siendo el porcentaje de contribuidores que sobreviven al año.	
Ratio de miembros nuevos	Contar el número de nuevos miembros a la comunidad que se han afiliado en un periodo de tiempo.	Listas de correo	Siendo X el número de miembros activos, Y el número de miembros nuevos en un periodo de tiempo, entonces CNMR = Y/X	Mayor ratio, mejor.
Ratio de contribuidores nuevos	Contar el número de contribuidores a la comunidad que han hecho su primera contribución en un periodo de tiempo.	Repositorios de software, listas de correo y bug trackers.	Siendo X el número de contribuidores activos, Y el número de contribuidores nuevos en un periodo de tiempo	Mayor ratio, mejor, porque si hay más contribuidores nuevos, más posibilidades hay de supervivencia de la comunidad.

3.1.2.4.3 Equilibrio de esfuerzos

Una forma de facilitar la sostenibilidad de la comunidad es que el trabajo se reparte el trabajo de manera equitativa entre los miembros de la comunidad. Este tipo de reparto ayudaría a evitar estrés y *burnout* de los contribuidores más activos, además que evitaría que la comunidad sufriera en caso de que estos desarrolladores clave dejen el proyecto, lo que permite que la comunidad sobreviva mucho más tiempo.

En la tabla que se muestra a continuación, se describen las métricas relacionadas con la sub-característica de Equilibrio de Esfuerzos.

Tabla 13: Lista de métricas de Equilibrio de Esfuerzos. Lista obtenida de QuESO [24].

Métrica	Descripción	Fuentes	Método de obtención	Interpretación
Ratio de tiempo entre commits	Cuanto tiempo pasa entre los commits de todos los contribuidores.	Repositorios de software.	Calcular el tiempo entre commits, calcular la varianza de tiempo entre commits de un contribuidor y calcular la varianza de la variancia de todos los tiempos entre commits de todos los desarrolladores.	Menor varianza mejor.
Implicación de la comunidad en los proyectos	Cómo están distribuidos los commits de los proyectos de la comunidad.	Repositorios de software.	Contar el número de cada tipo de actividad en el tiempo por proyecto de la comunidad: A=Número de commits; B=Número de líneas cambiadas; D=Número de ficheros cambiados	Es mejor si el porcentaje de contribuciones es similar entre distintos proyectos.

Características estadísticas de los commits	Cómo es la variación de commits entre la historia de la comunidad.	Repositorios de software.	Calcular el número mínimo de commits, el menor cuartil, la mediana, el cuartil superior y el máximo número de commits.	Mayores valores, mejor.
Ratio de actividades de la comunidad	Comprobar si la distribución de actividades está equilibrada en los proyectos de la comunidad.	Listas de correo, repositorios de software y bug trackers.	Computar el índice Gini para todos o un subconjunto de commits, correos enviados, ficheros cambiados, bug reports. Esta computación se hace en distintos intervalos de tiempo.	Si el índice tiende a cero, mejor. Un valor de cero indica una distribución uniforme.
Ratio de miembros en sub-comunidades	Contar el número de sub-comunidades donde está presente un desarrollador	Listas de correo.	Siendo m el número de miembros de la comunidad, MR el número de subcomunidades donde el miembro i pertenece y n el número de sub-comunidades, entonces: $SoMR = \frac{\sum^{m} MR(i)}{n}$	Depende del contexto.
Ratio releases por desarrollador	Contar el número de releases que haya participado cada desarrollador	Repositorios de software.	Siendo SD la desviación estándar, X el número de releases por desarrollador, DRR = SD(X)	Menor es mejor.

3.1.2.4.4 Equilibrio de conocimiento

En cualquier desarrollo de proyectos software que sea medianamente complejo, se requiere la habilidad y conocimiento de un dominio específico. El conocimiento es una los atributos más importantes en la comunidad, donde muchos contribuidores (o personas muy involucradas en la comunidad) comparten un nivel de conocimiento técnico lo que permite la comunicación, discusión y mejora del software, fomentando las contribuciones en la comunidad. Si el conocimiento estuviera concentrado en unas pocas personas, estaríamos en una situación con alto riesgo para la comunidad, con lo que la distribución del conocimiento es una buena noticia para la supervivencia de cualquier comunidad de software libre.

En la tabla que se muestra a continuación, se describen las métricas relacionadas con la sub-característica de Equilibrio de Conocimiento.

Tabla 14: Lista de métricas de equilibrio de conocimiento. Lista obtenida de QuESO [24].

Métrica	Descripción	Fuentes	Método de obtención	Interpretación

Conocimiento de contribuidores	Cómo es el conocimiento en los contribuidores	Repositorios de software.	Contar el número de ficheros cambiados y clasificarlos por extensión.	Más balanceado mejor.
Ratio de longevidad en los contribuidores	Obtener el tiempo que ha estado un desarrollador contribuyendo a la comunidad.	Repositorios de software.	Siendo Time(Ci) el tiempo de contribuciones del desarrollador i, y m el número total de contribuidores, entonces $LoCR = \frac{\sum^{m} Time(C_i)}{m}$	Mayor ratio, mejor.
Ratio de experiencia por contribuidor	Contar el número de releases que cada contribuidor participó activamente	Repositorios de software.	Siendo NR(Ci) el número de releases que ha participado el contribuidor i, m el número total de contribuidores, entonces: $CER = \frac{\sum^{m} NR(C_i)}{m}$	Mayor ratio, mejor.
Número de proyectos por contribuidor	Obtener los proyectos que participa cada contribuidor de la comunidad	Repositorios de software.	Usar el índice Gini.	Si el índice tiende a cero, mejor. Un valor de cero indica una distribución uniforme.

3.1.2.4.5 Visibilidad

Otra de las sub-características importantes de la sostenibilidad de una comunidad es la capacidad de atraer personas que contribuyan y soporten los proyectos softwares. Esta sub-característica tiene el nombre de Visibilidad, donde se toman medidas como el número de eventos organizados por la comunidad, el número de patentes registras y otras métricas que se muestran a continuación.

Tabla 15: Lista de métricas para la sub-característica Visibilidad. Lista obtenida de QuESO [24].

Métrica	Descripción	Fuentes	Método de obtención	Interpretación
Número de miembros pidiendo nuevas características	Cuántos miembros de la comunidad piden nuevas características	Repositorios de software, bug trackers.	Contar los miembros que piden nuevas características.	Más miembros, mejor.
Número de ofertas de trabajo	Contar las ofertas de empleo para los miembros de la comunidad.	Redes sociales especializadas y páginas web.	Consulta en las webs.	Mayor número, mejor.
Número de descargas	Comprobar si los proyectos software son populares	Repositorios de software y páginas web de los proyectos.	Consulta a las bases de datos	Mayor número, mejor.

Número de subscriptores a las listas de correo	Cuál es el tamaño de los contribuidores de la comunidad.	Listas de correo	Consulta a la base de datos.	Mayor número, mejor.
Número de usuarios pasivos	Igual a la métrica presente en la sub-característica Tamaño.	-	-	-
Número de publicaciones científicas	Comprobar si los datos de la comunidad se usan en artículos científicos.	Librerías digitales y bases de datos científicas.	Búsqueda	Mayor número, mejor
Publicaciones en webs y social media	Ver si hay visibilidad de la comunidad en Internet	Blogs y redes sociales	Búsqueda con herramientas de análisis web.	Mayor número mejor.
Número de patentes	Cuántas patentes tiene la comunidad.	Miembros de la comunidad.	Encuestas	Mayor número, mejor.
Número de eventos	Cómo es la actividad social de la comunidad	Blogs de la comunidad.	Contar número de eventos y el número de asistentes a ellos.	Mayor número y número de asistentes mejor.
Reputación y opiniones de contribuidores	Cuál es la reputación de los contribuidores de la comunidad.	Contribuidores y páginas web	Siendo TFR el intervalo de tiempo entre una pregunta i y la primera respuesta y n el número de preguntas, entonces: $ET = \frac{\sum^{n} TRF(r_i)}{n}$	Mayor reputación, mejor.
Distribución geográfica de los miembros	Es igual que la métrica de Heterogeneidad.	-	-	-
Aceptación de la comunidad	Cómo es la aceptación de la comunidad para organizaciones comerciales	Encuestas	Encuestas a partners.	Mayor es mejor.
Número de páginas referenciando la comunidad	Cuántas páginas referencian la página web de la comunidad.	Herramientas de análisis web.	Contar el número de páginas web que referencian.	Mayor número es mejor.

3.1.2.4.6 Cohesión en la comunidad

Para garantizar la sostenibilidad de la comunidad, una parte importante sería la cohesión de la comunidad, es decir, una estructura de la comunidad que garantice una buena salud en ella. Aspectos que afectan a la cohesión son el flujo de información entre los miembros de la comunidad y los factores que ayuden a evitar el acceso de nuevos competidores al mercado.

En la tabla que se muestra a continuación, se describen las métricas relacionadas con la cohesión de comunidad.

Tabla 16: Lista de métricas para la sub-característica cohesión de comunidad. Lista obtenida de QuESO [23].

Métrica	Descripción	Fuentes	Método de obtención	Interpretación
Ratio de centralidad de la comunidad	Cuál es la habilidad de un nodo para actuar de mediador de la comunidad.	Repositorios de software, listas de correo.	Siendo BNi, la centralidad de un nodo i, NoN el número de nodos, entonces: $CBCR = \frac{\sum^{NoN}(NB_i > 0)}{NoN}$	Más cerca de cero, mejor.
Clúster de colaboración de desarrolladores	Identificar clústeres de desarrolladores en la comunidad.	Repositorios de software, listas de correo.	Seleccionar un método para identificar clústeres automáticamente basados en la estructura de la red.	Un clúster por proyecto, mejor.
Enlaces a otras sub-comunidades	Cómo están conectadas las distintas sub-comunidades	Repositorios de software.	Se mide el número de dependencias de código y de contribuidores y se suman ambas.	Mayor valor, mejor.
Grado de presencia de actores clave	¿Hay actores claves en la comunidad?	Listas de correo, repositorios de software y bug trackers.	Siendo O(i) el grado de conexión de un nodo i, m el número de miembros de la comunidad, entonces $OoKA = \sum^{m} O(i) \geq 2\sigma + mu$	Una red con actores clave está mejor relacionada, es decir, OoKA >= 1

3.1.3 Calidad de la red del ecosistema

En esta dimensión se tienen en cuenta varias características principales:

- Salud de la red.
- Salud de recursos.
- Interés comercial (específico de este modelo y no presente en el modelo QuESO).

Esta dimensión se centra en mirar la salud de la red de partners de la comunidad y de los recursos que estos disponen, incluyendo los económicos, para comprobar que la comunidad de un proyecto de software libre puede sobrevivir en el futuro. Como aportación de este trabajo fin de máster, se añade la característica de Interés Comercial que se centra en analizar el potencial comercial de un proyecto de software libre: si tiene mucho potencial, es más probable que atraiga contribuidores u otras organizaciones (empresas, instituciones públicas…).
Se describen cada una de estas características y sub-características, junto con sus métricas asociadas a continuación.

3.1.3.1 Salud de la red

La salud de la red se refiere a cómo están interconectados los partners y el impacto de cada uno de ellos en la comunidad y el ecosistema que la conforma.

3.1.3.1.1 Cohesión del ecosistema

En la cohesión del ecosistema se analiza mediante un grafo de la red del ecosistema de un proyecto de software libre, lo importante que son los actores claves en la misma, el grado de colaboración que hay entre los distintos partners, y cómo se conectan entre ellos.

En la tabla que se muestra a continuación, se describen las métricas relacionadas con la sub-característica de cohesión del ecosistema.

Tabla 17: Lista de métricas para la sub-característica cohesión del ecosistema. Lista obtenida de QuESO [23].

Métrica	Descripción	Fuentes	Método de obtención	Interpretación
Número de nodos para desconectar el ecosistema	¿Cómo está el ecosistema conectado?	Consultas a bases de datos y encuestas sobre repositorios de software, listas de correo, etc.	Definir una red social del ecosistema, calcular el número mínimo de nodos que serían necesario quitar para que la red se vuelva desconectada.	Más es mejor.
Grado de *outdegree* de actores clave	Cuántos actores clave tiene la comunidad.	Consultas a bases de datos y encuestas sobre repositorios de software, listas de correo, etc.	Definir una red social del ecosistema. El *outdegree* de un nodo v es el número de segmentos donde v es su nodo inicial.	Más alto es mejor, significa que un actor juega un rol importante en el ecosistema.
Coeficiente de clústering en la comunidad	Cómo de cercanos están los nodos para crear un grafo completo con sus vecinos.	Repositorios de software, listas de correo, bug trackers.	Siendo Li el número de enlaces entre nodos vecinos Ni del nodo i, N es el número de nodos en la red, entonces: $CC_i = \frac{L_i}{N_i(N_i-1)}$ $OCC = \bar{c} = \frac{1}{n}\sum_{1}^{n} CC_i$	Más cercano a 1 mejor.
Número de conexiones de los partners	¿Cuántas conexiones tienen los partners en la red?	Consultas a bases de datos y encuestas sobre repositorios de software.	Definir una red siendo los nodos los proyectos y los partners y cualquier comunicación es un enlace. Contar en número total de conexiones entre especies o partners centrales y no centrales.	Más es mejor. Mayor número de conexiones significa una mayor interrelación.

3.1.3.1.2 Consistencia de la información

Una manera de mejorar una comunidad es que haya un vocabulario común que ayuda a comunicarse fácilmente entre sus miembros, es decir, la presencia de un lenguaje de dominio es algo bueno para la comunidad.

En la tabla que se muestra a continuación, se describen las métricas relacionadas con la sub-característica de consistencia de la información.

Tabla 18: Lista de métricas para la sub-característica de consistencia de la información. Lista obtenida de QuESO [23].

Métrica	Descripción	Fuentes	Método de obtención	Interpretación
Sinónimos en el mapa de vocabulario	Obtener una vista general del dominio del lenguaje	Repositorios de software, listas de correo y blogs.	Construir un mapa de vocabulario e identificar sinónimos mediante consultas y minado de textos.	Menor número de sinónimos, mejor. Un lenguaje común, mejor.
Análisis de sentimiento	Cómo es el mensaje del contenido del vocabulario en la comunidad.	Listas de correo.	Preprocesar los mensajes de correo electrónico, configurar las palabras sentimentales y clasificarlas.	Un lenguaje positivo es mejor.

3.1.3.1.3 Evolución de las sinergias

La evolución de las sinergias es la habilidad de los subsistemas que conforman la comunidad de formar una estructura dinámica y estable en el espacio-tiempo, es decir, se mide la colaboración de los miembros claves de la comunidad y que esta colaboración se mantenga en el tiempo.

En la tabla que se muestra a continuación, se describen las métricas relacionadas con la sub-característica de evolución de las sinergias.

Tabla 19: Lista de métricas para la sub-característica de evolución de las sinergias. Lista obtenida de QuESO [23].

Métrica	Descripción	Fuentes	Método de obtención	Interpretación
Distribución de partners	Cual es la distribución de partners en el ecosistema	Partners	Calcular el índice Gini para contar el número de partners por proyecto.	Cerca de cero es bueno, un valor de cero indica una distribución uniforme de los partners entre los proyectos.
Popularidad de la comunidad	Cómo es la popularidad de la comunidad en empresas externas.	Sponsors	Calcular la entropía de la información.	Mayor es mejor. Un valor bajo significa que un pequeño número de compañías apoyan el proyecto.
Partnerships e integrabilidad	Cómo son de integrables los	Blog	Consultas en webs y encuestas.	Más alto mejor.

de los proyectos de la comunidad	proyectos de la comunidad			
Reciprocidad de la comunidad	Cómo es la reciprocidad de la comunidad.	Repositorios de software, listas de correo y blog.	La reciprocidad es el coeficiente de correlación entre las entradas de una matriz de adyacencia de un grafo dirigido, formado por la red social donde los nodos son los miembros de la comunidad y los segmentos dependen del tipo de análisis del ecosistema (código, correos, dependencias, recursos).	P>=0 es mejor.

3.1.3.1.4 Habilidad de interrelación

Otro aspecto importante para mejorar la salud de la red que forma la comunidad sería la interrelación, que es la habilidad de establecer conexiones entre nodos de la red basado en las formas en las que ellos contribuyen colaborativamente en los proyectos de la comunidad.

A continuación se describen las métricas que afectan a esta sub-característica.

Tabla 20: Lista de métricas para la sub-característica de la habilidad de interrelación. Lista obtenida de QuESO [23].

Métrica	Descripción	Fuentes	Método de obtención	Interpretación
Evolución de la conectividad de los partners	Cómo es la evolución de la conectividad de los partners en la comunidad.	Repositorios de software, listas de correo y blog.	Base de datos y consultas a páginas web.	Si la evolución indica crecimiento, es mejor.
Evolución de la conectividad de los partners con otros miembros de la comunidad	Cómo evoluciona la conexión de los partners con otros miembros de la comunidad	Repositorios de software, listas de correo y blog.	Base de datos y consultas a páginas web.	Si la evolución indica crecimiento, es mejor.
Evolución de la centralidad de la comunidad	Descubrir qué miembros tienen a estar más conectados entre ellos.	Repositorios de software, listas de correo y blog.	Proceso definido en [48].	La centralidad es usada con un análisis de la red como una medida para indicar la importancia de un nodo en la red.

3.1.3.2 Salud de recursos

En muchas comunidades, la presencia de un flujo de dinero en la comunidad asegura su supervivencia a largo plazo, con lo que una de las características es asegurar que hay los recursos necesarios en una comunidad para poder mantenerse. Esta característica se centra en ver si se ha creado un nicho donde se pueda mantener activa la comunidad, el

conocimiento que provee la propia comunidad, su vitalidad y la confiabilidad de los partners.

Que haya unos partners que se relacionen bien entre ellos, con una buena salud financiera que permita sostener los recursos que necesita una la comunidad para mantenerse o incluso crecer de manera sostenible es muy importante.

En las siguientes páginas se van a describir las sub-características que lo conforman, así como las métricas que se asocian a cada una de ellas.

3.1.3.2.1 Creación de nicho

Un aspecto importante para la comunidad de software libre es incrementar la diversidad de miembros relevantes en el tiempo. Esta sub-característica indica básicamente la ventana de oportunidad de una comunidad para ser relevante en un nicho de mercado.

En la tabla que se muestra a continuación, se describen las métricas relacionadas con la sub-característica de creación de nicho.

Tabla 21: Lista de métricas para la sub-característica de creación de nicho. Lista obtenida de QuESO [23].

Métrica	Descripción	Fuentes	Método de obtención	Interpretación
Número de tipos de contexto en la aplicación de los proyectos de la comunidad	Saber si los proyectos de la comunidad tienen diferentes tipos de contexto en su aplicación.	Repositorios de software y blog.	Identificar las dependencias de los proyectos, buscar en el blog, identificar los contextos del proyecto.	Más es mejor, una gran variedad de contextos indica que se puede crear un nicho de uso.
Número de lenguajes naturales soportados	Ver si es una comunidad multilenguaje.	Blog	Identificar los diferentes lenguajes usados en la comunidad.	Más lenguajes mejor.
Variedad en tecnologías de los proyectos	Ver si los proyectos de una comunidad soportan varias tecnologías.	Repositorios de software y blog.	Identificar las diferentes tecnologías usadas en el desarrollo.	Mayor variedad de tecnologías, mejor.
Número de extensiones de plataforma	Cuántas extensiones de plataforma soporta la comunidad.	Repositorios de software	Obtener el número de extensiones de los repositorios de software.	Más es mejor. Cada extensión es una oportunidad para un potencial nicho.
Número de nichos de la comunidad	Cuántos nichos tiene la comunidad.	Blog	Encuestas y consultas a la base de datos.	Más es mejor.
Número de mercados donde la comunidad está participando	Comprobar si los proyectos de la comunidad están en diferentes mercados.	Repositorios de software	Identificar las dependencias de los proyectos.	Una mayor variedad, mejor.

3.1.3.2.2 Conocimiento de la comunidad

Dentro de una comunidad de software libre, el desarrollo se realiza de manera abierta y compartida entre los contribuidores. Estos contribuidores añaden conocimiento en forma de información, desde blog posts hasta manuales, con los que se crea una base de conocimiento.

A continuación, se muestran las métricas que afectan a esta sub-característica.

Tabla 22: Lista de métricas para la sub-característica de conocimiento de la comunidad. Lista obtenida de QuESO [23].

Métrica	Descripción	Fuentes	Método de obtención	Interpretación
Número de tipos de actividad	Cuántos tipos de actividades que tiene la comunidad.	Repositorios de software.	Obtener el tipo de actividad de los miembros mediante consultas a la base de datos y encuestas.	Más es mejor.
Número de artefactos de la comunidad	Comprobar si los contribuidores añaden conocimiento a la comunidad.	Blog.	Contar los artefactos (manuales, blog posts, traducciones, materiales de marketing, artículos científicos, etc) de la comunidad.	Más es mejor.

3.1.3.2.3 Vitalidad

La vitalidad es la habilidad de una comunidad para crecer y la viabilidad para conseguir dicho crecimiento.

En la tabla que se muestra a continuación, se describen las métricas relacionadas con la sub-característica de vitalidad.

Tabla 23: Lista de métricas para la sub-característica de vitalidad. Lista obtenida de QuESO [23].

Métrica	Descripción	Fuentes	Método de obtención	Interpretación
Liquidez de los partners	Saber si los partners pueden realizar las obligaciones a corto plazo.	Datos públicos de los partners	Obtener información financiera de los partners, calcular su liquidez LoP es la división de los activos a corto plazo entre las obligaciones a corto plazo	Mayor es mejor, si es menor que 1 es peligroso para el partners, valor entre 1 y 2 es normal y mayor de 2 está bien.
Porción de mercado de los proyectos de la comunidad	Cómo está posicionado en el mercado cada proyecto.	Blog, listas de correo y repositorios de software.	Hacer encuestas a usuarios finales para recoger conocimiento e información relevante como reportes de mercados, evaluaciones open-source, y datos de popularidad. Finalmente se agregan los datos.	Más es mejor

Solvencia de los partners	Saber si los partners pueden pagar sus deudas.	Datos públicos de los partners	Obtener información financiera de los partners, contar el número de partners con una solvencia menor que uno, siendo la solvencia calculada como la división de las acciones de un partners entre sus deudas.	Menor es mejor.
Desarrollo de activos de los partners	Cómo es la productividad de los partners.	Datos públicos de los partners	Obtener la productividad sobre el tiempo con los datos financieros de cada partner.	Mayor es mejor.
Obsolescencia limitada	Saber si la infraestructura de la comunidad está obsoleta.	Encuestas, consultas a webs.	Obtener información sobre las tecnologías usadas por la comunidad.	Si no está obsoleta, mejor.
Continuidad de la experiencia de usuario y casos de uso.	Saber cómo evoluciona la comunidad como respuesta a los cambios tecnológicos.	Blog y web de la comunidad	Obtener una lista con información de las tecnologías de los proyectos y cuándo fueron publicadas. Comparar cómo las versiones evolucionan con la plataforma de la comunidad, lenguajes de programación y sistemas operativos.	Evolucionar es mejor que cambiar abruptamente.
Aceptación de la comunidad	Cómo es la aceptación de la comunidad a organizaciones comerciales	Blog y partners.	Encuestas	Mayor aceptación, mejor.
Número de usuarios pasivos	Es igual a la medida de la sub-categoría Tamaño.	-	-	Mayor número de usuarios pasivos, mejor.
Número de nuevas comunidades	Saber si se crean nuevas comunidades en la comunidad.	Listas de correo y repositorios de software.	Contar las comunidades nuevas en un periodo de tiempo.	Más es mejor.
Colaboración e integrabilidad de la comunidad.	Cómo es la integrabilidad de la comunidad.	Repositorios de software.	Mediante encuestas intentar obtener información sobre el modelo de colaboración.	Mayer nivel de integración, mejor.

3.1.3.2.4 Confiabilidad

La confiabilidad es una sub-característica que se refiere a la colaboración confiable entre partners con una responsabilidad compartida en crear un ecosistema open-source. En esta confiabilidad hay medidas relacionadas con los modelos de banca rota (como la

puntuación Z y el modelo Zeta) que son adecuadas porque tienen en cuenta la supervivencia a corto y largo plazo.

En la tabla que se muestra a continuación, se describen las métricas relacionadas con la sub-característica de confiabilidad.

Tabla 24: Lista de métricas para la sub-característica de confiabilidad. Lista obtenida de QuESO [23].

Métrica	Descripción	Fuentes	Método de obtención	Interpretación
Puntuación ZETA de los partners	Cuál es la puntuación de peligro de bancarrota de los partners.	Datos financieros públicos de los partners.	Emplear la fórmula presente en QuESO [23].	Mayor puntuación, mejor.
Edad de la comunidad.	Cómo es de vieja el ecosistema.	Repositorios de software.	Diferencia temporal entre el primer y el último commit.	Mayor edad, mejor.
Número de patentes de los partners.	Cuántas patentes tienen los partners de la comunidad.	Partners	Mediante encuentas, obtener el número de patentes que tienen.	Más es mejor.
Reputación y opiniones de contribuidores	Cuál es la reputación de los contribuidores de la comunidad.	Contribuidores y páginas web	Siendo TFR el intervalo de tiempo entre una pregunta i y la primera respuesta y n el número de preguntas, entonces: $ET = \frac{\sum^{n} TRF(r_i)}{n}$	Mayor reputación, mejor.

3.1.3.3 Interés comercial

En el caso de proyectos de software libre, una característica relevante para su supervivencia a largo plazo y que afecta a la calidad de la red del ecosistema sería el interés comercial que suscita dicho software. Muchas empresas están interesadas en emplear un proyecto de software libre en sus productos, para ofrecer servicios alrededor tales como soporte, formación, mantenimiento, desarrollo de nuevas características…

En muchas ocasiones si un determinado software tiene éxito comercial en algún sector, éste atrae más empresas interesadas en su evolución que pueden ser diferentes a las que comenzaron el proyecto, ya que se beneficiarían de un trabajo colaborativo, abierto y con posibilidad de influenciar el desarrollo del software [49]. Por ejemplo, empresas consultoras colaborarán en mejorar el proyecto de software libre que usan sus clientes para así ofrecer servicios de mantenimiento, de desarrollo de nuevas características, formación, etcétera. Otras empresas pueden utilizar el software como apoyo para sus propios productos, como ejemplo están los navegadores web que se usa en sistemas de Smart-TV, donde se usan para poder ofrecer funcionalidad web a las televisiones y para soportas aplicaciones en lenguaje HTML5.

Algunos aspectos de esta característica, como son los servicios alrededor de dicha comunidad, son similares a algunos ya presentes en OpenBRR. Desgraciadamente, el resto modelos analizados que son posteriores en el tiempo a OpenBRR ya no los incluyen.

Como esta característica no está presente en el modelo QuESO, esta es una aportación original del trabajo final de máster a dicho modelo.

A continuación, se describe una lista de métricas que afectan a esta característica.

Tabla 25: Lista de métricas para la característica de interés comercial.

Métrica	Descripción	Fuentes	Método de obtención	Interpretación
Variedad de los partners de la comunidad	Misma métrica que la disponible en la sub-característica de Heterogeneidad.	-	-	--
Productos software	Averiguar si el software se embebe en productos software de terceros	Webs, buscadores web.	Encontrar las empresas que utilizan los productos software de la comunidad en sus propios productos, y contar los mismos.	Mayor número de productos, mejor.
Ámbito de uso	Averiguar si el software es usado en un ámbito concreto	Webs, buscadores web.	Clasificar las regiones de los usuarios y desarrolladores para ver si el software se utiliza en un ámbito local, regional, nacional o internacional.	Mayor ámbito, mejor.
Servicios alrededor de la comunidad	Averiguar si hay servicios ofertados por compañías alrededor del software	Webs, buscadores web	Clasificar los servicios ofertados: formación, desarrollo, mantenimiento, soporte, etc.	Mayor número de servicios, mejor

3.1.4 Perfiles de evaluación en el modelo

Una de las mejoras propuestas al modelo de QuESO es la utilización de perfiles de evaluación en el modelo ya que hay datos que pueden no son relevantes o pueden no estar disponibles en la evaluación realizada con dicho modelo y puede que no sean tan relevantes en la evaluación final.

Un ejemplo de una evaluación que sería afectada por la falta de datos sería la evaluación de un proyecto de software libre por terceras personas ajenas a la comunidad, con lo que los datos financieros de los partners de la comunidad no estarían disponibles excepto en los casos que esta información sea pública. Otro caso distinto sería la evaluación de distintos softwares por parte de usuarios concienciados por la ética y la filosofía del software libre, mientras que la parte de empresas en el ecosistema no les interesa lo más mínimo; otro ejemplo sería una empresa que busca cuál es la mejor comunidad para contribuir, influir en el desarrollo y buscar talento para su contratación, entre otros muchos ejemplos. Es, por tanto, tarea del evaluador averiguar cual es el interés subyacente en la realización de la evaluación de software libre.

Para proporcionar las herramientas necesarias para este caso, en este trabajo fin de máster se propone la posibilidad de definir perfiles, que ya se utilizaban en el modelo SQO-OSS [17]. El uso de los perfiles no es obligatorio en absoluto, pero sí recomendable en los casos donde el interés subyacente en la evaluación permita reducir el peso de, o incluso ignorar, algunas métricas del modelo, lo que ayuda a simplificar la evaluación y centrarse en las partes que interesan al evaluador.

3.1.4.1 Creación de nuevos perfiles personalizados

Cualquier evaluador puede proponer un nuevo perfil en base a los intereses que persigue con la evaluación de software. En este caso, se pueden emplear distintos pesos para cada una de las métricas del modelo, para así obtener un resultado que sea más interesante para sus propósitos.

También puede darse el caso que para realizar la comparación entre evaluaciones de distintos proyectos, no haya los suficientes datos o herramientas necesarias para obtener resultados de algunas métricas. En este caso, se podría plantear la posibilidad de ignorar la correspondiente métrica si el evaluador cree que no es muy relevante para obtener el resultado final, o se podría tener en cuenta la existencia de dicha métrica como un punto diferenciador dentro de la comparación cuando dos proyectos tengan valores muy similares en el resto de las métricas evaluadas.

3.1.4.2 Ejemplos de perfiles propuestos

Se han creado dos perfiles de ejemplo para ayudar a la creación de nuevos perfiles o facilitar la evaluación de los proyectos de software libre en los casos propuestos. Los valores propuestos son arbitrarios y son motivados por la experiencia laboral del autor de este trabajo fin de máster. Los valores se presentarán en forma de porcentaje que se debe aplicar a cada valor individual.

Perfil para personas concienciadas con el software libre

El primer caso se basa en la evaluación de distintos softwares por parte de usuarios concienciados por la ética y la filosofía del software libre, donde el interés comercial, el número de patentes y otros aspectos similares son secundarios. Esta motivación social no solamente afecta a usuarios particulares, sino a compañías que comparten valores y normas con la comunidad [49].

En el caso de la característica Apertura, donde se tienen en cuenta las licencias del software libre, la ética que tiene la comunidad y el fomento a la participación es un aspecto muy a valorar en este caso, ya que implica que la comunidad está implicada con el software libre y tiene unos valores que intentan mejorar en lo posible al conjunto de miembros que la componen. La elección de una buena licencia puede afectar a la motivación de los usuarios a usar un determinado software, así como que la comunidad esté controlada principalmente por organizaciones privadas y comerciales [50].

Como en cualquier software, la capacidad de mantenimiento y la sostenibilidad son aspectos muy relevantes ya que, sin ellos, el propio software no podría sobrevivir mucho tiempo. En cambio, la dimensión de la calidad de red de ecosistema no es relevante para una persona preocupada puramente por el software libre, ya que prioriza otros aspectos

siendo ésta la razón por la que características como la salud de la red y la salud de los recursos tienen pesos bajos.

Hay métricas que pudieran tener sentido que fueran negativas, por ejemplo si en un proyecto de software libre hubiera patentes que fueran propiedad de empresas u otros organismos es algo que puntuaría negativo para un perfil como este (ver capítulo 16, "The Danger of Software Patents" en el libro de ensayos de Richard Stallman [51]).

El interés comercial es absolutamente ignorado en este perfil ya que no es relevante. Ejemplos de personas interesadas en el software libre la tenemos en los usuarios de distribuciones de software puramente libres que son recomendadas por la Free Software Foundation, donde la parte comercial de las mismas no existe o es insignificante.

A continuación, se muestra una tabla con los pesos asignados para cada una de las (sub)características para este perfil.

Tabla 26: Pesos para las métricas de la dimensión Calidad de Comunidad para el primer perfil de ejemplo.

Característica	Subcaracterística	Medidas	Peso
Apertura	Licencias	Licencias de software	1
		Acuerdo de licencia de contribuidor	1
		Módulos propietarios	1
	Ética	Código de conducta	1
		Medidas de integración de minorías	1
		Valores	1
	Participación	Programas de captación de voluntariado	1
		Toma de decisiones en la comunidad	1
		Organización jerárquica de la comunidad	1
Capacidad de mantenimiento	Tamaño	Número de partners	1
		Número de usuarios pasivos	1
		Número de contribuidores	1
		Tamaño de la red de comunidad	1
		Número de miembros de la comunidad del proyecto	1
	Actividad	Actividad en el bug tracker	1
		Correlación entre comunicación y resolución de bug	1
		Fecha del último commit	1
		Historial de versiones	1
		Fecha de la última release	1
		Número de ficheros cambiados	1
		Número de ficheros por versión	1
		Actividad de los contribuidores	1
		Ratio de commits	1
		Punto de declive	1
		Punto cúspide	1
		Período de actividad de la comunidad	1
		Tiempos de la comunidad	1
		Número de eventos	1
Madurez de procesos			1
Sostenibilidad	Heterogeneidad	Distribución geográfica de miembros	1
		Tipos de actividad de miembros	1
		Variedad de proyectos	1
		Variedad de partners de la comunidad	1
		Distribución en organizaciones de los miembros de la comunidad	1
	Habilidad de regeneración	Ratio de supervivencia de contribuidores	1

		Ratio de miembros nuevos	1
		Ratio de contribuidores nuevos	1
	Equilibrio de esfuerzos	Ratio de tiempo entre commits	1
		Implicación de los proyectos de la comunidad	1
		Características estadísticas de los commits	1
		Ratio de actividades de la comunidad	1
		Ratio de miembros en sub-comunidades	1
		Ratio releases por desarrollador	1
	Equilibrio de conocimiento	Conocimiento de contribuidores	1
		Ratio de longevidad en los contribuidores	1
		Ratio de experiencia por contribuidor	1
		Número de proyectos por contribuidor	1
	Visibilidad	Número de miembros pidiendo nuevas características	1
		Número de ofertas de empleo	1
		Número de descargas	1
		Número de subscriptores a las listas de correo	1
		Número de usuarios pasivos	1
		Número de publicaciones científicas	1
		Publicaciones en webs y social media	1
		Número de patentes	-1
		Número de eventos	1
		Reputación y opiniones de contribuidores	1
		Distribución geográfica de miembros	1
		Aceptación de la comunidad	1
		Número de páginas referenciando la comunidad	1
	Cohesión de la comunidad	Ratio de centralidad de la comunidad	1
		Clúster de colaboración de desarrolladores	1
		Enlaces a otras sub-comunidades	1
		Grado de presencia de actores clave	1

Tabla 27: Pesos para las métricas de la dimensión de Calidad de la red del ecosistema para el primer perfil de ejemplo.

Característica	Subcaracterística	Medidas	Peso
Salud de la red	Cohesión del ecosistema	Número de nodos para desconectar el ecosistema	1
		Grado de *outdegree* de actores clave	1
		Coeficiente de clústering de la comunidad	0
		Número de conexiones de los partners	0
	Consistencia de la información	Sinónimos en el mapa de vocabulario	1
		Análisis de sentimiento	1
	Evolución de las sinergias	Distribución de los partners	1
		Popularidad de la comunidad	1
		Partnerships e integrabilidad de los proyectos de la comunidad	0
		Reciprocidad de la comunidad	1
	Habilidad de interrelación	Evolución de la conectividad	0
		Evolución de la conectividad de los partners con otros miembros de la comunidad	0
		Evolución de la centralidad de la comunidad	0
Salud de recursos	Creación de nicho	Número de tipos de contexto en la aplicación de los proyectos de la comunidad	0
		Número de lenguajes naturales soportados	1
		Variedad en tecnologías de los proyectos	1
		Número de extensiones de plataforma	1
		Número de nichos de la comunidad	0
		Número de mercados donde la comunidad está participando	0
	Conocimiento de la comunidad	Número de tipos de actividad	1
		Número de artefactos de la comunidad	1
	Vitalidad	Liquidez de los partners	0

		Porción de mercado de los proyectos de la comunidad	0
		Solvencia de los partners	0
		Desarrollo de activos de los partners	0
		Obsolescencia limitada	1
		Continuidad de la experiencia de usuario y casos de uso	1
		Aceptación de la comunidad	1
		Número de usuarios pasivos	1
		Número de nuevas comunidades	1
		Colaboración e integrabilidad de la comunidad	1
	Confiabilidad	Puntuación ZETA de los partners	0
		Edad de la comunidad	1
		Número de patentes de los partners	1
		Reputación y opiniones de contribuidores	1
Interés comercial		Variedad de partners de la comunidad	0
		Productos software	0
		Ámbitos de uso	0
		Servicios alrededor de la comunidad	0

Perfil para empresas centradas en el beneficio propio

El segundo caso se basa en la evaluación de distintos productos software por parte de usuarios (normalmente empresas) que buscan el mejor software que cubra sus necesidades y que puedan integrar en sus productos.

Es en este caso donde los aspectos puramente económicos y tecnológicos son especialmente importantes [49], mientras que la ética de la comunidad no es relevante. No obstante, sí que hay un par de sub-características de la característica Apertura que podrían interesar a una empresa centrada en buscar un software para cubrir una necesidad concreta: Licencias y Participación. La parte de Licencia es muy interesante para saber si puede utilizar el producto o necesita de algún tipo de módulo propietario o una licencia comercial o cualquier otro requerimiento legal que afecte a su elección. Además, en la parte de Participación, la empresa podría considerar aquellos proyectos en los que puede influir en las decisiones técnicas [49].

Características como la capacidad de mantenimiento, madurez de procesos y sostenibilidad son muy importantes para las empresas que busquen usar un software, ya que eso garantiza una estabilidad y un desarrollo futuro. La salud de la red del ecosistema es importante: si la comunidad está cohesionada, se emplea un lenguaje de dominio común, hay interrelación y cooperación entre los distintos miembros y hay sinergias que puedan ofrecer nuevas oportunidades de negocio en la comunidad entre otros aspectos; por tanto, la empresa que busque seleccionar un software para cubrir sus necesidades estará tentada a usarlo frente a otros que no sea así, de ahí que los pesos se mantengan los máximos. De manera similar ocurre en la parte de salud de los recursos ya que, si hay partners que tienen una estabilidad financiera sólida, se confía que ésta perdure en el tiempo y la comunidad genera conocimiento útil, todo ello redunda en una comunidad saludable y preferible a otra que no lo tenga. Finalmente, el interés comercial es importante, si hay empresas que usan este software en productos similares y de manera satisfactoria, la empresa tenderá a usarlos ya que podría convertirse en el estándar de facto y concentraría la mayor parte de la innovación frente a otros productos competidores.

A continuación, se muestra una tabla con los pesos asignados para cada una de las (sub)características teniendo en cuenta las necesidades de este perfil.

Tabla 28: Pesos para las métricas de la dimensión Calidad de Comunidad para el segundo perfil de ejemplo.

Característica	Subcaracterística	Medidas	Peso
Apertura	Licencias	Licencias de software	1
		Acuerdo de licencia de contribuidor	1
		Módulos propietarios	1
	Ética	Código de conducta	0
		Medidas de integración de minorías	0
		Valores	0
	Participación	Programas de captación de voluntariado	0
		Toma de decisiones en la comunidad	1
		Organización jerárquica de la comunidad	0
Capacidad de mantenimiento	Tamaño	Número de partners	1
		Número de usuarios pasivos	1
		Número de contribuidores	1
		Tamaño de la red de comunidad	1
		Número de miembros de la comunidad del proyecto	1
	Actividad	Actividad en el bug tracker	1
		Correlación entre comunicación y resolución de bug	1
		Fecha del último commit	1
		Historial de versiones	1
		Fecha de la última release	1
		Número de ficheros cambiados	1
		Número de ficheros por versión	1
		Actividad de los contribuidores	1
		Ratio de commits	1
		Punto de declive	1
		Punto cúspide	1
		Período de actividad de la comunidad	1
		Tiempos de la comunidad	1
		Número de eventos	1
Madurez de procesos			1
Sostenibilidad	Heterogeneidad	Distribución geográfica de miembros	1
		Tipos de actividad de miembros	1
		Variedad de proyectos	1
		Variedad de partners de la comunidad	1
		Distribución en organizaciones de los miembros de la comunidad	1
	Habilidad de regeneración	Ratio de supervivencia de contribuidores	1
		Ratio de miembros nuevos	1
		Ratio de contribuidores nuevos	1
	Equilibrio de esfuerzos	Ratio de tiempo entre commits	1
		Implicación de los proyectos de la comunidad	1
		Características estadísticas de los commits	1
		Ratio de actividades de la comunidad	1
		Ratio de miembros en sub-comunidades	1
		Ratio releases por desarrollador	1
	Equilibrio de conocimiento	Conocimiento de contribuidores	1
		Ratio de longevidad en los contribuidores	1
		Ratio de experiencia por contribuidor	1
		Número de proyectos por contribuidor	1
	Visibilidad	Número de miembros pidiendo nuevas características	1
		Número de ofertas de empleo	1
		Número de descargas	1

		Número de subscriptores a las listas de correo	1
		Número de usuarios pasivos	1
		Número de publicaciones científicas	1
		Publicaciones en webs y social media	1
		Número de patentes	-1
		Número de eventos	1
		Reputación y opiniones de contribuidores	1
		Distribución geográfica de miembros	1
		Aceptación de la comunidad	1
		Número de páginas referenciando la comunidad	1
	Cohesión de la comunidad	Ratio de centralidad de la comunidad	1
		Clúster de colaboración de desarrolladores	1
		Enlaces a otras sub-comunidades	1
		Grado de presencia de actores clave	1

Tabla 29: Pesos para las métricas de la dimensión de Calidad de la red del ecosistema para el segundo perfil de ejemplo.

Característica	Subcaracterística	Medidas	Peso
Salud de la red	Cohesión del ecosistema	Número de nodos para desconectar el ecosistema	1
		Grado de *outdegree* de actores clave	1
		Coeficiente de clústering de la comunidad	1
		Número de conexiones de los partners	1
	Consistencia de la información	Sinónimos en el mapa de vocabulario	1
		Análisis de sentimiento	1
	Evolución de las sinergias	Distribución de los partners	1
		Popularidad de la comunidad	1
		Partnerships e integrabilidad de los proyectos de la comunidad	1
		Reciprocidad de la comunidad	1
	Habilidad de interrelación	Evolución de la conectividad	1
		Evolución de la conectividad de los partners con otros miembros de la comunidad	1
		Evolución de la centralidad de la comunidad	1
Salud de recursos	Creación de nicho	Número de tipos de contexto en la aplicación de los proyectos de la comunidad	1
		Número de lenguajes naturales soportados	1
		Variedad en tecnologías de los proyectos	1
		Número de extensiones de plataforma	1
		Número de nichos de la comunidad	1
		Número de mercados donde la comunidad está participando	1
	Conocimiento de la comunidad	Número de tipos de actividad	1
		Número de artefactos de la comunidad	1
	Vitalidad	Liquidez de los partners	1
		Porción de mercado de los proyectos de la comunidad	1
		Solvencia de los partners	1
		Desarrollo de activos de los partners	1
		Obsolescencia limitada	1
		Continuidad de la experiencia de usuario y casos de uso	1
		Aceptación de la comunidad	1
		Número de usuarios pasivos	1
		Número de nuevas comunidades	1
		Colaboración e integrabilidad de la comunidad	1
	Confiabilidad	Puntuación ZETA de los partners	1
		Edad de la comunidad	1
		Número de patentes de los partners	1
		Reputación y opiniones de contribuidores	1
Interés comercial		Variedad de partners de la comunidad	1
		Productos software	1
		Ámbitos de uso	1
		Servicios alrededor de la comunidad	1

3.2 Ejemplo de aplicación del modelo

3.2.1 Descripción del proyecto de software escogido

Es interesante observar cómo se aplica el modelo QuESO con las modificaciones propuestas en este trabajo fin de máster en un caso real. Dentro de la gran variedad de proyectos de software libre, se va a escoger Mesa[21], que es una librería de gráficos 3d con licencia libre que implementa las API estándares de OpenGL y Vulkan para una gran variedad de tarjetas gráficas. En la Ilustración 28 se muestra el *stack* gráfico de GNU/Linux y el lugar que ocupa Mesa cuando una aplicación quiere realizar un pintado de gráficos 3d acelerados por hardware.

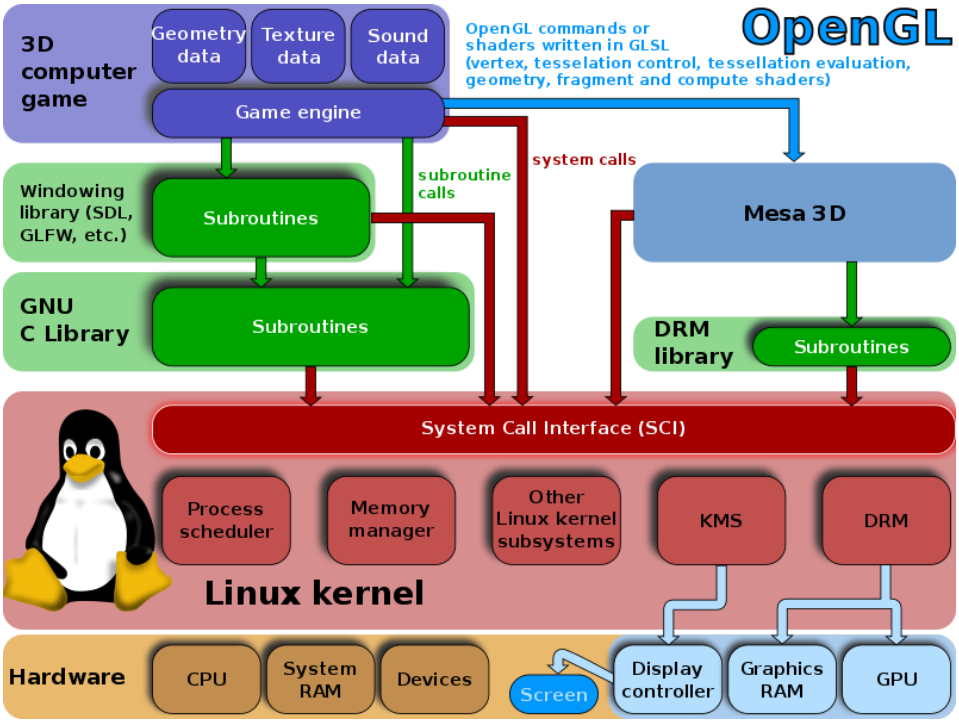


Ilustración 28: Stack gráfico de GNU/Linux. Imagen obtenida de la wikipedia (licencia CC-BY-SA 3.0).

Mesa implementa una serie de drivers para realizar gráficos 3d de manera acelerada sobre hardware real, ejemplos de fabricantes de tarjetas gráficas soportadas son Intel, AMD, NVIDIA, Qualcomm, Broadcom entre otros. Además, se implementan varios drivers que realizan una emulación por software tales como los utilizados por VMWare en sus máquinas virtuales, y otros como swrast o llvmpipe.

Mesa es ampliamente utilizada en sistemas operativos libres como GNU/Linux, BSD, Android, ChromeOS y otros, dada su amplio soporte de tarjetas gráficas, su licencia abierta (licencia MIT principalmente[22]) y la activa comunidad de desarrolladores que tiene, donde muchas empresas colaboran en la comunidad ya sea en forma de aportación

[21] http://www.mesa3d.org
[22] https://www.mesa3d.org/license.html

económica como en la contratación de desarrolladores para realizar mantenimiento de la librería y el desarrollo de nuevas características.

Ilustración 29: Listado de algunas empresas que forman parte de la comunidad de Mesa.

Mesa es un ejemplo muy relevante de lo que puede hacer una comunidad de software libre que tiene empresas colaborando a su alrededor de manera cooperativa, así como es un ejemplo de cómo se puede dividir en sub-comunidades, siendo representadas por cada uno de los drivers que la componen. Mesa fue creada en agosto de 1993 por Brian Paul y cuenta en la actualidad con decenas de contribuidores activos, lo cual es interesante desde un punto de vista de obtención de datos relevantes para realizar una evaluación del modelo propuesto en este trabajo final de máster.

3.2.2 Metodología empleada

En este trabajo final de máster no se realizará un estudio de la calidad de plataforma ya que para ello se emplearía el estándar ISO-25010, ni el análisis de la madurez de procesos que sería hecho con cualquiera de los métodos de evaluación (por ejemplo, CMMI-DEV), ya que ambos no son el motivo principal de este trabajo. Para el resto de las dimensiones (calidad de comunidad y calidad de red del ecosistema), se van a obtener medidas siguiendo las métricas definidas para cada una de las sub-características del modelo propuesto.

Se ha empleado como herramienta de apoyo una hoja de cálculo, pero se incluirán los datos recogidos en este trabajo final de máster junto con una breve descripción de los resultados obtenidos, cuando éstos sean relevantes. En los casos en que no hubiera una herramienta adecuada o faltara información, se marca la casilla de la correspondiente métrica en color rojo y se indica en el apartado de "notas" la razón del mismo.

A la hora de realizar las mediciones de métricas relacionadas con estadísticas sobre el repositorio de software, se emplearon los distintos comandos de la herramienta *git*, scripts

escritos en *bash,* junto con el apoyo de una herramienta matemática para realizar los cálculos necesarios llamada *Octave*[23], cuya licencia es libre y es compatible con *Matlab*.

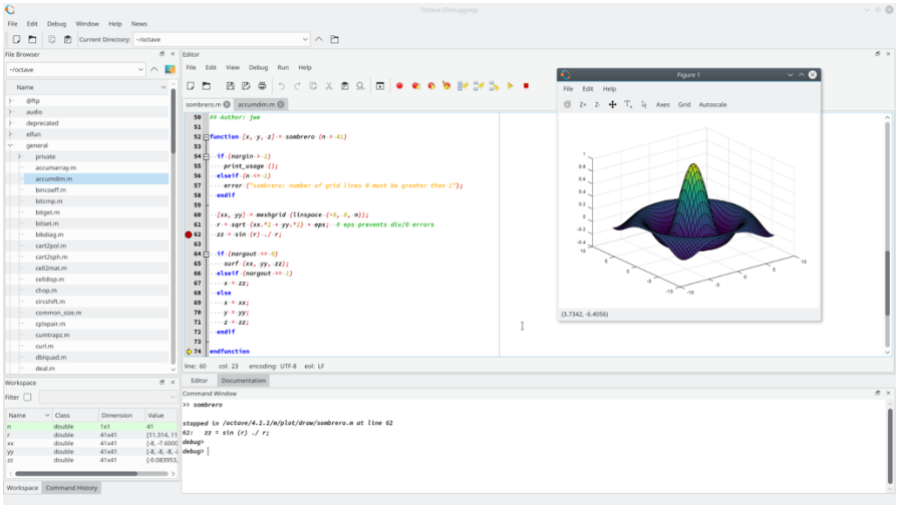


Ilustración 30: Octave, la aplicación matemática libre compatible con Matlab.

3.2.3 Calidad de plataforma

No se ha realizado ningún estudio de la dimensión de la calidad de plataforma en este caso de estudio. Para realizarlo, se emplearían las métricas definidas en el estándar ISO-25000 y se calcularía siguiendo el método empleado por dicho estándar. El objetivo de este trabajo final de máster es analizar las particularidades de los proyectos de software libre y cómo mejorar un modelo ya existente para tenerlas en cuenta.

3.2.4 Calidad de comunidad

La dimensión de Calidad de Comunidad está dividida en cuatro características: Apertura, Capacidad de Mantenimiento, Madurez de Procesos y Sostenibilidad.

La característica de Apertura se subdivide, a su vez, en tres sub-características: Licencias, Ética y Participación. Para todas ellas, se han recogido datos de todas las métricas planteadas. Cabe destacar que, para todas ellas, se pudieron obtener los valores mediante el análisis de la página web del proyecto, así como un seguimiento de las discusiones en las listas de correo.

La característica de Capacidad de Mantenimiento se divide en Tamaño y Actividad. Es en estas sub-características donde se encuentra las primeras métricas donde no se disponía de la información o que no se disponían de las herramientas adecuadas para su cálculo. Estos dos problemas se irán repitiendo a lo largo de otras métricas en la realización de este caso de estudio.

Por ejemplo, en el caso de la métrica que mide el número de usuarios pasivos no se dispone de acceso a la base de datos de la página web de descargas. Sin embargo, se puede estimar que su número andará por millones ya que Mesa se encuentra instalado por defecto en prácticamente todas las distribuciones de GNU/Linux que dispongan de

[23] http://www.octave.org

escritorio gráfico. Debido a las particularidades de los sistemas de paquetes de las distribuciones GNU/Linux, donde cada distribución tiene sus propias estadísticas y no se comparten con terceros, medir el número de usuarios pasivos mediante las estadísticas de la web de descarga, si se tuviera acceso a las mismas, no se ajustaría a la realidad.

Un ejemplo de falta de herramienta adecuada se encuentra en la métrica que calcula la correlación entre comunicación y resolución del bug. Para realizar adecuadamente esta métrica, se requeriría una copia de la base de datos del sistema de seguimientos de bugs para así calcular la correlación sobre el número de bugs cerrados, así como un analizador de la lista de correo de la comunidad para comprobar cuántas comunicaciones se han hecho a lo largo de ciertos periodos de tiempo y cómo están correladas entre ellas. Idealmente, se dispondría de una copia del archivo de la lista de correo para poder hacer este análisis offline.

No obstante, la mayoría de las métricas pudieron ser obtenidas mediante el empleo de comandos por consola *bash*, junto con el uso de sub-comandos presentes en la aplicación de control de versiones que usa el proyecto, llamada *git*[24].

La característica Madurez de Procesos no fue analizada en este caso de estudio ya que no presenta diferencias a lo que se haría con un software propietario. Si se fuera a realizar, se evaluaría la Madurez de Procesos mediante el empleo de un modelo especializado, como puede ser CMMI-DEV.

La característica de Sostenibilidad se subdivide en Heterogeneidad, Habilidad de regeneración, Equilibrio de esfuerzos, Equilibrio de conocimiento, Visibilidad y Cohesión de la comunidad. Al igual que antes, algunas métricas no se pudieron evaluar por falta de herramienta adecuada o falta de información (más información en la Tabla 30), aunque sí que hay datos interesantes que se pueden comentar.

En el caso de la distribución geográfica de los contribuidores, se empleó el análisis de las fechas de los *commits* del repositorio de software, donde se indica en qué día y hora se ha realizado junto con la zona horaria. Esta manera de obtener la distribución geográfica se basa en la fecha y hora establecida en el sistema operativo que utiliza el desarrollador, no es la manera más precisa pero sí que puede ayudar a obtener una idea de la distribución geográfica de los desarrolladores [52].

Es esta zona horaria la parte de la información que más interesa, ya que *git* emplea la que tiene el reloj del sistema donde está instalado y que se ha usado para realizar el *commit*.

[24] https://git-scm.com

Ilustración 31: Distribución por zona horaria de los commits realizados en Mesa.

Como se puede observar en la Ilustración 31, los desarrolladores están distribuidos por todo el mundo, pero principalmente se concentran en América y Europa. Hay un considerable número menor de desarrolladores que viven en Asia y Oceanía.

El número de contribuidores únicos desde el inicio del proyecto es de 867 personas. Sin embargo, es interesante analizar cómo es la evolución a lo largo del tiempo del número de desarrolladores activos (Ilustración 32). En la parte inferior de la Ilustración 32, se puede observar la evolución desde el año 1998 (año donde se tienen los primeros registros en el repositorio *git*), mientras que la parte superior de la imagen es una ampliación para ver los últimos 3 años del proyecto. Como se puede observar, el número de contribuidores activos del proyecto se ha estabilizado en un número entre 60 y 80 contribuidores.

Ilustración 32: Evolución del número de contribuidores activos desde 1998 hasta hoy. Datos obtenidos de la herramienta Openhub de la compañía Black Duck.

Para el caso de número de miembros de la pidiendo nuevas características, se creó un informe específico con la herramienta de seguimiento de errores (*bug tracker* o *issue tracker*, en inglés) llamado Bugzilla que tiene la comunidad. El número de miembros es 15 y se obtuvo a través de contar el número de autores de peticiones en dicho informe (ver Ilustración 33)

		Total
bob@o-hand.com	1	1
chadversary@chromium.org	1	1
e.singularitycat@gmail.com	1	1
germano.massullo@gmail.com	1	1
idr@freedesktop.org	1	1
jason@jlekstrand.net	4	4
jljusten@gmail.com	2	2
Reporter k.philipp@gmail.com	1	1
kenneth@whitecape.org	1	1
mercen45@gmail.com	1	1
pauk.denis@gmail.com	1	1
phillip.m.jordan@gmail.com	1	1
plaes@plaes.org	1	1
stewartlittle@outlook.com	1	1
vedran@miletic.net	2	2
Total	20	20

Ilustración 33: Número peticiones de nuevas características en un año (12 de Abril de 2017 a 12 de Abril 2018) en la herramienta de seguimiento de errores de Mesa. Captura de pantalla realizada el 12 de Abril de 2018.

Para la sub-característica de la Cohesión de la Comunidad no se dispone de una herramienta libre adecuada para realizar un grafo de una red para cada una de las métricas sugeridas y que utilice tanto el repositorio de software como las listas de correo como fuentes de entrada.

En la Tabla 30 se adjuntan todos los valores obtenidos para cada una de las métricas de la dimensión de Calidad de Comunidad.

Tabla 30: Valores obtenidos para la dimensión de Calidad de Comunidad para el proyecto Mesa.

Característica	Subcaracterística	Medidas	Valor	Notas
Apertura	Licencias	Licencias de software	3	MIT licence, SGI Free Software License B, boost (licencia permisiva)
		Acuerdo de licencia de contribuidor	1	El acuerdo es que el contribuidor esté de acuerdo con la licencia
		Módulos propietarios	0	La patente sobre el formato de texturas comprimidas llamado S3TC ha expirado, no hay módulos propietarios.
	Ética	Código de conducta	1	https://www.freedesktop.org/wiki/CodeOfConduct/
		Medidas de integración de minorías	2	https://www.x.org/wiki/XorgEVoC/, for students, travel sponsorship for speakers and students -> https://www.x.org/wiki/Events/

		Valores	16	https://www.freedesktop.org/wiki/MissionStatement/
	Participación	Programas de captación de voluntariado	2	Google Summer of Code https://www.x.org/wiki/XorgEVoC/
		Toma de decisiones en la comunidad	7	Discusión abierta en la lista de correo hasta llegar a un acuerdo, se suele necesitar mostrar código para discutir la propuesta. No obstante, se centra principalmente entre desarrolladores y no hay votaciones
		Organización jerárquica de la comunidad	Sí (inclusiva y abierta)	Hay usuarios y commiters. Cualquier persona puede hacer revisiones. Para ser commiter, se requiere una decena de commits previos (inclusiva y abierta)
Capacidad de mantenimiento	Tamaño	Número de partners	11	AMD, Collabora, Feral Interactive, Google, Intel, Igalia, NVIDIA, Red Hat, Samsung, Valve, VMware
		Número de usuarios pasivos		Millones de usuarios, todas las distribuciones de Linux con entorno gráfico tienen Mesa instalado. No hay acceso a los datos de descarga de la web
		Número de contribuidores	867	867 contribuidores (12/04/2018) totales, 130 contribuidores desde enero 2018
		Tamaño de la red de comunidad		Falta herramienta adecuada
		Número de miembros de la comunidad del proyecto	1210	1210 personas subscritas a la lista de desarrollo (mesa-dev) a día 12/04/2018
	Actividad	Actividad en el bug tracker	11868	Número de comentarios escritos en el bugzilla en el periodo de un año para todos los componentes de Mesa
		Correlación entre comunicación y resolución de bug		Coefiente de correlación de Pearson, hay calcularlo sobre el número de bugs cerrados y las comunicaciones por listas de correo. Falta herramienta adecuada
		Fecha del último commit	Hoy	Hoy, 10am UTC+2 (12/04/2018)
		Historial de versiones	4	Hay una versión mayor por trimestre (4 al año), obtenido de la página web

		Fecha de la última release	27 de marzo de 2018	Versión mayor, 18.0 (27 de marzo) (dato del 12/04/2018), versión menor 17.3.8 (3 de Abril)
		Número de ficheros cambiados	2	Un año de datos: 12 abril 2017 a 12 de abril de 2018.
		Número de ficheros por versión	5857	Version 18.0.
		Actividad de los contribuidores	42	
		Ratio de commits	20 años	14 de Febrero de 1998 es la fecha del primer commit.
		Punto de declive	Julio de 2017	El número máximo de contribuidores de manera histórica son 90, desde Julio de 2017 no se supera el 80% de esta marca ininterrupidamente.
		Punto cúspide	Marzo de 2017	Número máximo de commits en un mes. No es el número de emails, pero creo que el valor puede ser el mismo.
		Período de actividad de la comunidad	4 meses	Aunque la comunidad es estable en el tiempo desde entonces. No hubo huecos en los desarrollos desde hace 19 años (ver gráfica en el apartado del TFM correspondiente)
		Tiempos de la comunidad		Falta herramienta adecuada
		Número de eventos	2	X.org Developer's Conference (anual), FOSDEM's graphics devroom (anual)
Madurez de procesos				Se aplicaría un modelo de evaluación de madurez de procesos. Por ejemplo, CMMI.
Sostenibilidad	Heterogeneidad	Distribución geográfica de miembros	Todo el mundo	Principalmente Europa y América
		Tipos de actividad de miembros		Falta herramienta adecuada
		Variedad de proyectos	9	Drivers de Intel, AMD, NVIDIA, Imagination Technologies, Broadcom, Qualcomm, Software renderers, OpenSWR, Vmware virtual GPU.
		Variedad de partners de la comunidad		Partners privados centrados en empresas TIC tanto de fabricación de HW como desarrolladoras de SW. No se realizaron encuestas

		Distribución en organizaciones de los miembros de la comunidad	74	74 dominios de correo, pero si quitamos los correos gratuitos de gmail y similares, son 20 organizaciones
	Habilidad de regeneración	Ratio de supervivencia de contribuidores	0,124497992	De 249 contribudores desde 12 abril 2016 a 12 de abril de 2017, sólo 31 eran activos un año después.
		Ratio de miembros nuevos		No hay acceso a la base de datos de la lista de correo para ver el número de miembros nuevos
		Ratio de contribuidores nuevos	0,856481481	De 216 contribuidores desde 12 abril 2017 a 12 de abril de 2018, 185 son nuevos (no habían contribuido el año anterior).
	Equilibrio de esfuerzos	Ratio de tiempo entre commits	3.2272e+17	Se calcula la varianza del tiempo entre commits por contribuidor (calculado en segundos), luego la varianza de las varianzas.
		Implicación de los proyectos de la comunidad	A=10583; B=715072; C=3203	A=Numero de commits en un año; B=Líneas cambiadas (añadidas + quitadas); C=Número de ficheros cambiados
		Características estadísticas de los commits	Mínimo número de commits = 23 Cuartil inferior de commits = 982,25 Mediana de commits = 4841 Cuartil superior de commits = 8615,25 Media de commits = 2734,77106 Máximo número de commits = 13270	Datos calculados desde 1998 hasta 12 abril de 2018
		Ratio de actividades de la comunidad	Indice Gini commits = 0,517777778; Indice Gini ficheros cambiados = 0,401286492	
		Ratio de miembros en sub-comunidades		No hay subcomunidades
		Ratio releases por desarrollador	3,9483	Calculado de los últimos 8 años

	Equilibrio de conocimiento	Conocimiento de contribuidores		La mayoría del proyecto es en C/C++ (90+% código), pero falta analizarlo con una herramienta adecuada
		Ratio de longevidad en los contribuidores	428,70 días	
		Ratio de experiencia por contribuidor	3,5369	Calculado de los últimos 8 años, con los datos finales, hacer una media para saber el número medio de releases
		Número de proyectos por contribuidor		La mayoría contribuye a varios proyectos (consderando proyecto como drivers) pero falta analizarlo con una herramienta adecuada
	Visibilidad	Número de miembros pidiendo nuevas características	15	Búsqueda en Bugzilla por enhacements del 12 Abril 2017 al 12 Abril 2018: https://bit.ly/2JICKmc
		Número de ofertas de empleo	7	Directamente relacionadas con Mesa (Día 14 de Abril de 2018): 2 (AMD), 3 (Intel), 1 (Collabora), 1 (NVIDIA)
		Número de descargas		No hay acceso a la base de datos de descargas. Se presuponen millones de instalaciones por que se distribuye con las distribuciones de linux con escritorio gráfico
		Número de subscriptores a las listas de correo	1210	Subscriptores a mesa-dev
		Número de usuarios pasivos		Millones de usuarios, todas las distribuciones de Linux con entorno gráfico tienen Mesa instalado. No hay acceso a los datos de descarga de la web
		Número de publicaciones científicas	90	Proyecto citado en 90 artículos científicos: https://scholar.google.es/scholar?cites=4138337176409462590&as_sdt=2005&sciodt=0,5&hl=es
		Publicaciones en webs y social media	135000	No hay acceso a google analytics, pero hay 135000 referencias en Google y miles de referencias en Twitter
		Número de patentes	0	
		Número de eventos	2	X.org Developer's Conference (anual), FOSDEM's graphics devroom (anual)

		Reputación y opiniones de contribuidores		No hubo tiempo para encuestas, se necesita una herramienta más compleja para obtener estadísticas
		Distribución geográfica de miembros	Todo el mundo	Principalmente Europa y América
		Aceptación de la comunidad	Aceptación alta	Se aceptan contribuciones de cualquier organización o individuo.
		Número de páginas referenciando la comunidad	135000	
	Cohesión de la comunidad	Ratio de centralidad de la comunidad		No se dispone de herramienta adecuada
		Clúster de colaboración de desarrolladores		No se dispone de herramienta adecuada
		Enlaces a otras sub-comunidades		No se dispone de herramienta adecuada
		Grado de presencia de actores clave		No se dispone de herramienta adecuada

3.2.5 Calidad de la red del ecosistema

La dimensión de Calidad de la red del ecosistema está formada por las características Salud de la red, Salud de Recursos e Interés comercial.

En la evaluación de la dimensión de Calidad de la red del ecosistema en el proyecto Mesa se ha encontrado con el problema de no disponer de una herramienta adecuada en muchas de las métricas, mientras que para algunas métricas no se ha realizado las encuestas necesarias a los partners de la comunidad para obtener los datos necesarios.

La característica Salud de la red se divide en la siguientes sub-características: Cohesión del ecosistema, Consistencia de la información, Evolución de las sinergias, Habilidad de interrelación.

No hubo ninguna métrica perteneciente a las sub-características de Cohesión de la comunidad, Evolución de las sinergias y Habilidad de interrelación, que pudiera ser obtenida por la falta de una herramienta adecuada que procesara las fuentes de entrada (listas de correo, repositorios de software, herramientas de seguimiento de errores, etc.) para obtener los valores necesarios; en algunos casos también faltaría la realización de una encuesta como es en la métrica de la Popularidad de la comunidad. Una excepción se encuentra en la sub-característica de Consistencia de la información, donde la métrica Análisis de sentimiento se realizó mediante la versión gratuita de la herramienta *Tone Analyzer* de la suite IBM Watson Developer Cloud; en dicha métrica se obtuvo los valores

que el tono de la comunidad es "analítico y con un poco de tristeza" tras analizar los correos electrónicos de marzo de 2018[25].

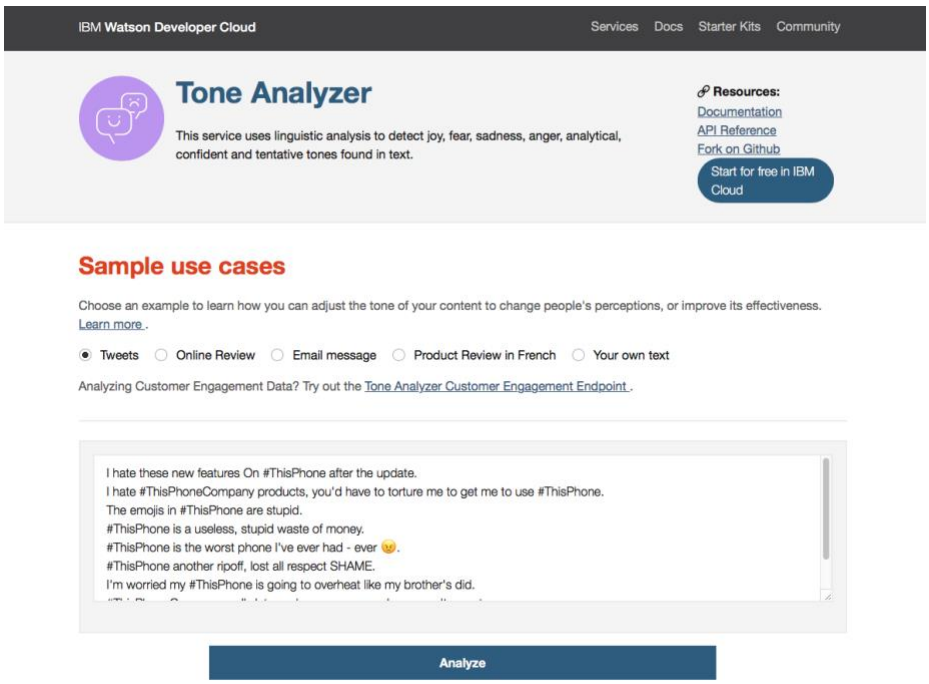


Ilustración 34: Captura de la web de la herramienta Tone Analyzer. Captura realizada el 19 de Abril de 2018.

La característica Salud de recursos se divide en las siguientes sub-características: Creación de nicho, Conocimiento de la comunidad, Vitalidad y Confiabilidad.

Al contrario de lo ocurrido en la característica de Salud de la red, en Salud de recursos hubo mucho éxito en la realización de mediciones en las métricas propuestas. Por ejemplo, en las sub-características de Creación de nicho y Conocimiento de la comunidad se obtuvieron valores para todas las métricas sin excepción.

En cambio, en la sub-características de Vitalidad y Confiabilidad no se obtuvieron tantas métricas como cabría de esperar. Como ejemplo, todas las métricas relacionadas con la salud financiera de los partners (Liquidez, Solvencia, Desarrollo de activos, Puntuación ZETA) no se pudieron obtener por falta de datos públicos sobre las finanzas de los partners que componen la comunidad. Esta información es muy sensible y normalmente no se encuentra disponible públicamente. En los casos que haya información pública (como podría ser la cuenta de resultados de un año contable), no dispone de información del impacto concreto del producto software analizado en la evaluación sobre las cuentas, ya que suele estar agrupado en conceptos más genéricos como son las ventas de una división en general o los ingresos por cuatrimestres sin desglosar.

Finalmente, la característica de Interés Comercial, propuesta como parte de este trabajo fin de máster para su inclusión en el modelo, sí obtuvo datos para tres de las cuatro

[25] No se amplió el periodo de tiempo de análisis por limitaciones en la versión gratuita de la herramienta.

métricas propuestas. La métrica que no obtuvo ningún valor, Variedad de partners en la comunidad, se podría haber completado una vez que se hubiera realizado encuestas sobre los partners presentes en la comunidad.

A continuación, se presenta la Tabla 31 donde se indican los valores obtenidos para la dimensión de Calidad de la red del ecosistema en el proyecto Mesa.

Tabla 31: Valores obtenidos para la dimensión de Calidad de la red del ecosistema en el proyecto Mesa.

Característica	Subcaracterística	Medidas	Valor	Notas
Salud de la red	Cohesión del ecosistema	Número de nodos para desconectar el ecosistema		No se dispone de herramienta adecuada
		Grado de *outdegree* de actores clave		No se dispone de herramienta adecuada
		Coeficiente de clústering de la comunidad		No se dispone de herramienta adecuada
		Número de conexiones de los partners		No se dispone de herramienta adecuada
	Consistencia de la información	Sinónimos en el mapa de vocabulario		No se dispone de herramienta
		Análisis de sentimiento	Analítica y un poco tristeza	https://tone-analyzer-demo.ng.bluemix.net Correos de marzo de 2018
	Evolución de las sinergias	Distribución de los partners		No se dispone de herramienta
		Popularidad de la comunidad		Falta realizar encuestas y herramienta adecuada
		Partnerships e integrabilidad de los proyectos de la comunidad		No se hicieron encuestas
		Reciprocidad de la comunidad		No se dispone de herramienta
	Habilidad de interrelación	Evolución de la conectividad		No se dispone de herramienta
		Evolución de la conectividad de los partners con otros		No se dispone de herramienta

		miembros de la comunidad		
		Evolución de la centralidad de la comunidad		No se dispone de herramienta
Salud de recursos	Creación de nicho	Número de tipos de contexto en la aplicación de los proyectos de la comunidad	1	Uso de drivers para tarjetas gráficas
		Número de lenguajes naturales soportados	1	Inglés
		Variedad en tecnologías de los proyectos	5 o más	Se usan lenguajes como C/C++, Python. Herramientas como automake, meson, etc.
		Número de extensiones de plataforma	0	No se soportan extensiones.
		Número de nichos de la comunidad	9	Tantos como drivers tiene.
		Número de mercados donde la comunidad está participando	1	Drivers para tarjetas gráficas.
	Conocimiento de la comunidad	Número de tipos de actividad	6	Contribución de código, documentación, bug triage, bug fixing, testing, releasing
		Número de artefactos de la comunidad	Más de 100	Página web, documentación en 01.org, planet.igalia.com, planet.collabora.com, phoronix, blogs de desarrolladores
	Vitalidad	Liquidez de los partners		No se dispone de información
		Porción de mercado de los proyectos de la comunidad		No se dispone de información
		Solvencia de los partners		No se dispone de información
		Desarrollo de activos de los partners		No se dispone de información

		Obsolescencia limitada	No	
		Continuidad de la experiencia de usuario y casos de uso	Evolución	Cada versión se basa en la anterior y no hay cambios drásticos en la experiencia de usuario.
		Aceptación de la comunidad	Aceptación alta	Se aceptan contribuciones de cualquier organización o individuo.
		Número de usuarios pasivos		Millones de usuarios, todas las distribuciones de Linux con entorno gráfico tienen Mesa instalado. No hay acceso a los datos de descarga de la web
		Número de nuevas comunidades	0	No hubo nuevas comunidades en el periodo de un año (12 abril 2017 a 12 abril 2018)
		Colaboración e integrabilidad de la comunidad		No se realizaron encuestas
	Confiabilidad	Puntuación ZETA de los partners		No se dispone de información
		Edad de la comunidad	20 años	Primer commit registrado es de 14 de febrero de 1998.
		Número de patentes de los partners		No se dispone de información
		Reputación y opiniones de contribuidores		No se realizaron encuestas, se necesita una herramienta más compleja para obtener estadísticas
Interés comercial		Variedad de partners de la comunidad		Partners privados centrados en empresas TIC tanto de fabricación de HW como desarrolladoras de SW. No se realizaron encuestas
		Productos software	Sí	Chrome OS, Android, set-top boxes, ordenadores, sistemas embebidos…
		Ámbitos de uso	Internacional	
		Servicios alrededor de la comunidad	4	Consultoría, soporte, mantenimiento, formación. Existen consultoras (Igalia, Collabora) que también ofrecen formación y mantenimiento, empresas de soporte (Red Hat, SUSE).

3.2.6 Discusión

Una vez analizado el caso de uso, se obtienen varias conclusiones que merecen la pena discutir en este trabajo final de máster.

La primera observación es la facilidad de obtener datos sobre las características que se proponen añadir al modelo de evaluación QuESO: Apertura e Interés comercial. En ambos casos, se obtiene información de valor que no era contemplada previamente en este modelo y que ofrece un punto de vista distinto a las características ya contempladas en el modelo QuESO original.

La segunda observación es la falta de información para la realización de algunas métricas ya sea por la necesidad de realizar encuestas (como las encuestas a los partners de la comunidad), bien porque no haya acceso externo a la información por parte de los evaluadores (por ejemplo, bases de datos para obtener información del número de descargas) o, en el peor caso, que dicha información no es pública (por ejemplo, los datos financieros de los partners de la comunidad). En estos dos últimos casos, la falta de información provoca que algunas métricas no sean posibles de obtener en un caso de estudio real, como es el propuesto en este trabajo fin de máster sobre el proyecto Mesa, salvo que se trabaje para uno de los partners que contribuyan a la comunidad y se disponga de información interna. Sería interesante explorar en futuras líneas de investigación la posibilidad de utilizar métricas alternativas que proporcionen datos de significado similar a las sub-características afectadas y que éstas se puedan obtener de fuentes de información públicas.

En el proceso de evaluación de otras métricas se ha encontrado que no se pudieron obtener debido a la falta de una herramienta software bajo licencia libre apropiada que procese los datos necesarios para obtener un valor. Durante la realización de este trabajo final de máster, no se han encontrado estas herramientas libres tras varias búsquedas en buscadores web populares. Ni si quiera hay un repositorio público de herramientas software bajo licencia libre, ni de evaluaciones ya realizadas para el modelo QuESO en Internet. Este hecho sugiere que una futura línea de investigación sería el desarrollo de estas herramientas de una manera pública y colaborativa, siguiendo los principios del software libre, así como la creación de un repositorio público de evaluaciones.

En la Tabla 32 se indican las estadísticas de las métricas encontradas por característica y el porcentaje frente al total. Cabe resaltar que la peor sub-característica es Salud de la red, donde la falta de una herramienta adecuada para cada métrica es la razón de este mal resultado.

Tabla 32: Estadística de las métricas obtenidas por cada característica.

Dimension	Característica	Subcaracterística	Número de métricas	Métricas con medidas	Porcentaje de métricas medidas por subcaracterística	Porcentaje de métricas medidas por característica
Calidad de plataforma	-	-	-	-	-	-
Calidad de comunidad	Apertura	Licencias	3	3	100	100
		Ética	3	3	100	
		Participación	3	3	100	
	Capacidad de mantenimiento	Tamaño	5	3	60	78,94736842
		Actividad	14	12	85,71428571	
	Madurez de procesos	-	-	-	-	-
	Sostenibilidad	Heterogeneidad	5	3	60	62,85714286
		Habilidad de regeneración	3	2	66,66666667	
		Equilibrio de esfuerzos	6	5	83,33333333	
		Equilibrio de conocimiento	4	2	50	
		Visibilidad	13	10	76,92307692	
		Cohesión de la comunidad	4	0	0	
Calidad de la red del ecosistema	Salud de la red	Cohesión del ecosistema	4	0	0	7,692307692
		Consistencia de la información	2	1	50	
		Evolución de las sinergias	4	0	0	

		Habilidad de interrelación	3	0	0	
	Salud de recursos	Creación de nicho	6	6	100	59,09090909
		Conocimiento de la comunidad	2	2	100	
		Vitalidad	10	4	40	
		Confiabilidad	4	1	25	
	Interés comercial	-	4	3	75	75

4. Conclusiones y líneas futuras

4.1 Conclusiones

Este trabajo final de máster ha analizado los modelos de evaluación de calidad de software genéricos y se ha comentado el por qué no son suficientes para evaluar la calidad de los proyectos de software libre, ya que no tienen en cuentas aspectos propios del software libre como son la comunidad, las licencias y la participación colaborativa.

Posteriormente se describieron los principales modelos de evaluación de calidad creados específicamente para analizar proyectos de software libre, y se realizó una comparativa entre ellos siguiendo el método FOCOSEM [7]. En dicha evaluación, se encontró que la gran mayoría de estos métodos tuvieron escaso éxito fuera del ámbito académico y la mayoría de ellos fueron abandonados tras unos pocos años de su definición. El único modelo que continúa en activo y que continúa desarrollándose es QuESO [23], [24], no obstante QuESO no es perfecto y tiene varias lagunas que se intentaron cubrir con las aportaciones de este trabajo final de máster.

Las aportaciones realizadas al modelo QuESO fueron principalmente tres. Las dos primeras son la definición de las características Apertura e Interés comercial para las dimensiones de Calidad de comunidad y Calidad de la red del ecosistema, respectivamente. La tercera aportación fue el empleo de perfiles para personalizar la evaluación mediante la aplicación de pesos sobre las métricas para ajustar los resultados al interés que se busca con la evaluación, permitiendo al evaluador a que pueda, por ejemplo, ignorar métricas que no sean importantes para su caso de uso o que no se dispongan de suficientes datos como para poder evaluarlas de manera confiable.

En el caso de la característica Apertura, se sugirieron tres sub-características (Licencias, Ética y Participación) que cubren los aspectos como las licencias que tiene el software, si se requieren módulos propietarios, si existe un código de conducta, los valores de la comunidad y cómo es la toma de decisiones de la comunidad, entre otros aspectos. Para estas tres sub-características se definieron métricas específicas para cada una de ellas que pudieron ser validadas en el ejemplo de aplicación del modelo realizado en este trabajo fin de máster.

Para el caso de la característica de Interés comercial, se han definido métricas relacionadas con los servicios alrededor de la comunidad, los productos que usan el software analizado y los ámbitos de uso, entre otros; con el objetivo de ayudar la elección del software adecuado principalmente a organizaciones que busquen utilizarlo en un ámbito comercial.

Finalmente, se ha aplicado el modelo QuESO junto con las aportaciones realizadas sobre un proyecto de software libre de referencia como es Mesa, la librería de gráficos 3D usada ampliamente en el mundo del software libre. A lo largo del proceso de evaluación

realizado se ha encontrado distintos problemas con algunas de las métricas del modelo propuesto. En muchos casos, hay métricas donde no disponen de una fuente de información pública como es la información sobre datos financieros de los partners de la comunidad. En cambio, para algunas métricas el problema se encontró en la falta de realización de una encuesta para obtener la información necesaria. No obstante, el caso más preocupante fue la falta de herramientas software adecuadas para evaluar algunas de las métricas del modelo.

Algunas de estas herramientas software de evaluación son de pago, como por ejemplo *Tone Analyzer* de IBM, empleado para analizar el sentimiento de los correos electrónicos intercambiados en la comunidad. Esta herramienta tiene una versión gratuita que está limitada por el número de caracteres a procesar, con lo que no se pudo realizar un análisis exhaustivo que hubiera dado datos más precisos. En otros casos, las herramientas libres necesarias no se encontraron tras realizar varias búsquedas en buscadores web.

Además, hay problemas añadidos en la búsqueda de herramientas adecuadas ya que tienen que ser compatibles con las tecnologías empleadas en la comunidad del software a analizar e idealmente bajo licencias libres. Por ejemplo, si la comunidad analizada emplea *git* para el repositorio de software, *bugzilla* para la herramienta de seguimiento de errores, *mailman* para las listas de correo, etc, esto podría ser un problema para que algunas herramientas obtengan datos para las métricas que fueron diseñadas.

Durante la realización del ejemplo de aplicación del modelo propuesto, las métricas relacionadas con los repositorios de software se pudieron realizar fácilmente mediante *git* y un conocimiento medio de *bash scripting*, junto con *Octave* (alternativa libre de *Matlab*) para realizar los cálculos matemáticos necesarios.

Con todo ello, se ha visto que el modelo propuesto basado en QuESO y que incluye las aportaciones explicadas previamente es válido y usable en el mundo real. En el apartado de Líneas de investigación futuras se indicarán las posibles áreas donde se puede mejorar este modelo y breve descripción de los pasos a realizar.

4.2 Líneas de investigación futuras

Este trabajo final de máster ha hecho aportaciones interesantes que sientan las bases para realizar futuras líneas de investigación. A continuación, se indican varias áreas que podrían ser interesantes para explorar sus posibilidades como punto de partido.

La primera propuesta es la realización de un estudio exhaustivo a usuarios y empresas que utilicen software libre, para comprobar qué aspectos valoran más a la hora de utilizar y/o contribuir a un proyecto de software libre. El objetivo final es la creación de un modelo de evaluación de calidad de un proyecto de software libre en base a la experiencia obtenida con modelos anteriores junto con los datos recogidos en dicho estudio. Con esta información se buscarán las características y métricas asociadas para cubrir los aspectos resaltados como importantes en dicho estudio.

A partir de la definición del modelo, se propone el análisis de las métricas existentes buscando optimizar aquéllas donde haya margen de mejora. También se propone la

investigación y desarrollo de nuevas métricas que puedan ser utilizadas mediante el uso de información pública, cuyos valores aporten a la evaluación y que puedan ser recogidas mediante una herramienta software de manera automatizada. Opcionalmente se pueden buscar nuevas métricas que reemplacen a las ya existentes y que no cumplan estos requisitos.

Es el caso de las herramientas software, se propone la realización de herramientas para cada una de las métricas que soporten varias tecnologías que emplean los proyectos de software libre (software de control de versiones, listas de correo, herramientas de seguimiento de errores) y que puedan ser ampliables en el futuro mediante módulos en el caso de la aparición de nuevas tecnologías que surjan en el futuro. Idealmente, estas herramientas software de medición serán publicadas bajo software libre y disponibles de manera gratuita en repositorios de software públicos para que otros investigadores puedan colaborar y así ayudar a hacer una comunidad alrededor del modelo propuesto.

Finalmente, sería interesante la creación de un repositorio central y público con los datos recogidos en las evaluaciones realizadas por los modelos de evaluación de calidad de proyectos de software libre. Este repositorio sería la base para realizar estudios avanzados sobre tendencias en las evaluaciones, qué patrones se repiten, qué áreas son susceptibles de mejora mediante el estudio con herramientas de análisis de datos estadísticos y otros tipos de análisis para evaluar la salud general de la comunidad del software libre y comprobar la adecuación de los modelos existentes para realizar una evaluación fidedigna con la realidad.

Bibliografía

[1] R. Glott, A. K. Groven, K. Haaland, y A. Tannenberg, «Quality Models for Free/Libre Open Source Software Towards the #147;Silver Bullet #148;?», en *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications*, 2010, pp. 439-446.

[2] J.-C. Deprez y S. Alexandre, «Comparing Assessment Methodologies for Free/Open Source Software: OpenBRR and QSOS», en *Product-Focused Software Process Improvement*, 2008, pp. 189-203.

[3] K. Haaland, A. Groven, N. Regnesentral, R. Glott, y A. Tannenberg, *Free/Libre Open Source Quality Models- a comparison between two approaches*. .

[4] «Comparison of open source maturity models - ScienceDirect». [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S1877050917312061. [Accedido: 18-ene-2018].

[5] A. Adewumi, S. Misra, N. Omoregbe, B. Crawford, y R. Soto, «A systematic literature review of open source software quality assessment models», *SpringerPlus*, vol. 5, n.º 1, p. 1936, nov. 2016.

[6] S. Andrade y F. Saraiva, «Principled Evaluation of Strengths and Weaknesses in FLOSS Communities: A Systematic Mixed Methods Maturity Model Approach», en *Open Source Systems: Towards Robust Practices*, 2017, pp. 34-46.

[7] K.-J. Stol y M. A. Babar, «A Comparison Framework for Open Source Software Evaluation Methods», en *Open Source Software: New Horizons*, 2010, pp. 389-394.

[8] J. P. Miguel, D. Mauricio, y G. Rodriguez, «A Review of Software Quality Models for the Evaluation of Software Products», *Int. J. Softw. Eng. Appl.*, vol. 5, n.º 6, pp. 31-53, nov. 2014.

[9] K. Haaland, R. Glott, y A. Tannenberg, «Free/Libre Open Source Quality Models- a comparison between two approaches», nov. 2017.

[10] S. K. Khatri y I. Singh, «Evaluation of open source software and improving its quality», en *2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, 2016, pp. 114-119.

[11] B. W. Boehm, J. R. Brown, y M. Lipow, «Quantitative Evaluation of Software Quality», en *Proceedings of the 2Nd International Conference on Software Engineering*, Los Alamitos, CA, USA, 1976, pp. 592–605.

[12] J. A. McCall, P. K. Richards, y G. F. Walters, «Factors in Software Quality. Volume I. Concepts and Definitions of Software Quality», GENERAL ELECTRIC CO SUNNYVALE CA, GENERAL ELECTRIC CO SUNNYVALE CA, nov. 1977.

[13] «CMMI Institute». [En línea]. Disponible en: http://cmmiinstitute.com/. [Accedido: 27-nov-2017].

[14] M. B. Chrissis, M. Konrad, y S. Shrum, *CMMI for Development: Guidelines for Process Integration and Product Improvement*. Pearson Education, 2011.

[15] J. M. Navarro y J. Garzás, «Experiencia en la implantación de CMMI-DEV v1.2 en una micropyme con metodologías ágiles y software libre», *REICIS Rev. Esp. Innov. Calid. E Ing. Softw.*, vol. 6, n.º 1, 2010.

[16] R. G. Dromey, «A model for software product quality», *IEEE Trans. Softw. Eng.*, vol. 21, n.º 2, pp. 146-162, feb. 1995.

[17] I. Samoladas, G. Gousios, D. Spinellis, y I. Stamelos, «The SQO-OSS Quality Model: Measurement Based Open Source Software Evaluation», en *Open Source*

Development, Communities and Quality, 2008, pp. 237-248.
[18] M. Soto y M. Ciolkowski, «The QualOSS open source assessment model measuring the performance of open source communities», en *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, 2009, pp. 498-501.
[19] Wasserman, A and Pal, M and Chan, C, «Business Readiness Rating Project "BRR Whitepaper 2005 RFC 1"». .
[20] «Collaborative technological watch». [En línea]. Disponible en: http://www.qsos.org/. [Accedido: 12-dic-2017].
[21] D. Izquierdo-Cortazar, J. M. Gonzalez-Barahona, S. Duenas, y G. Robles, «Towards Automated Quality Models for Software Development Communities: The QualOSS and FLOSSMetrics Case», en *2010 Seventh International Conference on the Quality of Information and Communications Technology*, 2010, pp. 364-369.
[22] V. del Bianco, L. Lavazza, S. Morasca, D. Taibi, y D. Tosi, «The QualiSPo Approach to OSS Product Quality Evaluation», en *Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, New York, NY, USA, 2010, pp. 23–28.
[23] O. Franco-Bedoya, D. Ameller, D. Costal, y X. Franch, «QuESo a quality model for open source software ecosystems», en *2014 9th International Conference on Software Engineering and Applications (ICSOFT-EA)*, 2014, pp. 209-221.
[24] F. Bedoya, Ó. Hernán, D. Ameller, D. Costal Costa, y J. Franch Gutiérrez, «QuESo V2.0 a quality model for open source software ecosystems: List of measures», may 2016.
[25] D. Spinellis *et al.*, «Evaluating the Quality of Open Source Software», *Electron. Notes Theor. Comput. Sci.*, vol. 233, n.º Supplement C, pp. 5-28, mar. 2009.
[26] «Continued Voluntary Participation Intention in Firm-Participating Open Source Software Projects | Information Systems Research». [En línea]. Disponible en: https://pubsonline.informs.org/doi/abs/10.1287/isre.2016.0687. [Accedido: 20-feb-2018].
[27] J. West y S. O'Mahony, «Contrasting Community Building in Sponsored and Community Founded Open Source Projects», en *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, 2005, pp. 196c-196c.
[28] Aknouche Lamine, Shoan Goran, «Motivations for Open Source Project Entrance and Continued Participation», Master Thesis, Lund University, 2013.
[29] J. A. Roberts, I.-H. Hann, y S. A. Slaughter, «Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects», *Manag. Sci.*, vol. 52, n.º 7, pp. 984-999, jul. 2006.
[30] Y. Fang y D. Neufeld, «Understanding Sustained Participation in Open Source Software Projects», *J. Manag. Inf. Syst.*, vol. 25, n.º 4, pp. 9-50, abr. 2009.
[31] K. Crowston, K. Wei, J. Howison, y A. Wiggins, «Free/Libre Open-source Software Development: What We Know and What We Do Not Know», *ACM Comput Surv*, vol. 44, n.º 2, pp. 7:1–7:35, mar. 2008.
[32] G. von Krogh, S. Haefliger, S. Spaeth, y M. W. Wallin, «Carrots and Rainbows: Motivation and Social Practice in Open Source Software Development», *MIS Q.*, vol. 36, n.º 2, pp. 649-676, 2012.
[33] «gnu.org». [En línea]. Disponible en: https://www.gnu.org/licenses/licenses.html. [Accedido: 03-mar-2018].
[34] «FSF Licensing & Compliance Team — Free Software Foundation — working

together for free software». [En línea]. Disponible en: https://www.fsf.org/licensing/. [Accedido: 03-mar-2018].
[35] «Contributor licence agreement | Ubuntu». [En línea]. Disponible en: https://www.ubuntu.com/legal/contributors. [Accedido: 03-mar-2018].
[36] «mjg59 | Mir, the Canonical CLA and skewing the playing field». [En línea]. Disponible en: https://mjg59.dreamwidth.org/25376.html. [Accedido: 03-mar-2018].
[37] «Intel Reverts Plans, Will Not Support Ubuntu's XMir - Phoronix». [En línea]. Disponible en: https://www.phoronix.com/scan.php?page=news_item&px=MTQ1NjY. [Accedido: 03-mar-2018].
[38] J. Terrell *et al.*, «Gender differences and bias in open source: pull request acceptance of women versus men», *PeerJ Comput. Sci.*, vol. 3, p. e111, may 2017.
[39] «Women in Software Engineering: The Sobering Stats». [En línea]. Disponible en: http://talent.linkedin.com/blog/index.php/2014/03/women-in-engineering-the-sobering-stats. [Accedido: 04-mar-2018].
[40] A. McPherson, «So Are You A Contributor?: Women's Contributions to Linux and Open Source Span Technology and Business», *Open Source Bus. Resour.*, n.º June 2009, 2009.
[41] Gregorio Robles, Bernhard Krieger, Ruediger Glott, Rishab A. Ghosh, «FLOSS Final Report - Part 4 - Survey of Developers». .
[42] «FLOSSpols: home». [En línea]. Disponible en: http://flosspols.merit.unu.edu/. [Accedido: 03-mar-2018].
[43] James Leach, Bernhard Krieger, «FLOSSPOLS D16 Gender Integrated Report of Findings». 01-mar-2006.
[44] M. S. Elliott y W. Scacchi, «Free Software Developers As an Occupational Community: Resolving Conflicts and Fostering Collaboration», en *Proceedings of the 2003 International ACM SIGGROUP Conference on Supporting Group Work*, New York, NY, USA, 2003, pp. 21–30.
[45] E. Raymond, «The cathedral and the bazaar», *Knowl. Technol. Policy*, vol. 12, n.º 3, pp. 23-49, sep. 1999.
[46] S. O'Mahony y F. Ferraro, «The Emergence of Governance in an Open Source Community», *Acad. Manage. J.*, vol. 50, n.º 5, pp. 1079-1106, ene. 2007.
[47] B. Vasilescu, A. Serebrenik, M. Goeminne, y T. Mens, «On the variation and specialisation of workload—A case study of the <Emphasis Type="SmallCaps">Gnome</Emphasis> ecosystem community», *Empir. Softw. Eng.*, vol. 19, n.º 4, pp. 955-1008, ago. 2014.
[48] P. Bonacich, «Power and Centrality: A Family of Measures», *Am. J. Sociol.*, vol. 92, n.º 5, pp. 1170-1182, 1987.
[49] L. Dahlander y M. G. Magnusson, «Relationships between open source software companies and communities: Observations from Nordic firms», *Res. Policy*, vol. 34, n.º 4, pp. 481-493, may 2005.
[50] K. J. Stewart, A. P. Ammeter, y L. M. Maruping, «Impacts of License Choice and Organizational Sponsorship on User Interest and Development Activity in Open Source Software Projects», *Inf. Syst. Res.*, vol. 17, n.º 2, pp. 126-144, jun. 2006.
[51] R. Stallman, *Free Software, Free Society: Selected Essays of Richard M. Stallman*. Lulu.com, 2002.
[52] J. M. Gonzalez-Barahona, G. Robles, y D. Izquierdo-Cortazar, «Determining the Geographical Distribution of a Community by Means of a Time-zone Analysis», en

Proceedings of the 12th International Symposium on Open Collaboration, New York, NY, USA, 2016, pp. 3:1–3:4.

Glosario de términos utilizados

API: *Application Programming Interface*, interfaz de programación de aplicaciones.
CLA: acuerdos de licencia de contribución o *Contributor License Agreement* en inglés.
CMMI: Integración de modelos de madurez de capacidades o *Capability Maturity Model Integration* en sus siglas en inglés.
FOSS/FLOSS: *Free/Libre Open Source Software.*
OpenBRR: *Open Business Readiness Rating.*
OSMM: *Open Source Maturity Model*, nombre de un modelo de evaluación de software open-source.
PYME: Pequeña y Mediana Empresa.
QSOS: *Qualification and Selection of Open Source software*, nombre de un modelo de evaluación de software open-source.
SQuaRE: *Software Product Quality Requeriments and Evaluation*.
SQO-OSS: *Software Quality Observatory for Open Source Software*, nombre de un modelo de evaluación de software open-source.
TFM: Trabajo Final de Máster.