F/OSS para el reuso: Métricas, Desarrollo de Herramientas y Marco para su Evaluación

Alumno: Jorge Ramirez

Director: M.Sc. Gustavo Gil

Co-Director: Dr. Gustavo Rossi

Tesis presentada para obtener el grado de Magister en Ingeniería de Software

Facultad de Informática - Universidad Nacional de La Plata

Julio de 2015

Resumen

La disponibilidad de una creciente variedad de Software Libre y de Código Abierto (F/OSS por sus siglas en inglés) distribuidos bajo licencias que permiten explícitamente su modificación y el desarrollo de aplicaciones derivadas, abre nuevas posibilidades para el desarrollo con reutilización.

Las características y la información disponible en relación a cada producto de F/OSS es muy heterogénea; no obstante, todas tienen en común la libre disponibilidad del código fuente.

Esta tesis presenta nuevas formas de abordar el problema de evaluar y seleccionar productos de F/OSS, centrándose en dos aspectos:

- La utilización de métricas que pueden obtenerse del código fuente para la evaluación de F/OSS, de modo de caracterizar el diseño general de los productos de software que se consideren para la reutilización.
- La elaboración de un marco de trabajo para la selección y evaluación de software que sirva de base para una sistematización paulatina de la incorporación de la reutilización de F/OSS en diferentes procesos de desarrollo de software.

A partir del estudio de una muestra de 560 versiones diferentes de aplicaciones de F/OSS, y en base al estudio de distribuciones de frecuencia, se proponen umbrales para métricas que reflejan aspectos generales del diseño de una aplicación. Específicamente, se consideran el promedio de la cantidad de métodos por clase y la proporción de referencias a métodos de otras clases respecto del total de métodos definidos en la aplicación. Estas dos medidas se refieren a la aplicación en su conjunto y no a clases o módulos particulares, lo que supone una utilidad diferente de estos valores respecto del papel que cumplen en otros aspectos de la gestión de proyectos de software.

En relación al segundo aspecto, se presenta un marco de trabajo en tres etapas, a saber, la detección de oportunidades de reutilización, la búsqueda de recursos candidatos y la selección y evaluación. Este marco es de carácter general, pudiendo adaptarse a diferentes metodologías y contextos de desarrollo.

La elaboración del marco parte de la realización de una experiencia de desarrollo con reutilización de F/OSS, la revisión de la literatura relacionada con las temáticas de selección de elementos reutilizables y la evaluación de éstos para su incorporación en nuevos desarrollos.

Este marco se aplica posteriormente al desarrollo de una herramienta para computar un conjunto de métricas basadas en el código fuente para dar soporte al proceso de selección de F/OSS para la reutilización tomando en cuenta los umbrales propuestos previamente.

De esta forma, se ofrece aquí una base para abordar la reutilización de F/OSS de manera de permitir la sistematización paulatina de esta práctica, brindando un criterio de valoración provisional basado en métricas obtenidas del código fuente.

Índice de contenido

Capítulo 1.- Introducción

Se ha escrito mucho acerca de los beneficios potenciales de la reutilización en el desarrollo de software.

Parece evidente que conviene más disponer de un software o de una porción de software ya desarrollada que tener que producirla desde cero. Como dice Brooks en su célebre ensayo, “la solución más radical al desarrollo de software es no desarrollarlo”[1].

Entre los beneficios que se enumeran se mencionan habitualmente[2] [3] la disminución del costo de desarrollo, la utilización de porciones de software ampliamente testeadas, la reducción del tiempo de entrega, etc.

Estas afirmaciones datan desde los primeros tiempos de la reutilización[4] [2]; sin embargo, la adopción de la reutilización de manera sistemática en el desarrollo de software sigue siendo marginal.

Se han publicado numerosos trabajos acerca de los factores de éxito y de fracaso en la adopción de la reutilización sistemática de software[5] [6], así como sobre las dificultades[7] que surgen a la hora de que una organización o un equipo de desarrollo se proponga establecer procesos sistemáticos de reutilización.

Uno de los aspectos que se consigna con frecuencia como una limitante para la reutilización es la dificultad de encontrar software reutilizable que se ajuste a las necesidades de desarrollo específicas. La integración de los productos adquiridos para ese fin puede terminar siendo onerosa, debido a dificultades en la comprensión del elemento obtenido, o la adecuación sólo parcial a los requerimientos que debería cubrir.

Otra complicación para la búsqueda y recuperación de software reutilizable es la falta de uniformidad en cuanto a la descripción de las características de un elemento, lo que dificulta la coincidencia entre las especificaciones redactadas por los desarrolladores que pretenden reutilizar y las características expuestas por los autores de los componentes.

La aparición de una gran cantidad de productos de software públicamente accesible y que incluye el código fuente -al que genéricamente denominaremos aquí Software Libre y de Código Abierto o F/OSS por sus siglas en inglés-, constituye una fuente novedosa de recursos potencialmente reutilizables; no obstante, el aprovechamiento de este tipo de software trae consigo nuevas dificultades específicas.

El desarrollo de la investigación que sustenta el presente texto produjo diversas presentaciones en eventos académicos, según se detalla en la tabla 1.

Autores y Títulos	Temática	Año
Ramirez, Gil, Romero, Gimson, “Indicadores de La Utilización Del Bug Tracker En Proyectos F/OSS.”[8]	Calidad de la información de proyectos F/OSS respecto de la gestión de errores	2009
Ramirez, Gimson, Gil, “Evaluación de La Evolución Del Diseño En F/OSS:un Caso de Estudio.”[9]	Estudio de la evolución de SweetHome 3D en función de los valores de diferentes métricas	2010
Ramirez, Gil, Gimson. “El Nacimiento de Una Herramienta Educativa Libre.”[10]	Experiencia de desarrollo con reutilización de F/OSS	2011
Ramirez, “Mantenibilidad y Evolutividad en el Software Libre y de Código Abierto.”[11]	Investigación Bibliográfica sobre Mantenibilidad y Evolutividad en F/OSS	2011
Ramirez, Gil, Massé Palermo, “Hacia un catálogo práctico de componentes de F/OSS para la reutilización.”[12]	Búsqueda y recuperación de F/OSS para la reutilización	2012
Ramirez, Reyes, Gil, “Métricas de Código fuente y Evolución de F/OSS: un estudio exploratorio.”[13]	Estudio exploratorio sobre la evolución de 15 F/OSS, contrastando con los resultados de [9]	2013
Ramirez, Reyes, Gil, Durgam., “Evolución Y Reusabilidad En F/OSS.”[14]	Resumen de las investigaciones sobre reusabilidad y descripción de las actividades actuales sobre evolución de F/OSS	2015
Ramirez, Gil, Reyes, “Umbrales Sugeridos Para Promedios de Métricas de Diseño de Una Aplicación En Java”(a la espera de evaluación).[15]	Propuesta de umbrales para promedios de métricas de diseño, a partir del estudio estadístico de las mismas sobre F/OSS con larga evolución	2015

Tabla 1: Presentaciones ante eventos académicos

El presente trabajo estudia aspectos característicos del F/OSS a fin de facilitar su reutilización en diferentes contextos. Se plantea, además, un marco de trabajo que permita aproximarse a una reutilización sistemática o pragmática; estos términos se analizarán con más detalle posteriormente.

Para ello, repasaremos en este capítulo los tipos de reutilización, nociones sobre componentes de software, las características y definiciones del F/OSS, un breve panorama sobre las dificultades de reutilización del F/OSS y la problemática de la evaluación de componentes. Estas temáticas se retomarán con mayor profundidad en el

resto del texto.

Posteriormente, presentaremos la estructura de esta tesis.

1.1 Conceptos Previos

1.1.1 Tipos de Reutilización

En un equipo de desarrollo o en una organización, la reutilización puede adoptarse de maneras diferentes[3] [16], desde el aprovechamiento ocasional de código ya escrito hasta la adopción de procesos que contemplan la previsión de las partes a reutilizar desde las primeras etapas del desarrollo, pasando por diversas situaciones intermedias.

Cada desarrollo, de manera individual, podría incluir partes reutilizadas a partir de la experiencia de los desarrolladores en trabajos similares; puede partir del desarrollo de piezas reutilizables diseñadas específicamente para ese fin, adquirirse partes de terceros, o aprovechar la disponibilidad de elementos dentro o fuera de la organización.

Se pueden reutilizar código, binarios, documentación diversa (requisitos, diseños), algoritmos, e incluso procesos de desarrollo. Debido a esta heterogeneidad, los elementos en cuestión suelen denominarse “activos” o “recursos” reutilizables (en inglés se emplea el término assets)[3] [2] [17].

Prieto-Díaz [18] presenta una taxonomía de la reutilización desde diferentes perspectivas o facetas, de acuerdo con la siguiente tabla:

Taxonomía de Reutilización					
Substancia	**Alcance**	**Modo**	**Técnica**	**Propósito**	**Producto**
Ideas, conceptos Artefactos, componentes Procedimientos, habilidades	Vertical Horizontal	Planificado, sistemático Ad-hoc, oportunista	Composicional Generativo	Caja Negra, tal cual es Caja Blanca, modificado	Código fuente Diseño Especificación Objetos Textos Arquitecturas

Tabla 2: Taxonomía de la Reutilización propuesta por Prieto-Díaz

La reutilización horizontal se refiere a recursos aplicables a un amplio abanico de aplicaciones, como elementos de interfaces gráficas o bibliotecas de uso común; por el contrario, la reutilización vertical comprende unidades que pueden emplearse dentro de aplicaciones de un mismo tipo o correspondiente a un mismo dominio.

Otra distinción es entre reutilización composicional, que consiste en integrar elementos ya

existentes, de la generativa, asociada a frameworks y generadores de código.

1.1.1.1 Otras categorías

La reutilización en el ámbito del F/OSS frecuentemente no se ajusta a las categorías propuestas por Prieto-Diaz, por lo cual es importante revisar otros esquemas que pueden aprovechar la disponibilidad del código fuente, sin que necesariamente signifique una forma de reutilización ocasional, asistemática o carente de método.

La clasificación según el Propósito se refiere a la utilización de los recursos tal cual se los encuentra (caja negra) frente a la alternativa de modificarlos. Por otro lado, hay autores que consideran también la reutilización “de caja de vidrio” [3] -en la que se conoce el código pero no se modifica- y de “caja gris”[19] -donde el componente ofrece formas para modificar su comportamiento, por ejemplo a través de plugins.

En la categoría “Modo”, la tipología propuesta por Prieto-Diaz no contempla lo que algunos autores denominan “reutilización pragmática”[20] [21] [22]. Ese tipo de reutilización se emparenta con el enfoque oportunista, pero puede realizarse de manera sistemática, administrada y evaluada[7]. La utilización del término “pragmático” intenta despojar de connotaciones negativas a la reutilización oportunista y plantea la importancia de estudiar y sistematizar estas prácticas para superar los procesos ad-hoc.

Si no se consideran las perspectivas pragmáticas, es difícil clasificar los enfoques que se basan en técnicas para obtener componentes o partes de código reutilizables a partir de desarrollos que no necesariamente tuvieron entre sus propósitos tal reutilización[23] [22] [24]. Uno de los principales objetivos de estos enfoques es el de aprovechar el esfuerzo realizado en desarrollos previos de manera sistemática; en ese sentido, el problema a abordar tiene similitudes con la búsqueda de recursos a los que se puede acceder libremente, aunque no fueron desarrollados dentro de la propia organización.

1.1.2 Componentes de Software

Ya en 1968 McIlroy[4] postuló la producción masiva de elementos de software que pudieran integrarse en los desarrollos. Este matemático e ingeniero consideraba que una de las causas de la debilidad de la industria de software es la falta de una subindustria de componentes. Para subsanar esta debilidad, planteó la necesidad de disponer de catálogos de rutinas, clasificadas por precisión, robustez, performance, tamaños límite y tiempo de enlace de los parámetros.

McIlroy imaginaba la disponibilidad de piezas de software que realizaran tareas o cumplieran funciones similares, de modo que fueran intercambiables entre sí. El desarrollador podría elegir entre esas partes, lo que impulsaría el avance de esta

subindustria en virtud de la competencia.

McIlroy utiliza la expresión “componente de software”, pero no define con precisión de qué se tratarían; en los años posteriores se propusieron diferentes definiciones[17] [2] [25], que reflejan distintos posicionamientos respecto de qué debe incluirse bajo esa expresión y de qué manera se incorporan al desarrollo de software. No obstante, todas las definiciones coinciden en que se trata de piezas de software que pueden integrarse a otros desarrollos y que pueden desplegarse de manera independiente.

A los fines de nuestro trabajo tendremos en cuenta los dos aspectos mencionados, es decir, la posibilidad de integración y la distribución independiente; el software a analizar no se limitará a los que se ajusten a alguna de las definiciones de componentes de software.

1.1.3 Evaluación y Certificación de componentes de Software

Si se superan las dificultades que derivan de la falta de estandarización y consenso en la descripción de componentes y, por lo tanto, se simplificara el proceso de búsqueda de elementos reutilizables, la siguiente tarea sería la de comparar entre los productos encontrados para decidir cuál se adquirirá.

Como mencionamos anteriormente, la visión de McIlroy[4] contemplaba la posibilidad de evaluar y comparar componentes entre sí, para poder elegir entre ellos. Sin embargo, hay enfoques muy diferentes acerca de cómo realizar la evaluación.

La selección de un componente suele comprender la funcionalidad [26] [27] (es decir, el componente debe realizar las tareas que requiere el diseño del software) y la capacidad de integrarse. En general, el proceso de selección trata de obtener un ranking que disminuya el número de componentes candidatos, sobre los cuales el análisis puede incluir nuevas pruebas o estudios individuales.

Se han propuesto diversos marcos de trabajo para organizar el proceso de selección de componentes de software[27] [26]. En el caso del F/OSS no se han elaborado marcos similares que contemplen las especificidades de estos productos y las diferentes formas en que puede abordarse su reutilización, lo que será uno de los temas a abordar en el presente trabajo.

En lo que respecta a la certificación de software, se han propuesto perspectivas diferentes: por un lado, la conformación de organizaciones o laboratorios independientes que se encarguen de esa tarea a partir de la observación de los procesos de seguidos en el desarrollo de componentes o del análisis minucioso de casos de prueba [28] [29]; Morris y otros, en cambio, observaron que este modelo no se adecua a pequeñas organizaciones que desarrollan componentes ni a la producción de F/OSS, por lo que

plantea un sistema de autocertificación[30].

1.1.4 Reusabilidad

En el ámbito de la reutilización de software, los recursos a ser reutilizados pueden ser más fáciles o más difíciles de integrar en nuevos desarrollos. Esta característica se denomina reusabilidad, y comprende diferentes aspectos[18]. El F/OSS se crea con la explícita intención de que puedan realizarse desarrollos derivados y por ende reutilizarse; sin embargo, como se verá más adelante, tal reutilización no es sencilla.

Existen diferentes enfoques acerca de cómo evaluar este atributo, aunque no hay un consenso generalizado[18] [31].

1.1.5 El software libre y de código abierto

Desde comienzos de los '80 surge un movimiento que enfatiza aspectos éticos, legales y financieros de un modo de desarrollo de software cuyos principios y prácticas difieren esencialmente del modelo comercial que dominaba (y sigue dominando) el desarrollo de software[32].

A la fecha, hay miles de productos de software distribuidos con licencias libres, las que se caracterizan por permitir la modificación del software para adaptarlo o crear nuevas aplicaciones. Algunos de ellos alcanzaron gran difusión, convirtiéndose incluso en estándares *de facto* en algunos dominios, como es el caso de BIND.

Tal proliferación de recursos libremente disponibles, frecuentemente gratuitos (la gratuidad no es una exigencia para que un producto sea F/OSS) y distribuidos bajo licencias que explícitamente alientan a realizar modificaciones, plantea la posibilidad de considerarlos como una fuente de material reutilizable.

Desde el año 1999, el SEI (Software Engineering Institute) comenzó a estudiar el F/OSS[33] con el objetivo de alcanzar un mayor conocimiento sobre el tema, contemplando el estudio de las prácticas de desarrollo involucradas y el mundo de usuarios de ese tipo de software.

La expresión “Software Libre” enfatiza los aspectos éticos por sobre los técnicos. Hay quienes prefieren referirse a estos productos como “Software de Código Abierto”, destacando la disponibilidad de las fuentes. Ambas expresiones difieren semánticamente, si bien se refieren a un conjunto prácticamente equivalente.

La Free Software Foundation es la organización que establece la definición Software Libre[34], en virtud del cumplimiento de las “cuatro libertades”:

- Ejecutarlo como se quiera y con cualquier propósito

- Estudiarlo y modificarlo para que se adecue a las necesidades del usuario
- Redistribuir las copias
- Distribuir versiones modificadas. Para esto es imprescindible que se entregue junto al código fuente.

La Open Source Initiative, en tanto, especifica los criterios que debe cumplir un software para considerarse de “Código Abierto”[35]:

- Libre redistribución
- Distribución con el código fuente
- La licencia debe permitir trabajos derivados
- Se pueden contemplar licencias que restrinjan la modificación del código original, en tanto se distribuya con archivos “parche” mediante los cuales pueda generarse trabajos derivados
- La licencia no debe discriminar entre personas, grupos de personas o campos laborales.
- La licencia no debe ser específica del producto
- La licencia no debe establecer restricciones a otro software (en particular, aplicaciones que se distribuyan junto al producto en cuestión)
- La licencia debe ser tecnológicamente neutral

Ambas definiciones describen conjuntos similares, aunque desde puntos de vista diferentes y enfatizando valores diferentes, como lo detalla Richard Stallman[36].

En este trabajo usaremos la denominación “Software Libre y de Código Abierto”, con la sigla F/OSS por las palabras en inglés “Free and Open Source Software”; de esta forma reflejamos el universo de software sobre el cual planteamos la tarea, sin adentrarnos en las discusiones filosóficas y políticas. No obstante, nos parece necesario destacar que las problemáticas planteadas en el debate descrito son de enorme relevancia, aunque no se aborden de manera directa en este trabajo debido a que exceden el alcance planteado para su desarrollo.

1.1.5.1 Dificultades para la reutilización de F/OSS

El F/OSS constituye una fuente abundante y accesible de potenciales componentes de software a bajo costo. No obstante, tales componentes no siempre constituirán una alternativa viable o económicamente conveniente.

La oferta de aplicaciones, componentes, bibliotecas, etc. que se distribuyen bajo licencias libres es enorme pero también extremadamente heterogénea[37] [38]. La calidad de los productos, la información disponible y la documentación que acompaña a los proyectos es completamente disímil, de modo que el conjunto que puede adecuarse a las necesidades de usuarios y desarrolladores es mucho menor que el universo F/OSS.

Las licencias del F/OSS promueven la adaptación y modificación; sin embargo, una estructura deficiente y una pobre documentación pueden volver extremadamente dificultosas estas tareas[39].

En este sentido, es necesario disponer de información confiable y de criterios de evaluación y selección que faciliten la adopción de este tipo de componentes.

Como mencionamos más arriba, hay una gran heterogeneidad en la oferta de software libre. Un estudio de Shweik [40] presentado en 2013 estima que más del 67% de los proyectos de F/OSS son abandonados.

La información disponible es también muy diversa en los distintos proyectos. Se puede observar que el proceso de corrección de errores se trata de manera dispar y la documentación es frecuentemente escasa[37] [41].

En su tesis doctoral[42], Manuel Sojer analiza factores que inciden en la reutilización de código de F/OSS, tanto en el marco de otros proyectos F/OSS como en desarrollos comerciales.

Lo que tienen en común los proyectos de F/OSS es su distribución junto con el código fuente. En virtud de estas consideraciones, el presente trabajo se orienta principalmente al análisis de código fuente con miras a obtener información relevante respecto de la posible reutilización de F/OSS.

1.1.6 Investigación y F/OSS

A primera vista, la reutilización de F/OSS comparte los problemas de la reutilización en general e introduce algunos nuevos. Muchos de los productos F/OSS no fueron desarrollados cuidadosamente con miras a la reutilización, la documentación puede ser deficiente y las funcionalidades que cumple se presentan en formatos variados y con niveles de detalles disímiles.

Sin embargo, la enorme difusión del F/OSS y el significativo éxito de algunos de sus productos (Apache, Linux, BIND, por citar a algunos) lo ha convertido en un tema de creciente interés para los investigadores[38] [43].

Von Krogh[44] señala que el F/OSS presenta un a serie de características especialmente

atractivas para los investigadores. Dos de ellas resultan particularmente relevantes para nuestro trabajo: la *transparencia*, en virtud de la cual se dispone de una enorme cantidad de datos sobre desarrollos reales; y la *proximidad*, lo que se refiere a las similitudes entre el proceso de desarrollo del F/OSS y la producción de conocimientos en el ámbito de las ciencias.

En los últimos años se observa un incremento en el número de estudios que aprovechan la masa de datos disponibles para emplear técnicas estadísticas para desentrañar diferentes aspectos del desarrollo[45]. En particular, se han presentado numerosas publicaciones relacionadas con la evolución del software[46], [47].

1.2 Alcance de este trabajo

Las consideraciones presentadas más arriba nos permiten precisar mejor el alcance de este trabajo. Se advierten las similitudes con estudios realizados en otras áreas de la Ingeniería de Software, las que servirán de necesaria referencia teórica en el resto de este trabajo.

El siguiente gráfico muestra los distintos aspectos que inciden en la reusabilidad y que resultan de relevancia para el presente trabajo.

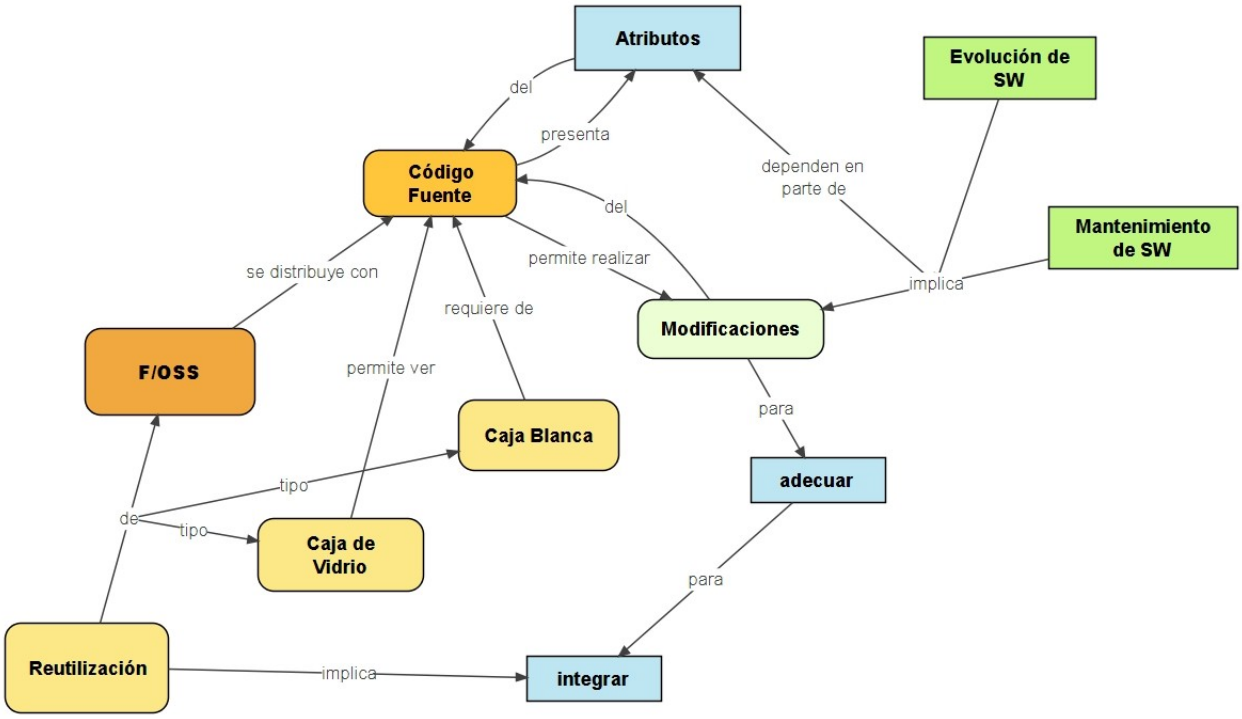

Ilustración 1: Aspectos que inciden en la reusabilidad de una pieza de software

A partir de las consideraciones precedentes, nos planteamos en este trabajo:

- Realizar una revisión del estado del arte respecto de la mantenibilidad, evolutividad y reusabilidad del software, especialmente en lo que respecta a la vinculación entre características del código fuente y las posibilidades de mantenimiento, evolución y reutilización.
- Mediante la investigación empírica, detectar características que permitan evaluar la potencialidad de reutilización de una aplicación de F/OSS, comprendiendo factores como la reusabilidad, adaptabilidad y/o modificabilidad.
- Aportar elementos para la reutilización de F/OSS de tipo horizontal, composicional, de caja gris, de caja blanca o de caja de vidrio
- Generar criterios de evaluación de recursos F/OSS en base principalmente a información obtenida a partir de código fuente. La investigación se centrará en desarrollos orientados a objetos, que permitan obtener información del diseño a partir del código fuente.
- Proponer un marco de trabajo que permita la incorporación de productos F/OSS al desarrollo con reutilización
- Desarrollar una herramienta de apoyo al proceso de selección de F/OSS para la reutilización en base a métricas de código fuente que hayan surgido como relevantes de la investigación empírica.

En los capítulos siguientes se repasarán diferentes aproximaciones al problema de la evaluación de la facilidad o dificultad de modificación de software, no sólo en lo atinente a la adaptación para la reutilización sino también a la corrección de errores o la incorporación de nuevas funcionalidades.

Las secciones posteriores se centrarán en la investigación empírica realizada. En primer lugar se presentarán las consideraciones metodológicas; a continuación se expondrán los trabajos realizados y los resultados obtenidos en esas experiencias.

A continuación se describe el proceso de elaboración y se presenta el marco de trabajo para la reutilización de F/OSS; para ello se partirá de un desarrollo experimental a partir del cual exploramos los distintos aspectos que intervienen en la reutilización de F/OSS. Posteriormente se presentará una revisión de distintos enfoques relativos a la evaluación de F/OSS y a la selección de recursos para la reutilización, incluyendo trabajos específicos sobre el tipo de software que nos ocupa; en base a estos antecedentes, y las consideraciones surgidas de la investigación empírica, se expondrá un marco de trabajo

para la reutilización de F/OSS.

Posteriormente, se tratará el desarrollo de la herramienta planteada en base a los resultados de la investigación empírica, y como aplicación del marco presentado anteriormente.

Finalmente, se expondrán las conclusiones generales de esta tesis y se propondrán trabajos a futuro.

Capítulo 2.- La evaluación de la viabilidad de modificar el software

La reutilización de software implica disponer de elementos (código, ejecutables, documentos, etc.) que se incorporan a un nuevo desarrollo.

Estos recursos pueden no cumplir exactamente con las funciones que requiere un desarrollo específico, o las interfaces que ofrecen pueden no coincidir con las que esperan los elementos con los que interactuará.

En virtud de estas consideraciones, uno de los aspectos que habrá que tener en cuenta al momento de seleccionar un recurso para la reutilización es la posibilidad de modificarlo (en reutilización de caja blanca) o extenderlo (reutilización de caja gris). Por otra parte, el conocimiento del código fuente habilita la reutilización de caja de vidrio, ya que posibilita estudiar la forma en que trabaja, así como la realización de evaluaciones basadas en las características del mismo.

El software suele considerarse maleable o modificable, dado que no se trata de una manufactura clásica sino una entidad que se desarrolla y modifica mediante el intelecto y que -a diferencia de los productos manufacturados- no se somete a las leyes del mundo físico ni se “desgasta”[19]. Sin embargo, la modificación de código o la adaptación requiere comprender la estructura, y a menudo las alteraciones traen aparejados nuevas dificultades.

Los problemas relacionados con la modificación de software afectan a distintas etapas de su ciclo de vida. En las diferentes fases del desarrollo se realizan habitualmente cambios en virtud de la detección de errores, las variaciones en los requerimientos (especialmente en modelos de desarrollo iterativos o bajo metodologías ágiles) u otras consideraciones/

Una vez que un software se encuentra en producción, es posible que se detecten nuevos errores que deban ser corregidos, constituyendo una etapa de “mantenimiento”. El nombre alude claramente al ciclo de vida en otros ámbitos ingenieriles[48].

Para que un software pueda seguir cumpliendo su función, es posible que deba modificar sus características, incorporando nuevos requerimientos o adecuándose a nuevos contextos. Estas características se estudian bajo el concepto de Evolución del Software[49] [50]. Suele denominarse evolutividad a la facilidad para el desarrollo futuro de un software[51].

Este repaso nos muestra que el problema de la modificabilidad del software se aborda

desde las perspectivas del mantenimiento y la evolución, por lo que constituyen áreas afines a nuestro trabajo.

En esta sección revisaremos conceptos y aproximaciones vinculadas a la evaluación de la modificabilidad del software, sea desde las perspectivas del mantenimiento, la evolución del software, y la adecuación para la reutilización.

En cada caso, repasaremos diferentes enfoques para la medición de estas características. Una revisión más completa, de algunos años atrás, puede verse en “Mantenibilidad y Evolutividad en el Software Libre y de Código Abierto”[11], donde se presenta una recopilación bibliográfica.

No existe un consenso amplio en cuanto a la relación entre las diferentes métricas y los atributos que se pretende evaluar; por eso, en la última parte del capítulo presentamos una revisión bibliográfica sobre la determinación de umbrales de métricas, indicadores que pueden servir de referencia o advertencia sobre algunas características del software potencialmente riesgosos.

2.1 Mantenimiento y mantenibilidad

Como se menciona más arriba, citando a Pressman[19], el software no se desgasta; sin embargo, y aunque suena contradictorio, sí se deteriora.

La definición de mantenimiento planteada por Grubb y Takang[52] pone de relieve las circunstancias bajo las cuales se realizan tareas de modificación del software. Según estos autores, “Mantenimiento” es el acto de mantener una entidad en un estado de conservación, eficiencia o validez; preservándolo de futuras fallas o declinación[52].

Esta definición contempla no sólo el mantenimiento operacional estricto sino también la adecuación para que una entidad (en este caso, un sistema de software) pueda seguir cumpliendo adecuadamente su papel.

Los autores mencionados clasifican las modificaciones del software en función de la causa que las origina. Señalan que los cambios en el software caen bajo alguna de las siguientes categorías:

- Modificaciones iniciadas por defectos en el software
- Cambios orientados por la necesidad de adecuar el software a modificaciones en el entorno en el que se utiliza.
- Cambios para expandir los requerimientos del sistema
- Modificaciones para prevenir mal funcionamiento posterior

Martin y McClure[53] señalan por su parte que el mantenimiento se realiza para:

- Corregir errores
- Corregir fallas de diseño
- Interfaz con otros sistemas
- Realizar mejoras
- Realizar cambios necesarios en el sistema
- Realizar cambios en archivos o bases de datos
- Mejorar el diseño
- Modificar software para posibilitar el uso de diferente hardware, software, características del sistema o facilidades de comunicación

Este listado muestra una serie de circunstancias bajo las cuales se realizan modificaciones en el software; algunas de ellas se vinculan claramente con las que requeriría la reutilización de caja blanca, como la interfaz con otros sistemas y la adaptación a nuevos contextos enumerados en el último punto.

2.2 Mantenimiento de software como parte del ciclo de vida

Erdil, Finn y otros[54] analizan el rol del mantenimiento de software bajo distintas metodologías de desarrollo y diferentes modelos de procesos. Citan a Lientz y Swanson[55] quienes definen 4 tipos de mantenimiento: correctivo, adaptativo, perfectivo y preventivo.

Ilustran con diversos ejemplos el alto costo que puede suponer el mantenimiento dentro del ciclo de vida general del software.

A continuación, consideran las características del software que inciden en el mantenimiento.

También mencionan el trabajo de Martin y McClure[53], quienes destacan los siguientes factores: tamaño del sistema, edad del sistema, cantidad de ítem de datos de entrada/salida, tipo de aplicación, lenguaje de programación y el grado de estructura.

Los mismos autores consideran los factores que disminuyen el costo de mantenimiento, mencionando la utilización de técnicas estructuradas, uso de software moderno, utilización de herramientas automatizadas, uso de técnicas de bases de datos, buena administración de datos y mantenedores experimentados.

El mantenimiento puede verse desde el punto de vista externo[56] [11], en relación con el proceso de corrección de errores, adaptación y mejora; los atributos internos, entre los que se encuentran los atributos estructurales y las características de diseño, inciden en la facilidad o dificultad de los procesos. Sin embargo, es difícil de establecer la relación entre las características internas y la facilidad de modificación.

2.3 Medición de la mantenibilidad y la evolutividad

En este trabajo adoptamos la definición del Instituto de Ingeniería de Software (SEI) sobre mantenibilidad, a la que entiende como la facilidad con que puede modificarse un sistema o un componente de software para corregir fallas, mejorar su funcionamiento o adaptarlo a un nuevo ambiente[11].

Esta definición de carácter general no permite establecer cómo evaluar esta característica, ni qué atributos se vinculan con esta cualidad.

La investigación sobre este tema es de enorme relevancia, dado que diversos estudios establecen que el costo de la fase de mantenimiento constituye entre el 60% u el 90% del costo total[52].

No existe un consenso general en cuanto a cómo medir la mantenibilidad o la evolutividad. Se distinguen en general los siguientes enfoques:

- Comparación entre características estructurales y mantenibilidad.
- Evaluación de la mantenibilidad por expertos.
- Relación empírica entre atributos y mantenibilidad evaluada por expertos.
- Detección de debilidades de diseño, como los antipatrones, “bad smells” o “code smells”[57].

Se han propuesto diversos “modelos de mantenibilidad”[58] [59] [60] en los que se desagrega la cualidad que se intenta evaluar en atributos componentes, que podrían evaluarse de manera separada.

Ya en 1994 Coleman y otros[61] analizaron diferentes enfoques que intentaban asociar la mantenibilidad con un conjunto pequeño de métricas computadas sobre el código fuente. Mediante técnicas estadísticas evaluaron los rangos aceptables para diferentes medidas, de modo que la obtención de valores por fuera de esos rangos constituyan una señal de alarma sobre la mantenibilidad del código.

Heitlager y otros[62] proponen un modelo práctico que parte de considerar que las medidas de mantenibilidad utilizadas previamente suelen orientarse no tanto hacia las

características internas, como el código, sino más bien hacia las actividades de mantenimiento.

Observan que medidas como el índice de mantenibilidad (MI) propuesto por Oman (y que se detalla más abajo) proporciona datos empíricos que, sin embargo, no dan suficiente información sobre las características del software que inciden en alcanzar un valor determinado de ese índice.

En la siguiente sección, retoman el modelo de calidad ISO 9126, destacando los atributos en los que se desagrega la mantenibilidad en ese documento: analizabilidad, modificabilidad (versatilidad), estabilidad, testeabilidiad, adecuación a estándares o pautas de mantenibilidad.

Los autores presentan un relevamiento de las medidas referidas a la mantenibilidad que fueron propuestas por los documentos que acompañan al estándar en cuestión, las que se dividen en 16 métricas externas y 9 internas.

Las externas se refieren a las actividades de mantenimiento. Entre ellas se mencionan el “tiempo transcurrido para la implementación de una modificación”, como medida de la versatilidad. Los autores observan que entre las medidas internas se evalúa también las actividades, ejemplificando su afirmación con la métrica “impacto del cambio”, que cuenta las modificaciones realizadas y el número de problemas causados por esas modificaciones.

Los autores critican que estas medidas no reflejan tanto las características del producto en sí sino más bien la interacción del producto con el entorno, los encargados de pruebas y mantenimiento, y otros aspectos externos.

2.3.1 El índice de mantenibilidad

Son varios los factores que inciden en la mayor o menor dificultad para mantener un sistema de software. Una aproximación que busca disponer de un número que nos de una indicación valedera de esta característica compleja, es la de desarrollar una fórmula que pondere los distintos factores y que se calibre de manera empírica.

A partir del trabajo en Hewlett Packard, Coleman y otros[61] plantearon una fórmula en la que consideran:

- Tamaño, medido en líneas de código y Volumen de Halstead[56]. La premisa es que a mayor tamaño, mayor será la dificultad para realizar el mantenimiento
- Complejidad, medida en términos de la Complejidad Ciclomática de McCabe[63]. Mientras mayor sea la complejidad, mayor será la dificultad de realizar cambios

- Cantidad de comentarios. Un número mayor de comentarios se relaciona con mejores posibilidades de comprender el código.

La fórmula parte entonces de un valor máximo al que se le van restando ponderadamente los promedios del Volumen (V), la Complejidad Ciclomática (CCE), el tamaño medido en cantidad de Líneas de Código (LOC) y una función del porcentaje de líneas con comentarios (PorCom).

$$Mant = A - B \cdot \overline{V} - C \cdot \overline{CCE} - D \cdot \overline{LOC} - f\left(\overline{PorCom}\right)$$

Se establecieron los factores que integran la fórmula mediante la calibración empírica, emparejando los valores con la opinión de expertos respecto de la mantenibilidad de productos de software concretos. La fórmula propuesta por Hietlager[62] se detalla en el Anexo I.

2.3.2 Limitaciones en la medición de la mantenibilidad

Se han realizado numerosos estudios que intentan establecer correlaciones entre características medibles del software y la mantenibilidad.

En el ámbito de la Programación Orientada a Objetos, se ha explorado la vinculación de métricas de diseño como las de Chidamber y Kemerer[64] (Métricas CK) con relación a la mantenibilidad. Esta última, en general, se evalúa en función del juicio de expertos o de indicadores tomadas de la vista externa del mantenimiento.

Sjoberg y otros[65] cotejaron el Índice de Mantenibilidad, las métricas de Chidamer y Kemerer y dos Code Smells[57] considerados frecuentemente en la práctica como indicadores de mantenibilidad: Envidia de Características (Feature Envy) y Clase Dios, los que pueden detectarse a partir de métricas de código[66].

Los autores registraron las métricas correspondientes para cuatro sistemas de software funcionalmente equivalentes y luego encargaron a diferentes desarrolladores que realizaran diferentes tareas de mantenimiento iguales en cada uno de los productos analizados. Posteriormente, se solicitó a los desarrolladores que evaluaran distintos aspectos del proceso a los fines de poder comparar la mantenibilidad real de los productos.

El estudio reflejó en estos casos que los indicadores de mantenibilidad seleccionados no toman valores compatibles entre sí: mientras que -por ejemplo- el índice de mantenibilidad sugería que un sistema dado era más mantenible que otro, una métrica diferente los calificaba de un modo diferente. Más aún: el estudio sólo muestra posibles correlaciones significativas con el esfuerzo evaluado por los desarrolladores y el tamaño y la cohesión.

Este sencillo trabajo advierte sobre las limitaciones de las métricas como indicadores de mantenibilidad, lo que debe ponderarse en la gestión de los proyectos y en la selección de software.

2.4 Modificabilidad

La evolución, el mantenimiento y la adaptación de un software para su reutilización implican la modificación del mismo. A pesar de la importancia que tendría disponer de indicadores acerca de este aspecto, se han publicado pocos estudios que intentan medir la posibilidad de modificación de un software en función de métricas de diseño o de código fuente; puede verse que la búsqueda de publicaciones que incluyan las palabras “software” y “changeability” junto a “measurement”, “metrics” o “assessment” en la biblioteca digital de la ACM entre los años 2000 y 2015 arroja sólo 7 resultados (al 3 de junio de 2015).

En el año 2001 Kabaili y otros [67] buscaron posibles correlaciones entre métricas vinculadas a la cohesión de métodos y clases (LCC y LCOM) con la modificabilidad del software resultante. Para ello indagaron sobre el efecto de esas métricas sobre el impacto de cambio, entendido como el número de clases que se debieron modificar como consecuencia de realizar el cambio en una de ellas. Los software analizados fueron tres: xForm, ET++ (distribuidos bajo licencias libres) y una aplicación privativa provista por la entidad que financiaba el proyecto.

Estos autores concluyeron que no había elementos que permitieran correlacionar las métricas de cohesión mencionadas con la modificabilidad. Además, observaron mediante análisis manual que algunas clases cuyos valores de estas métricas era bajo, presentaban alto grado de cohesión.

Ayalew y Mguni [68] analizaron la relación de la modificabilidad con las métricas CBO (acoplamiento entre objetos), WMC (métodos ponderados por clase), Ca (Acoplamiento aferente) y Ce (Acoplamiento eferente). Para ello observaron en una aplicación libre -OpenBravo- los valores computados de estas métricas para cada clase y las tareas de mantenimiento que se realizaron en ellas.

Para el caso estudiado, los autores observaron una importante correlación entre todas esas métricas y la cantidad de tareas de mantenimiento, de acuerdo al cómputo del coeficiente de Pearson. De todas las métricas bajo consideración, CBO muestra la correlación más fuerte.

Este resumen pone de relieve que no se han alcanzado conclusiones determinantes sobre este tema.

2.5 Umbrales

A pesar de las limitaciones señaladas por Sjøberg[65] y la insuficiente base empírica, es posible que los valores que se registran en distintas métricas sirvan como referencia para tener en cuenta respecto de las posibilidades de modificación del software.

Un enfoque que busca sistematizar la información sugerida por las métricas es la de establecer umbrales: valores límites que cuando se registran cifras por fuera de ellos llamen la atención sobre posibles debilidades en el diseño o en el código.

En general, los trabajos empíricos realizados para establecer umbrales apuntan a determinar valores útiles para la evaluación de clases particulares; en los capítulos posteriores adoptamos una perspectiva diferente, buscando referencias sobre los valores promedio de determinadas métricas en toda una aplicación o componente, de modo de advertir acerca de posibles debilidades de diseño en una pieza de software en su conjunto.

2.5.1 La búsqueda de umbrales

Se han propuesto diversos umbrales para métricas de diseño orientado a objetos definidas mediante diversos métodos. Por un lado, algunos trabajos se basan en la distribución de los valores de las métricas, buscando revelar cuáles serían cifras atípicas para los indicadores que consideran[69] [70] [71] [72] [73]. Otros, en cambio, utilizan técnicas estadísticas que intentan relacionar las mediciones obtenidas con atributos asociados a la mantenibilidad o a la propensión a errores de alguna parte de código o un elemento de diseño[74] [75] [76] [77]. Cabe señalar que el conjunto de métricas analizadas en la mayoría de los estudios son las propuestas por Chidamber y Kemerer[64], y Brito e Abreu y Carapuça[78] entre otras de uso común.

A grandes rasgos, se pueden clasificar los estudios en dos grupos, según el método adoptado: por un lado, algunos investigadores se centran en el análisis de las distribuciones de frecuencia de las métricas a fin de detectar valores que se puedan considerar extremos; otros, en cambio, buscan establecer correlaciones estadísticas con características del software, evaluando estas últimas según indicadores o juicios de expertos.

A continuación, presentamos una revisión sobre diferentes abordajes para la búsqueda de umbrales.

2.5.2 Umbrales propuestos según distribuciones de frecuencia de las métricas

Rosenberg [69] utilizó histogramas para analizar el efecto de los valores de las métricas

de Chidamber y Kemerer[64] sobre la calidad del software. Los valores de los umbrales se basan en el análisis de los histogramas, aunque no se analiza la posible relación entre esos valores y la probabilidad de que módulos presenten errores o deban modificarse.

Erni y Lewerentz[70] analizaron la distribución de métricas orientadas a objetos postulando como potenciales umbrales a la media ± la desviación estándar. Los autores plantean la adopción de uno de esos valores en función de los problemas que plantearía en cada caso cifras demasiado altas o demasiado bajas. Por ejemplo, si para una métrica se considera que un número alto es perjudicial, se adopta μ+σ como umbral (siendo μ la media y σ la desviación estándar).

Chidamber y otros[71] consideraron las distribuciones empíricas de las métricas sugiriendo que el tope para definir un valor como “alto” sería el 80º percentil (es decir, el 80% de las muestras alcanzarían valores por debajo del umbral señalado).

Varios autores apelaron a la regresión logística para estudiar la relación entre valores de las métricas y la propensión a fallos, evaluando la efectividad del establecimiento de umbrales.

Lanza y Marinescu[72] también propusieron umbrales para métricas comunes partiendo de la distribución de los valores en un grupo de 45 aplicaciones escritas en Java. Clasificaron los valores de las medidas según sus frecuencias en bajo, medio, alto y muy alto.

Ferreira y otros[73] apelaron al análisis de distribuciones para indagar sobre posibles umbrales para las métricas COF (Factor de Acoplamiento), cantidad de métodos públicos, cantidad de campos (atributos) públicos, LCOM (Falta de Cohesión en los métodos) y DIT (Profundidad del árbol de herencia), estos dos últimos propuestos por Chidamber y Kemerer. Analizaron 40 aplicaciones escritas en Java, obtenidas de SourceForge. Las métricas fueron computadas con la herramienta Connecta, y las distribuciones de frecuencia se ajustaron a distribuciones teóricas mediante la herramienta EasyFit. Los autores dividieron la muestra según diferentes dominios, planteando diferentes umbrales para cada uno de ellos.

2.5.3 Estudios de umbrales en base a correlaciones estadísticas

Como se mencionó anteriormente, otra estrategia utilizada para establecer posibles umbrales es la de analizar la relación entre determinadas métricas y atributos vinculados a diferentes aspectos de la calidad del software. Los trabajos que se resumen a continuación apuntan a determinar valores de las métricas que pudieran advertir sobre debilidades en un módulo o una clase en concreto.

Benlarbi y otros[74] plantearon la determinación de umbrales para los valores de las métricas orientadas a objetos. En el estudio consideran a los umbrales como valores heurísticos que sirven para determinar rangos de valores deseables y no deseables para métricas de software.

Los autores analizaron dos grandes sistemas de telecomunicaciones escritos en C++, extrayendo los valores de las métricas propuestas por Chidamber y Kemerer en busca de determinar si existen valores más allá de los cuales existe una mayor incidencia de fallos. Las métricas en cuestión fueron seleccionadas por ser altamente referenciadas en la Ingeniería de Software. Los sistemas analizados constan de 85 y 172 clases respectivamente.

Utilizaron para realizar las mediciones una aplicación comercial que no se describe en el texto. Cada clase fue clasificada como “defectuosa” o “no defectuosa” según si se reportaba al menos una falla durante la operación del sistema.

Los autores concluyen que el establecimiento de umbrales no aporta ventajas respecto de un modelo continuo; es decir, no logran determinar valores en ninguna de las métricas que al ser sobrepasadas impliquen claramente un incremento en la probabilidad de que una clase sea defectuosa.

Shatnawi realizó diversos estudios buscando determinar umbrales a partir de diferentes análisis estadísticos. En unode ellos [75], emplea la curva ROC para proponer umbrales de un conjunto de métricas CK y de Lorentz. Ese método permite evaluar la pertinencia de un umbral para clasificar en dos categorías (en este caso, en defectuoso y no defectuoso).

En un trabajo más reciente [76], Shatnawi utiliza regresión logística (como hiciera anteriormente Benlarbi) para estudiar valores aceptables para un conjunto de métricas. Mediante este método pretende establecer valores que pueden advertir si una clase es defectuosa.

El autor analiza los bugs consignados en el software Eclipse a lo largo de su evolución, registrados en el sistema Bugzilla. Vincula esos errores a las modificaciones efectuadas en las clases para corregir cada error, de modo de clasificar a las clases como defectuosas o no defectuosas.

Shatnawi propone umbrales en base al estudio de una versión del entorno Eclipse, popular IDE escrito en Java, comprobando luego su aplicabilidad con una versión posterior del mismo software. Posteriormente, compara los valores determinados con los obtenidos con el mismo método para otros dos productos (Rhino y Mozilla), observando que no pueden generalizarse las cifras postuladas en el primer caso.

El autor también compara los umbrales calculados con los propuestos por Rosenberg[69], concluyendo que los valores propuestos por esta autora son menos seguros para estimar clases defectuosas.

Kaur[77] y otros obtuvieron las métricas CK y otras indicativas de posibles “bad smells” sobre dos versiones del software JFreeChart. Realizaron un análisis basado en regresión logística, estimando posibles umbrales en base al Nivel Aceptable de Riesgo (Value for Acceptable Risk Level, VAR) de acuerdo a la metodología presentada por Bender[79] .

De este resumen se observa que los objetivos buscados en los trabajos mencionados se relacionan con la determinación de métricas que permitan caracterizar piezas particulares del software (por ejemplo, clases y métodos específicos) y no el diseño de la aplicación en su conjunto.

Capítulo 3: Metodología

Como planteamos al comienzo, este trabajo se propone revelar características del código fuente que constituyan indicadores de modificabilidad, entendida como la facilidad de realizar cambios en el código.

Para ello, se plantearon las siguientes actividades

- analizar las características de las fuentes de las que se obtendrán los datos, consignando las dificultades y las debilidades de las mismas.
- realizar estudios de caso que permitan alcanzar mayor comprensión del problema y elaborar hipótesis que conduzcan a nuevas investigaciones. De estos estudios surgirán métricas de código fuente y de diseño (derivadas de las primeras) candidatas a revelar información sobre las posibilidades de modificación del software
- relevar las métricas candidatas detectadas en la actividad anterior sobre un conjunto más amplio de productos F/OSS, que permita realizar análisis estadísticamente significativos.
- Establecer umbrales para las métricas mediante el análisis de las distribuciones de los promedios sobre un conjunto masivo de versiones de F/OSS

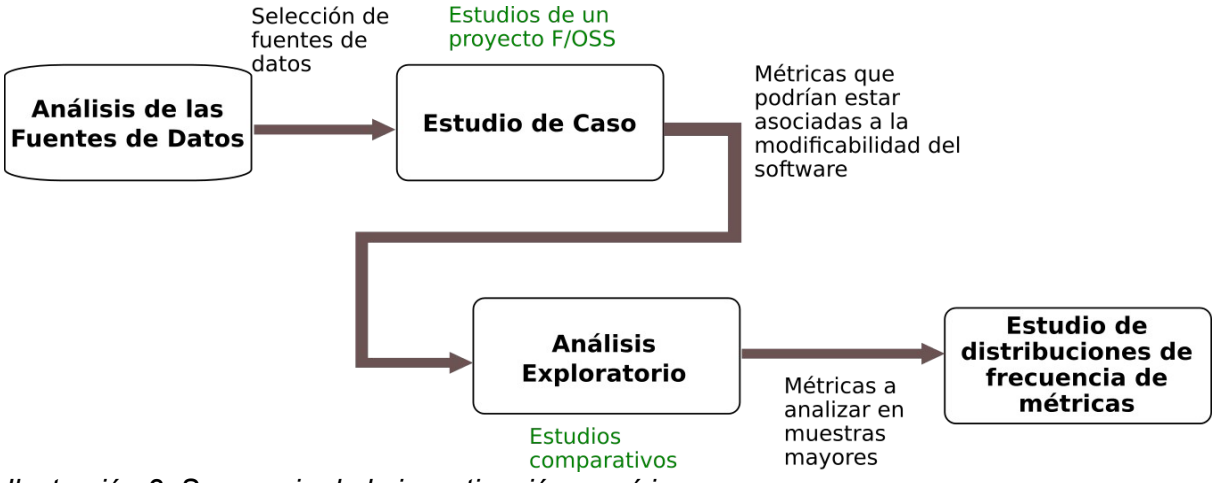

Ilustración 2: Secuencia de la investigación empírica

3.1 La investigación empírica en Ingeniería de Software y F/OSS

En la literatura sobre investigación empírica se observa una gran heterogeneidad en el uso de los términos[80]. Otro aspecto relevante es que el tamaño de las muestras sobre las cuales se basan las conclusiones frecuentemente es muy limitada, por lo que resulta difícil determinar el alcance de las conclusiones[81]. Si bien existen aproximaciones que permitirían en algunos casos ampliar el tamaño sobre el cual se realizan los análisis[82], las características específicas del F/OSS permiten potencialmente superar las limitaciones señaladas.

La investigación sobre F/OSS ha ido incrementándose en los últimos años, lo que se refleja también en la creciente aceptación de trabajos en diferentes eventos y revistas científicas[83].

Como se menciona en la introducción, el estudio del F/OSS abre un campo enorme de posibilidades de investigación, dada la gran masa de información públicamente disponible. No obstante, las fuentes de datos presentan una heterogeneidad muy grande en cuanto a los contenidos, la calidad y la representación de la información, entre otros aspectos[84] [37]. Esto significa un escollo que debe abordarse para obtener conclusiones valederas.

Gasser y otros[84] exponen las características esenciales que debería tener un estudio sobre F/OSS, de acuerdo con la siguiente tabla:

Característica	Detalle
Reflejar experiencia real	No debe tratarse de fenómenos construidos artificialmente
Dar cobertura adecuada	Tanto a los fenómenos que ocurren naturalmente como a perspectivas alternativas y marcos analíticos
Examinar niveles representativos de variabilidad	Atender los rangos completos de las dimensiones y fenómenos clave
Demostrar significatividad estadística	Seleccionar las muestras y los tamaños de las muestras a los efectos de alcanzar mayor generalidad en los resultados obtenidos
Proveer resultados comparables entre proyectos	Posibilitar la comparación de los resultados obtenidos entre distintos proyectos, diferentes comunidades o dominios de aplicación
Proveer resultados que pueden ser reconstruidos, probados, evaluados, extendidos y redistribuidos por otrxs	Posibilitar la replicabilidad de las experiencias, de modo de que otros investigadores puedan verificar, refutar o complementar las conclusiones que se alcancen

Tabla 3: Características esperables de la investigación sobre F/OSS

Estas consideraciones sirven de guía para el diseño y el desarrollo de las actividades de investigación.

3.2 Etapas de la investigación empírica

El desarrollo de la tarea de investigación siguió una serie de pasos sucesivos, donde cada etapa brindaba elementos para la instancia posterior, de acuerdo con la siguiente lista:

- Estudio de las fuentes de datos
- Estudio de caso
- Estudio comparativo de carácter exploratorio
- Análisis de un conjunto mayor de datos
- Elaboración de conclusiones y propuesta de trabajos futuros

3.3 Análisis de las Fuentes de datos

Los proyectos de F/OSS pueden publicar su información en repositorios (Como SourceForge, GoogleCode, GitHub, etc.), en sitios web específicos o incluso en espacios académicos[81], [84].

Observando la información disponible para cada proyecto de F/OSS en los repositorios

SourceForge y GitHub se verifica que no sólo se ofrece el acceso al código fuente sino también a una serie de otros artefactos relacionados con el proceso de desarrollo.

Estos repositorios brindan alojamiento (hosting) a los proyectos mediante sistemas que brindan a los desarrolladores una serie de recursos para la gestión.

Uno de los elementos que podría brindar información relevante sobre la factibilidad de modificación del código es el sistema de gestión de errores. SourceForge, por ejemplo, posibilita a los desarrolladores administrar “trackers” donde se registran “artefactos”, que pueden referirse típicamente a solicitudes de mejora o errores detectados en el software; se analizó la posibilidad de recopilar esta información para comprender los procesos de mantenimiento del F/OSS en el trabajo “Indicadores de la utilización del Bug Tracker en proyectos de F/OSS” [41].

En un análisis de los 15 proyectos mejor rankeados al 29 de mayo de 2009 bajo la categoría Enterprise de SourceForge se verificó que la utilización del Bug Tracker es heterogénea, registrando en algunos casos una alta proporción de bugs reportados que son cerrados sin que conste resolución[41].

Este trabajo puso en evidencia que esa información es demasiado fragmentaria como para ser pasible de tratamiento estadístico del proceso de resolución de errores. No obstante, brinda elementos para desestimar esa información en estudios de caso que analicen métricas de mantenimiento (ver Kan [85]) cuando se verifica una alta proporción como la descrita en el párrafo anterior.

Teniendo en cuenta las limitaciones mencionadas, se optó por el estudio del código fuente en proyectos de F/OSS maduros, que hubieran producido varias versiones a lo largo del tiempo. Para ello se descargó el código de cada una de las versiones publicadas, registrando la fecha de emisión de cada una de ellas, a los efectos de analizar posteriormente la secuencia de versiones.

3.4 Métricas Computadas

En la literatura hay una enorme heterogeneidad en cuanto a las métricas que se relacionan con la modificabilidad del software[86] [64] [87] [88]. Esta falta de consenso sobre el tema es una de las motivaciones del presente trabajo.

Se opta aquí por una visión global del software a estudiar; se indagará sobre las características generales que presente el código fuente.

Por ello se adopta la propuesta de Lanza y Marinescu[72], quienes obtienen una serie de métricas en base a las cuales presentan una perspectiva de conjunto. Estos autores señalan que las métricas por separado aportan poca información para caracterizar el

diseño en su conjunto; por ello, plantean el uso de proporciones que brinden un panorama de carácter más general.

Las métricas computadas se detallan en la tabla siguiente.

Métrica	Descripción
CYCLO	Complejidad ciclomática, según la definición de McCabe[63]
LOC	Líneas de código, incluyendo líneas en blanco y comentarios
NOM	Cantidad de métodos y otras operaciones definidas en toda la aplicación
NOC	Cantidad de clases
NOP	Cantidad de paquetes
CALL	Cantidad de invocaciones a operaciones
FOUT	Suma de las clases referenciadas por cada una de las clases

Tabla 4: Métricas computadas con la herramienta iPlasma, utilizadas en el presente trabajo

Cada una de estas métricas arroja un valor global dependiente del tamaño del software. Es esperable que una aplicación más grande tenga un mayor número de clases, mayor cantidad de líneas de código, etc.

Lanza y Marinescu plantean la caracterización del diseño de un software en base a un conjunto de proporciones entre estas métricas. La siguiente tabla muestra las proporciones en cuestión.

Nombre	Descripción	Fórmula
Estructuración de alto nivel	Cantidad de clases por paquete	NOC/NOP
Estructuración de clases	Cantidad de operaciones en relación con le número de clases	NOM/NOC
Estructuración de operaciones	Promedio de la cantidad de líneas de código por operación	LOC/NOM
Complejidad de operación intrínseca	Cantidad de caminos independientes en relación con el tamaño en líneas de código	CYCLO/LOC
Intensidad de acoplamiento	Proporción de llamadas a operaciones en relación con la cantidad total de operaciones	CALL/NOM
Dispersión de acoplamiento	Proporción de las llamadas de cada clase a métodos de otra respecto del total de llamadas	FOUT/CALLS

Tabla 5: Proporciones postuladas por Lanza y Marinescu para la caracterización del

diseño a partir del código fuente

3.5 Herramientas empleadas en la recolección de datos

En este trabajo, uno de los criterios para diseñar las tareas de investigación es la posibilidad de que sea repetido y revisado por otros/as investigadores.

En virtud de esa definición, las herramientas utilizadas para la obtención de las métricas del software, el análisis estadístico y la elaboración de los informes son de acceso libre.

Para la medición de las distintas métricas enumeradas en el punto anterior, se utilizó iPlasma. Se trata de una aplicación para la gestión integrada de la calidad de software. desarrollada en la Universidad Politécnica de Timisoara, que puede descargarse libremente de http://loose.upt.ro/reengineering/research/iplasma. El desarrollo de esta aplicación está vinculada al trabajo de Lanza y Marinescu referido más arriba [72], por lo que las métricas computadas por esa herramienta[89] son las que se describen en el apartado anterior y se detallan en la tabla 4.

La utilización de esta herramienta permite comparar los resultados con los obtenidos por los autores mencionados; no obstante, es necesario señalar que al no disponer del código fuente no siempre es posible precisar de qué manera esta herramienta obtiene las métricas. La descripción de los algoritmos usados puede no ser suficiente para asegurar que otros investigadores obtengan resultados comparables[90].

A los efectos de considerar la posible incidencia de la métrica LOC respecto del tamaño, se computó también SLOC (Líneas de Código Fuente), obtenidas mediante la herramienta SLOCCount[91].

Para realizar el tratamiento de los datos también se tuvo en cuenta la replicabilidad, por lo que se utilizaron las siguientes herramientas libres: la planilla de cálculo OpenOffice Calc y el entorno de tratamiento estadístico R, el que puede obtenerse de http://www.r-project.org.

Capitulo 4: Investigación Empírica

4.1 Estudio de caso

De acuerdo con la definición adoptada por Runeson[80], un estudio de caso en ingeniería de software *es una investigación empírica que se vale de diferentes fuentes de evidencia para investigar una instancia (o un pequeño número de instancias) de un fenómeno contemporáneo de ingeniería de software dentro de su contexto en la vida real, especialmente cuando los límites entre el fenómeno y el contexto no pueden ser claramente especificado*.

La información públicamente disponible de los proyectos F/OSS reflejan el desarrollo en su contexto específico; por lo tanto, la investigación en base a esa información se inscribe en la definición del párrafo anterior.

Se realizó un estudio de caso con el fin de observar la evolución de un producto F/OSS[9] maduro, que produjo versiones a lo largo de muchos años y que mantiene una posición alta en el ranking de SourceForge. El proyecto elegido fue Sweethome 3D, aplicación de diseño de interiores escrito en Java que publicó más de 30 versiones desde el 12 de setiembre de 2007, cuando se publicó la versión 1.0.

El trabajo se presentó en el XVI Congreso Argentino de Ciencias de la Computación en el año 2010 bajo el nombre de “Evaluación de la Evolución del Diseño en F/OSS: un caso de estudio”.

A los efectos de brindar un marco de interpretación y revisión del caso de estudio[80], el desarrollo del trabajo partió de las teorías de la Evolución del Software, trabajos empíricos relacionados con las afirmaciones enunciadas en dichas leyes y en los trabajos relacionados en cuanto a investigación empírica sobre la evolución y sobre las particularidades de la evolución en F/OSS. Se tomó como referencia las publicaciones sobre la Evolución del Software de Lehman y Fernández-Ramill que dieron origen a las llamadas “Leyes de la Evolución” del Software [92] [93] [94], las indagaciones sobre la evolución de F/OSS a partir de casos de estudio[46] [95] [96] [97], análisis comparativos que comprenden diferentes productos[98], e incluso algunas investigaciones sobre grandes cantidades de proyectos, pasibles de tratamiento estadístico más general[99] [100].

Se registraron las métricas mencionadas anteriormente para cada una de las versiones publicadas por el proyecto Sweet Home 3D hasta ese momento, prestando atención a los siguientes aspectos:

- Si el software analizado crece con el tiempo, en términos de líneas de código y de cantidad de clases.
- Si se verifican indicios de incremento de la complejidad en las sucesivas versiones
- Si se observan síntomas de desaceleración o estancamiento en el desarrollo.

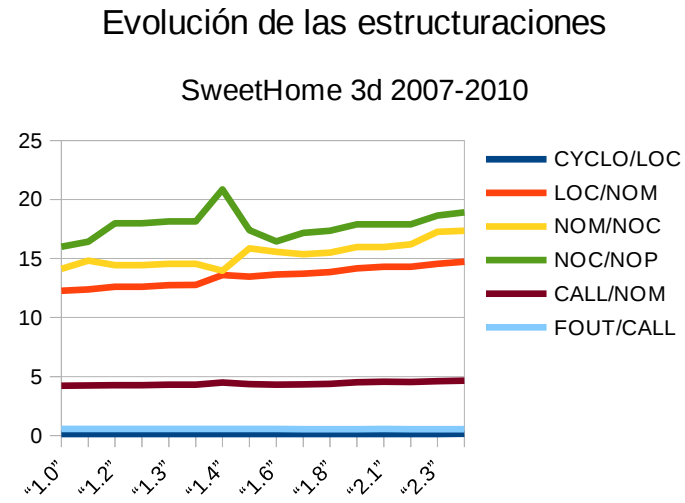

Ilustración 3: Evolución de distintas proporciones de métricas en las versiones publicadas por el proyecto Sweet Home 3D

A partir de los datos obtenidos, se observó cómo variaban las métricas y las proporciones descritas anteriormente, a lo largo de las distintas versiones publicadas; prestamos atención particularmente a la detección de posibles tendencias, buscando verificar eventuales correlaciones entre las versiones, el tamaño y las proporciones que caracterizan a las cuatro estructuraciones mencionadas, de acuerdo con el modelo adoptado.

Los datos obtenidos y reflejados en la ilustración anterior, sugirieron la existencia de una fuerte correlación lineal entre el tamaño (tanto en LOC como SLOC), el número de clases y la cantidad de métodos. En todos los casos, el coeficiente de Pearson arroja valores entre 0,97 y 0,99. Esto refleja que, en promedio, las distintas versiones tienden a mantener la cantidad de métodos por clase.

La gráfica muestra un pequeño salto en la estructuración de alto nive en la versión 1.4 del software, indicando un incremento de la cantidad de clases por paquete. A partir de la versión siguiente, esta proporción recupera la tendencia de crecimiento lineal que presentaba anteriormente.

Fuera de esta versión puntual, la cantidad de clases por paquetes tienden a crecer de manera lineal respecto del tamaño medido en líneas de código. Este valor atípico podría

sugerir que se produjo un rediseño importante entre esa versión y la siguiente.

Los resultados de estudio llevan a plantear si el comportamiento evolutivo del proyecto analizado son excepcionales o si -por el contrario- son habituales. Cabe destacar que se trata de un proyecto maduro, que produce regularmente nuevas versiones, lo que es poco común en el mundo del F/OSS.

A partir de esta experiencia, aparece como relevante la realización de otros estudios similares, en base a los cuales realizar comparaciones y confirmar o rechazar la generalidad de las tendencias que se observan en este caso.

4.2 Estudio exploratorio

El estudio de caso descrito en el capítulo anterior sirvió como base para una observación más amplia de carácter exploratorio.

El trabajo se presentó con el nombre de “Métricas de Código fuente y Evolución de F/OSS: un estudio exploratorio” ante la Argentine Symposium of Software Engineering ASSE 2013, en el marco de las 42 Jornadas Argentinas de Informática (JAIIO).

El caso precedente muestra que un software en particular (SweetHome 3D) había evolucionado de manera tal que un conjunto de métricas e indicadores seleccionados, extraídos a partir del código fuente, mostraba una tendencia de crecimiento marcadamente lineal a lo largo de las sucesivas versiones de esa aplicación; adeás, el incremento verificado en esos indicadores mostró una pendiente suave. En líneas generales, se puede decir que la evolución de SweetHome 3D se caracteriza por una tendencia a incrementar la complejidad de manera proporcional al crecimiento del tamaño.

Al igual que en el estudio de caso, el trabajo partió del conjunto de métricas propuesto por Lanza y Marinescu[72] para la caracterización del diseño de software orientado a objetos, y utilizó herramientas públicamente accesibles para posibilitar la replicabilidad tanto interna como externa[101].

El nuevo trabajo se planteó las siguientes preguntas:

- si el comportamiento observado en la evolución de SweetHome 3D es frecuente, si es característico en el software libre y de código abierto o si -por el contrario – constituye una excepción.
- si hay métricas o combinaciones de métricas cuyo comportamiento a lo largo de la evolución de este tipo de software muestra una tendencia definida, que podría servir de base para la predicción de las características de futuras versiones.
- si hay métricas o combinaciones de métricas que podrían ser representativas para caracterizar la evolución de F/OSS desde el punto de vista del diseño.

Para ello, se descargaron todas las versiones publicadas hasta diciembre de 2012 de un conjunto de proyectos F/OSS. Sobre cada una se extrajo el conjunto de métricas mencionado en la tabla 4.

En este trabajo se observaron 15 proyectos de F/OSS; los criterios de selección fueron:

- Desarrollados en Java

- Que hubieran producido 7 versiones o más
- Que estuviera disponible el código fuente de todas las versiones a analizaron

El umbral de 7 versiones se decidió tomando en cuenta el número de lanzamientos del proyecto FreePlan, que es un fork del proyecto FreeMind en el que los responsables del proyecto se plantean explícitamente el rediseño con relación al proyecto padre. Tanto FreeMind como FreePlane cuentan con una cantidad importante de usuarios y desarrolladores, que conforman una comunidad activa.

Nombre	Descripción	Versiones
Vuze	Cliente Bittorrent	14
PMD	Analizador de código fuente	13
Hodoku	Generador de Sudoku	12
iText	Librería para gestionar formato PDF	17
Art of Illusions	Modelador gráfico 3D	28
FreeMind	Herramienta para crear mapas mentales	16
FreePlane	Herramienta para crear mapas mentales	7
Chemistry Development Kit	Librería orientada a bioquímica y química	28
Tiled	Editor de mapas para juegos	11
Batik	Librería para gestionar formato SVG	19
jHotDraw	Framework para gráficos estructurados y técnicos	16
Plotdigitizer	Digitalización de información gráfica	11
jPicEdt	Editor de dibujo vectorial	18
JfreeChart	Librería para gráficos	20
Sweethome3D	Aplicación para diseño de interiores	19

Tabla 6: Proyectos seleccionados para el estudio exploratorio

Es importante señalar que el conjunto de proyectos no constituye una muestra aleatoria que habilite la realización de inferencia estadística. Hubo proyectos que fueron descartados por no estar ocasionalmente disponible el código fuente (por mal funcionamiento de la página del proyecto, por ejemplo) o porque se evidenciaran

inconsistencias en la información[1]; este criterio podría acarrear algún tipo de sesgo en la selección, que no fue analizado en mayor profundidad dado el carácter exploratorio del estudio. Por otra parte, en los proyectos Vuze, Hodoku e iText figuraban versiones sobre las cuales no fue posible recuperar el código fuente, lo que puede haber introducido anomalías en los datos correpondientes.

Respecto del trabajo anterior, se incorporó a las planillas de análisis el número de secuencia de la versión (RSN), utilizado frecuentemente en lugar de la fecha de publicación en los estudios de la evolución del software[102]. La razón de esta elección es que el estudio planteado apunta a obtener conocimiento de la evolución de las versiones, atendiendo al posible aumento paulatino de la complejidad.

4.2.1 Valores extraídos

Los datos extraídos en este trabajo figuran en el apéndice correspondiente.

4.2.2 Análisis de los datos

Dado que SweetHome 3D mostraba tendencia al crecimiento lineal en los indicadores observados a lo largo de las 18 versiones, se decidió evaluar esta característica en los diferentes proyectos.

Para ello, se calculó el coeficiente de Pearson entre el tamaño en LOC y los valores obtenidos para los diferentes indicadores. Los resultados se muestran en la tabla 7.

1 Por ejemplo, se descartó el proyecto Batik porque todas las versiones antiguas figuraban con la misma fecha de publicación.

Proyecto	CI	EO	EC	EAN	IAc	DA
jPicEdt	0,2882	-0,3251	0,0553	0,1515	-0,3685	-0,5813
AOI	-0,2810	-0,0330	0,7990	-0,5896	-0,3562	0,8520
PMD	-0,9238	-0,7310	-0,1740	-0,1987	0,6742	-0,4647
iText	-0,9257	0,1119	-0,2774	0,6361	0,6088	-0,8479
FreeMind	-0,8936	0,7618	-0,3405	0,9377	-0,4704	-0,0251
plotdigitizer	-0,8840	0,2062	0,4747	0,9442	-0,1964	0,8159
CDK	0,3643	0,5929	-0,6768	-0,8170	-0,7796	0,5755
Tiled	-0,3437	0,8542	-0,7688	0,9202	-0,2212	0,8908
Hodoku	-0,9932	0,8587	-0,5822	-0,6209	-0,3055	0,7079
Vuze	-0,8789	0,8681	0,6989	0,5737	-0,7052	-0,4220
Batik	0,5244	0,8794	0,9288	0,3755	-0,6558	0,9163
JfreeChart	-0,8054	0,7060	0,9526	0,3216	0,7133	0,9632
FreePlane	-0,9452	0,9141	-0,0141	0,9841	-0,7539	0,8763
jHotDraw	-0,8349	0,8762	0,6348	-0,7537	-0,8174	0,7070
SH3D	0,8331	0,9850	0,9588	0,4503	-0,9079	0,9255
Promedios	-0,4466	0,5017	0,1779	0,2210	-0,3028	0,3926
Desviación	0,6369	0,5291	0,6315	0,6582	0,5490	0,6573

Tabla 7: Coeficientes de correlación de los distintos indicadores de diseño en las diferentes versiones respecto de las líneas de código de cada una

La penúltima fila muestra el promedio de los coeficientes de Pearson obtenidos, en tanto que la última muestra la desviación estándar de esos valores

Se observa que la mayoría de los proyectos no exhibe la tendencia lineal que muestra SweetHome 3D en varios indicadores; únicamente FreePlane exhibe valores que sugieren una relación lineal entre varios indicadores y las líneas de código. Más aún, estas cifras se distribuyen según una amplia dispersión de los datos, según revela la última fila, donde se registra la desviación estándar correspondiente a cada columna.

De los 15 proyectos, 9 exhiben una correlación negativa entre la Complejidad Intrínseca y las Líneas de Código, con valores del coeficiente menores a -0,8, lo que podría ser significativo. Por otra parte, en 7 proyectos encontramos una posible correlación positiva entre la Estructuración de Operaciones (EO) y el tamaño en Líneas de Código, y también 7 exhiben similar comportamiento entre la Dispersión de Acoplamiento y el tamaño. De estos dos grupos, hay cuatro proyectos que muestran coeficientes de correlación superiores 0,8 tanto para la Estructuración de Operaciones como para la Dispersión de Acoplamiento: Sweethome3D, FreePlane, Batik y Tiled. Esto sugiere que para esos proyectos los métodos tienden a crecer linealmente con el tamaño de la versión (en LOC, recordemos) y el acoplamiento entre clases tiende a aumentar su dispersión con el

crecimiento de la aplicación (lo que apoya la afirmación de Lehman sobre el incremento paulatino de la complejidad[92].

El análisis precedente sugiere que estos indicadores podrían caracterizar la evolución de los proyectos de F/OSS; un análisis más masivo brindaría elementos para confirmar o rechazar si es posible agrupar o tipificar la evolución del diseño de aplicaciones F/OSS a partir de los valores de estas métricas derivadas.

Estos resultados llevan a analizar específicamente las variaciones de los valores de la Estructuración de Operaciones y la Dispersión de Acoplamiento respecto del tamaño, dado que tendencias crecientes en esas proporciones podrían indicar creciente complejización del software; en tal caso, el conocimiento de esta tendencia debería tomarse en cuenta al momento de seleccionar software para la reutilización.

Dado que la muestra sigue siendo pequeña, se plantea a continuación el estudio de las distribuciones de frecuencia de las diferentes métricas y proporciones descritas en las tablas 7.

A partir de ese análisis buscaremos establecer umbrales de referencia que operen como advertencias de potenciales debilidades de diseño cuando se observen valores por fuera de esos umbrales.

En el siguiente capítulo detallamos el estudio realizado sobre un conjunto de datos obtenidos de la evolución de 28 proyectos de F/OSS, conformando una muestra mucho mayor.

4.3 Determinación de umbrales

Fowler señaló[103] que ninguna combinación de métricas puede rivalizar con la intuición humana informada; por eso, no se busca postular relaciones causales sino indicadores que actúen como advertencia de posibles debilidades en el diseño. Lanza y Marinescu[72] señalaron que no existe un umbral perfecto, cuya superación signifique necesariamente un cambio cualitativo en el atributo que se pretende evaluar, pero pueden ser indicativos de utilidad en la práctica.

Es difícil establecer la vinculación entre las métricas y los distintos atributos del software, tales como la mantenibilidad, la evolutividad, la complejidad, etc.; entre otros factores se ha señalado que:

- Los atributos que se quiere reflejar con las métricas dependen de muchos factores. Esos factores pueden variar en su incidencia relativa[56] [104].
- Las métricas que intentan medir los factores no siempre están debidamente validadas o no cuentan con el consenso suficiente en la comunidad científica[65].
- Cuando se intenta relacionar los valores medidos con atributos como la mantenibilidad, evolutividad, etc., se apela al juicio de expertos a falta de medidas que reflejen esos atributos.
- Los juicios de expertos pueden ser contradictorios o al menos no ser completamente coincidentes.

En este trabajo nos proponemos establecer valores de referencia a partir de la distribución de frecuencias de un conjunto de métricas de diseño orientado a objetos en un conjunto de aplicaciones que han evolucionado de manera efectiva, publicando al menos 7 versiones.

En la sección 2.5.3 se presenta una revisión sobre estudios similares. Los datos relevados se encuentran en el Anexo II.

4.3.1 Características de la muestra analizada

Se computaron las métricas de 560 versiones de 28 proyectos diferentes. Las aplicaciones seleccionadas se caracterizan por haber publicado entre 7 y 48 versiones. En todos los casos la edad del proyecto supera los 3 años.

En total, se computaron 371.934 clases que incluyen 3.907.564 métodos, cubriendo 43.842.838 líneas de código.

El detalle de las cifras obtenidas se presentan en el Anexo II.

4.3.2 Análisis de los datos

A partir de cada una de las métricas obtenidas, se obtuvieron las proporciones descritas en la tabla 8.

La siguiente tabla resume las medidas de localización y dispersión[105] para las proporciones estudiadas:

	1º Cuartil	Media	Mediana	3º Cuartil	Desviación estándar	Coef. de Variación	Coef. de asimetría
Complejidad Intrínseca	0,1727	0,2076	0,2053	9,2236	0,041	0,13	0,87
Estructuración de Operaciones	7,973	10,27	10,86	14,310	4,277	0,41	0,89
Estructuración de Clases	9,581	10,91	11,74	13,33	3,453	0,29	0,93
Estructuración de Alto Nivel	7,503	9,98	12,55	17,81	6,965	0,7	1,41
Intensidad de Acoplamiento	0,556	0,588	0,585	0.616	0,168	0,29	17,63
Dispersión de acoplamiento	2,534	2,992	3,024	3,399	0,82	0,27	0,13

Tabla 8: medidas de localización y dispersión observadas en las proporciones de métricas bajo estudio

Las distribuciones de estas proporciones muestran que la Complejidad Intrínseca, la Estructuración de Clases, la Intensidad de Acoplamiento y la Dispersión de acoplamiento exhiben menor dispersión. De ellas, la Intensidad de acoplamiento presenta una

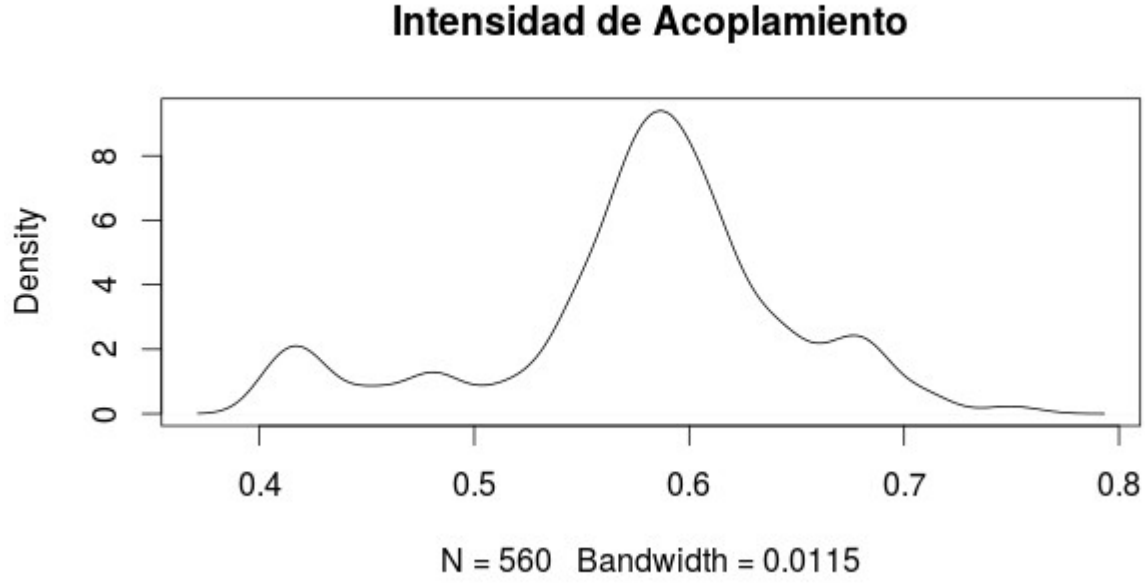

Ilustración 4: Curva de distribución de la Intensidad de Acoplamiento

distribución de frecuencias con una asimetría más marcada. Analizaremos con más detalle estas cuatro proporciones en busca de valores que sirvan como umbrales de referencia- Para ello analizamos las gráficas de distribución que presentamos más abajo; posteriormente, consideraremos la variación entre deciles, a fin de observar en cuál de ellos se constata mayor distancia entre los valores.

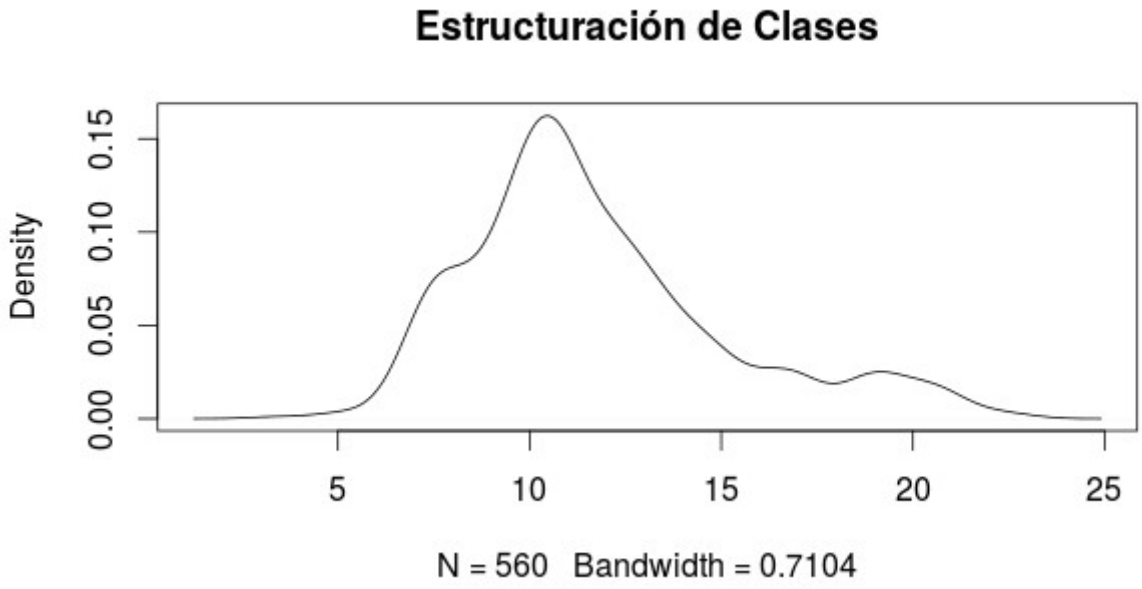

Ilustración 5: Distribución de la Estructuración de Clases

Respecto de la Estructuración de clases, la gráfica muestra una alta concentración en torno a la moda. En tanto, la Estructuración de Operaciones exhibe varios picos, con una caída pronunciada en las frecuencias de valores mayores a 20.

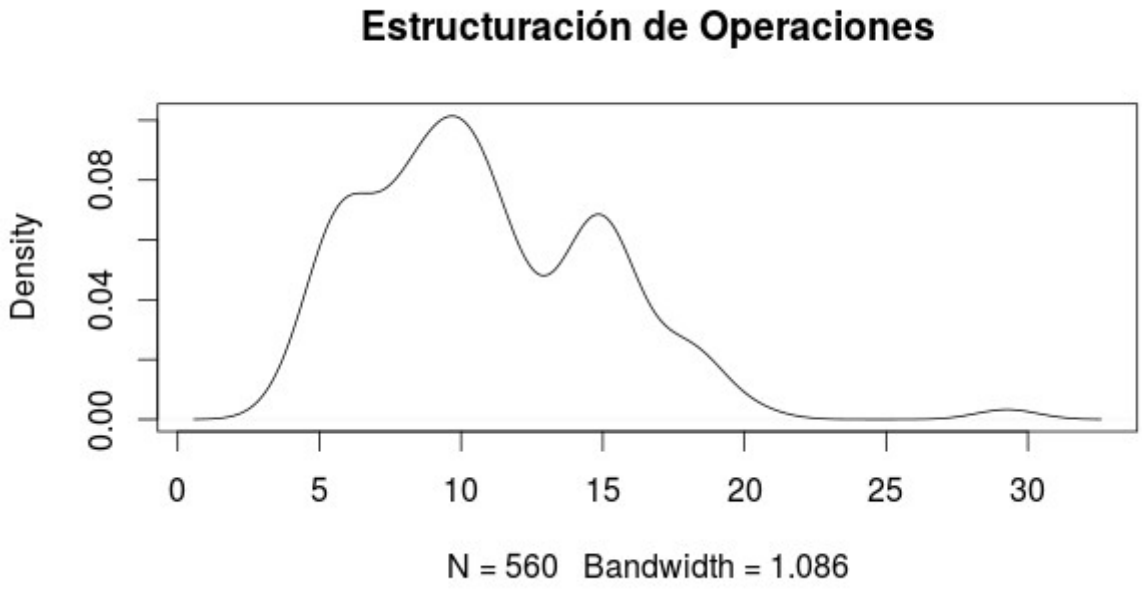

Ilustración 6: Distribución de la Estructuración de Operaciones

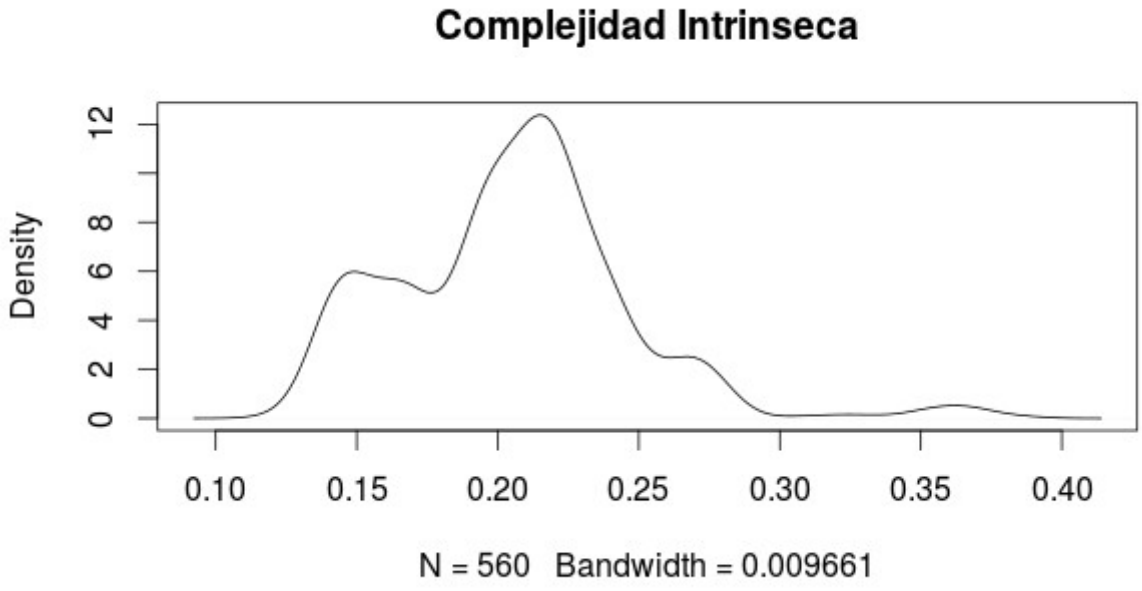

Ilustración 7: Distribución de la Complejidad Intrínseca

Se observa que el 80% de las versiones tienen menos de 14,09 métodos por clase. El

paso entre el octavo y noveno decil es el mayor, lo que sugiere adoptar como referencia ese valor como umbral superior.

Las distribuciones de frecuencias de la Estructuración de Clases y de la Intensidad de Acoplamiento muestran claramente una zona modal a partir de las cuales las frecuencias decrecen en ambos sentidos. Analizando la variación de los deciles podemos proponer umbrales superiores buscando cambios pronunciados entre los valores sucesivos de los mismos, lo que implicaría un descenso rápido de las frecuencias. La siguiente tabla resume los deciles para las distribuciones de las métricas consideradas.

Observemos también la distribución de WMC, calculada a partir de otras métricas según se señaló más arriba.

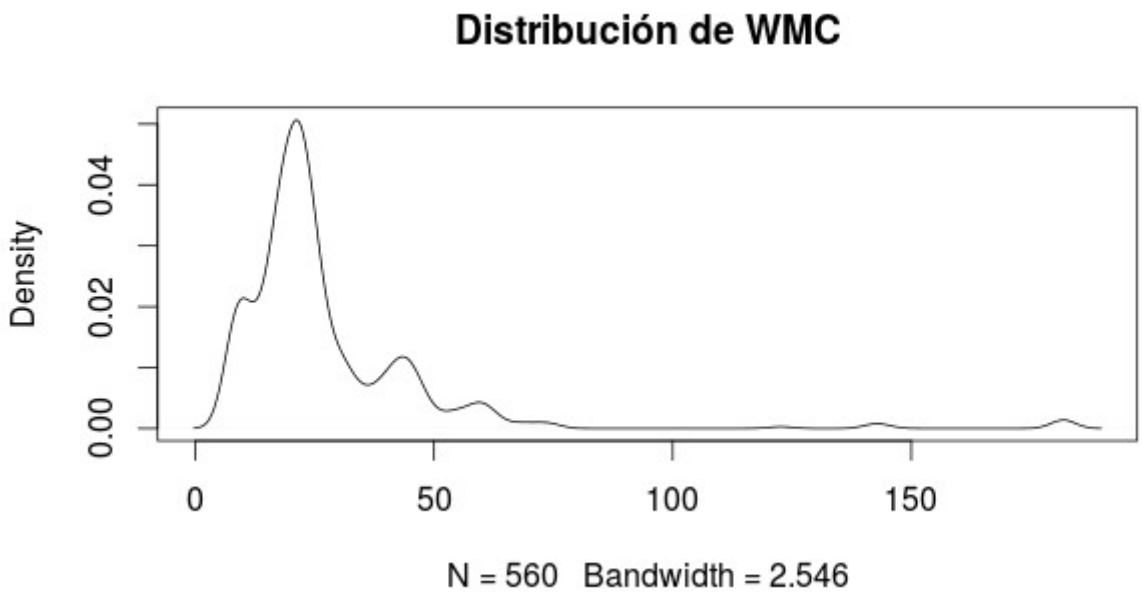

Ilustración 8: Distribución de WMC calculada en base a C.I., E.O. y E.C.

Decil	*C.I.*	*E.C.*	*E.O.*	*D.A.*	*I.A.*	*WMC*
"0%"	0,121	3,375	3,832	0,234	0,406	7,346
"10%"	0,148	7,721	5,707	1,968	0,468	9,774
"20%"	0,166	9,001	6,876	2,372	0,544	16,233
"30%"	0,191	9,950	8,218	2,603	0,567	18,103
"40%"	0,196	10,458	9,425	2,750	0,575	20,463
"50%"	0,208	10,905	10,272	2,992	0,588	22,000
"60%"	0,216	11,730	11,301	3,172	0,594	23,772
"70%"	0,221	12,639	13,622	3,337	0,607	26,589
"80%"	0,233	14,092	14,820	3,665	0,625	36,062
"90%"	0,251	16,936	15,830	4,270	0,668	45,683
"100%"	0,385	22,773	29,321	5,084	0,759	182,078

Tabla 9: Valores de los distintos deciles de la distribución de las proporciones consideradas

Respecto de la estructuración de clases, se observa que entre el tercer y el octavo decil (60% de los casos) tienen en promedio entre 9 y 14,092 métodos por clase. Por otra parte, analizando el histograma (sesgado hacia la izquierda) se observa que la frecuencia de los valores superiores a 13 decrece significativamente, lo que convierte a ese valor en un posible umbral.

En cuanto a la Intensidad de acoplamiento, el decrecimiento más pronunciado recién comienza con el noveno decil , por lo que un promedio de este valor superior a 0,668 advertiría sobre un posible acoplamiento excesivo.

Si contemplamos WMC, vemos que la curva de distribución de frecuencias tiene un máximo posterior a la moda; recién luego del 9º decil se verifica un descenso constante, por lo que el umbral para esta métrica sería de 45,68

4.3.2 Comparación con umbrales propuestos en otros trabajos

La siguiente tabla muestra los valores propuestos en diversos trabajos. No se incluyen los de Benlarbi y otros [74] dado que plantean valores específicos para distintas aplicaciones y no de carácter general.

Métricas	Shatnawi[76]	Rosenberg[69]	Lanza y Marinescu[72]	Herbold et al[106]
CBO	9	5	N/A	5
WMC	20	100	47	100

Tabla 10: Umbrales propuestos en otros trabajos

Como puede observarse, las cifras postuladas son muy disímiles. Cabe señalar, no obstante, que el valor planteado por Lanza y Marinescu para WMC es similar al obtenido en este trabajo sobre una muestra mucho mayor.

4.3.3 Observaciones sobre los resultados

El análisis de las distribuciones de frecuencia del conjunto de proporciones descritos en la Tabla 5, las métricas que surgen como potencialmente útiles son -al igual que en el estudio exploratorio- la Estructuración de Clases y la Intensidad de Acoplamiento. La primera de ellas tiene la ventaja de ser fácil de medir, existiendo numerosas herramientas que la computan.

Si se incluye en el análisis la Complejidad Ciclomática de los métodos, vemos que la distribución de la métrica WMC también puede servir como advertencia cuando supere el valor 47, si tomamos el mayor valor entre los resultados observados aquí y los encontrados por Lanza y Marinescu[72].

Capítulo 5: Marco para la Reutilización de F/OSS

5.1 Consideraciones Previas

Según las palabras de Frederick Brooks, “the most radical possible solution for constructing software is not to construct it at all”[1]. En ese sentido, los mayores beneficios de la reutilización se alcanzarían mediante la obtención de artefactos de software (documentos, requerimientos, diseños, código) desarrollados previamente u ofrecidos por terceros, de modo de integrarlos en un nuevo proceso de desarrollo.

Uno de los obstáculos para la adopción de la reutilización en los procesos de desarrollo es la dificultad de encontrar artefactos reutilizables que se ajusten a los requerimientos que demanda la reutilización o en los que pueda evaluarse con antelación -y con razonable precisión- el esfuerzo que demandaría la integración posterior[6]. Este problema atraviesa los diversos abordajes sobre el tema.

En esta sección proponemos un marco de trabajo para la selección y evaluación de software libre y de código abierto. El marco propuesto es de carácter general, pudiendo adecuarse a diferentes estrategias y técnicas de reutilización[18] [107]. Se utilizó este marco como referencia para el desarrollo de una herramienta sencilla, de modo de explorar la viabilidad de la utilización del mismo.

La formulación que presentamos aquí no pretende ser definitiva; será necesario adecuarla y ponerla a prueba en diferentes contextos de desarrollo, para validar sus posibilidades de manera más general. No obstante, se plantea como una primera aproximación a la necesidad de disponer de pautas regulares para el proceso de selección y evaluación de F/OSS para el reuso.

Para la elaboración del marco comenzamos desarrollando de manera experimental una aplicación educativa en la que las funciones más complejas se cubran mediante recursos de F/OSS. Buscamos así realizar una experiencia que nos permita conocer mejor los aspectos del proceso de reutilización, de modo de extraer orientaciones para futuros trabajos del mismo tipo. El esquema del proceso seguido para la elaboración se presenta en la siguiente ilustración.

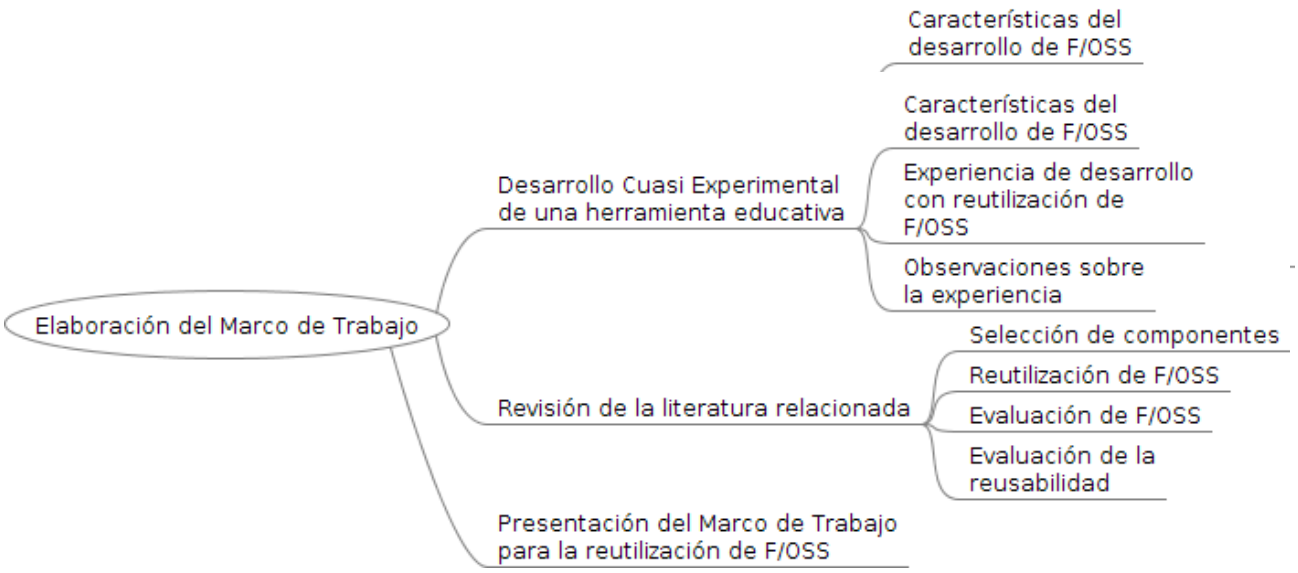

Ilustración 9: Esquema de la elaboración del marco de trabajo

Posteriormente, repasamos la literatura relacionada con la reutilización y el F/OSS.

5.1.1 Acercamiento al problema: desarrollo de una aplicación con reutilización

A fin de experimentar sobre la viabilidad y las dificultades de desarrollar nuevas aplicaciones reutilizando F/OSS, nos planteamos la construcción de una herramienta educativa libre. La aplicación en cuestión se orienta a atender necesidades particulares de un grupo de docentes universitarios; específicamente, se trata de una aplicación para crear mapas conceptuales sencillos y que a la vez se pueda utilizar como soporte de una presentación mediante una interfaz de zooming[10]. La selección de este desarrollo para llevar adelante el acercamiento a la problemática que nos ocupa no se realizó de manera aleatoria, pudiéndose considerar de carácter cuasi experimental[108].

Para ello, comenzamos relevando trabajos referidos a las metodologías de desarrollo de F/OSS. Posteriormente, se realizó el desarrollo de la herramienta.

5.1.1.1 La investigación sobre el desarrollo de F/OSS:

Para orientar el desarrollo, repasamos una serie de investigaciones relevantes referidas a los procesos de desarrollo propios de software libre y las características o condiciones de ese desarrollo tendientes a obtener un producto comunitario exitoso.

Una referencia fundamental sobre este tema es el ensayo de Eric Raymond “La Catedral y el Bazar”[109], donde esboza una serie de características que serán propias del desarrollo de F/OSS, entre las que destaca:

- Un buen proyecto de software parte de la necesidad del desarrollador (o, en sus términos más coloquiales, de “rascarse una comezón”)
- Publicar rápida y frecuentemente. La publicación rápida apunta a ampliar la base de interesados en el proyecto; la frecuencia busca la retroalimentación, de manera similar a las propuestas de las metodologías ágiles.
- Considerar a los usuarios como “co-desarrolladores”, encargados de probar las aplicaciones e informar fallos y hacer propuestas para futuros desarrollos.

El ensayo de Raymond se basa en su experiencia personal, no en un trabajo sistemático de investigación. Estudios empíricos posteriores relativizaron la centralidad de los postulados de ese autor; por ejemplo, Krishnamurty [110] observaba en 2002 que la enorme mayoría de los proyectos maduros de software libre se desarrollan a partir de muy pocas personas, contrariamente a lo que se esperaría del modelo de bazar mencionado más arriba. Otros estudios[111] [112] concluyen que los proyectos exitosos de Software Libre tienden a seguir un ciclo de vida en el que pasan por fases centralizadas (o de “catedral”) y comunitarias (o de “bazar”).

Senyard y Michlmayr[112] consideran que un proyecto exitoso *debe pasar de la fase de catedral a la de bazar* a partir de la publicación del código fuente y la incorporación de usuarios y desarrolladores al proyecto. Entre los factores de éxito propuestos por estos autores se destacan:

- Un prototipo funcional promisorio, que permita a los usuarios ejecutarlo y darse una idea de las funciones buscadas
- Diseño modular en el proyecto inicial, para permitir el trabajo simultáneo de los diferentes desarrolladores, mejorar la mantenibilidad y la evolutividad
- Mecanismos adecuados para la comunicación y la contribución por parte de usuarios y desarrolladores
- Estilos o estándares de codificación claramente definidos
- Ciclos de publicación breves que liberen versiones funcionales y estables (siguiendo la idea de Raymond de publicar rápida y frecuentemente)
- Documentación clara y apropiada para desarrolladores y usuarios
- Una licencia de distribución atractiva, tanto para desarrolladores como para usuarios

Fogel[113] también presenta una serie de ideas para desarrollar software libre exitoso,

enfatizando los errores comunes en este tipo de proyectos. Este autor coincide en destacar la importancia de que el desarrollo se relacione con las necesidades e intereses de los iniciadores del proyecto y de la incorporación de desarrolladores y usuarios.

5.1.1.2 Experiencia de desarrollo

Para comenzar con el desarrollo de la herramienta, tomamos en cuenta las consideraciones del punto anterior. Los requerimientos iniciales se establecieron de manera general, siguiendo el modelo de "lista de deseos"[113] [114], con la perspectiva de incorporar y refinarlos en la medida en que el proyecto crezca. La lista inicial de características del software surge de la función educativa que se pretende atender:

- Multiplataforma, para que todos los integrantes de la comunidad educativa puedan utilizarlo
- Creación visual de mapas conceptuales
- Zooming sobre cada nodo
- Almacenamiento en XML, por su carácter estándar

El proyecto SiZoop (Simple Zooming Presentations) fue registrado en SourceForge (el mayor repositorio de software libre y de código abierto) el 18 de enero de 2011.

Elegimos el lenguaje Java por su carácter multiplataforma y por contar dentro de la Universidad Nacional de Salta con potenciales desarrolladores capacitados en la programación con ese lenguaje.

Reutilización de software: buscamos componentes que pudieran resolver las funciones de Zooming y el almacenamiento en XML, las que a priori presentaban mayores dificultades para plantearse un desarrollo desde cero.

En primer lugar buscamos frameworks libres en java que brindaran la posibilidad de crear ZUI. La búsqueda se basó en los repositorios java-source.net y sourceforge; al mismo tiempo, se utilizó un buscador Web para relevar otras aplicaciones que pudieran no estar incluidas en los repositorios mencionados.

Analizamos dos alternativas: Prefuse Information Visualization Toolkit y Piccolo 2D. El primero de ellos está orientado a ofrecer distintas formas de visualizar información, en tanto que el segundo es propiamente un framework. Se descartó el primero fundamentalmente por el tiempo que lleva sin actividad de desarrollo: la última publicación data de octubre de 2007 (http://prefuse.org).

Piccolo2D, en cambio, es específicamente un framework para desarrollar aplicaciones con

ZUI. La última versión lanzada al momento del desarrollo era del 14 de abril de 2011. Otro aspecto destacable es la cantidad de aplicaciones que utilizan este framework (http://piccolo2d.org).

Para la tarea de almacenar en formato XML, analizamos JiBX y XOM. El primero es más flexible, pero requiere de un paso intermedio para incorporar el contenido XML a clases java. Por lo tanto, se adoptó la segunda.

Licencia: SiZoop se distribuye bajo una licencia dual (GPL y BSD) para brindar opciones diferentes a potenciales desarrolladores, al mismo tiempo atender a las diferencias entre las licencias de los componentes reutilizados (LGPL para XOM, BSD para Piccolo2D).

Documentación: dado que el proyecto aún es de tamaño pequeño (menos de 2K líneas de código nuevas), la única documentación existente del desarrollo es el JavaDoc generado durante la codificación.

5.1.1.3 Observaciones sobre la experiencia de desarrollo

De esta experiencia se desprenden algunos aspectos que deben tenerse en cuenta al momento de realizar un desarrollo que reutilice F/OSS:

- Es preciso definir las funciones que se espera que se obtengan de recursos F/OSS a reutilizar.
- Es conveniente determinar en una etapa temprana la licencia bajo la cual se distribuirá el nuevo desarrollo, de modo de cotejar la compatibilidad de la misma con las potenciales piezas de software a incorporar.
- Para una lista de funciones, es posible encontrar varias alternativas F/OSS que entre las cuales se deberá elegir
- La selección de los productos F/OSS estuvo influida en gran medida por la disponibilidad de documentación y la continuidad del desarrollo de las mismas.
- La reutilización permitió desarrollar rápidamente un prototipo funcional que facilitó mostrar las ideas y objetivos del desarrollo.
- Otro aspecto que incidió en la selección fue la presencia de versiones más recientes del producto elegido; esta característica asegura la continuidad del desarrollo a cargo de un equipo o de una comunidad independiente de nuestro propio proyecto. Por otro lado, los problemas que se pudiera encontrar en el software aún pueden ser atendidos por la propia dinámica de mantenimiento del proyecto en cuestión.

5.2 Revisión de trabajo relacionados

El marco que proponemos se plantea los siguientes objetivos generales:

- Proveer una guía para la reutilización de F/OSS en nuevos desarrollos
- Brindar una referencia para sistematizar las prácticas de reutilización de F/OSS

Para elaborar el marco tomamos en cuenta las estrategias y técnicas para la reutilización, presentadas en el capítulo 2 y las observaciones de la experiencia descrita en la sección anterior.

En este apartado presentamos una revisión de los enfoques para la selección de activos reutilizables, incluyendo propuestas de marcos de trabajo para la reutilización. A continuación; luego repasaremos diversas publicaciones sobre la reutilización en F/OSS y revisaremos propuestas de evaluación y selección específicamente para productos de de este tipo.

5.2.1 Selección de componentes para la reutilización

Se han propuesto diversas metodologías y marcos de trabajo para guiar el proceso de selección de activos reutilizables; la mayoría de estas propuestas apuntan a la reutilización de COTS (Components-off-the-shelf), aunque algunos contemplan la recuperación de código fuente para incorporar en nuevos proyectos.

Cechich y Piattini [115] presentaron una revisión de un conjunto de métodos basados en medición para la selección de COTS. Las propuestas analizadas fueron OTSO (Off-the-shelf-Option), CAP (Component Acquisition Process), PORE (Procurement-oriented Requirement Engineering) y CEP (Component Evaluation Process).

Los autores señalan que CAP se presenta como una mejora de OTSO, más simple y flexible. Ambos parten de la identificación de los criterios en base a los cuales se realizará la selección y recurren al Método Analítico Jerárquico para guiar la toma de decisiones. Estos modelos contemplan los aspectos económicos en el proceso de selección. CEP, en tanto, permite la adecuación al proceso de desarrollo concreto y parte de una base de datos en donde se realiza inicialmente la búsqueda. En cuanto a PORE, resaltan que el método aborda la ingeniería de requerimientos y la adquisición de componentes de manera simultánea; al respecto, señalan que el proceso iterativo de elicitación y evaluación/selección de componentes puede ser demasiado complejo.

Cechich y Piattini también advierten que este conjunto de métodos basan la selección en características genéricas de los componentes; al respecto, plantean la necesidad de contextualizar la selección orientándola al dominio para alcanzar resultados más seguros.

Un estudio comparativo[116] realizado por Taryano y Cechich sobre el empleo de OTSO y CEP aplicado al desarrollo de sistemas de pago electrónico, pone de manifiesto que estos métodos no proveen de una medida de comparación entre las posibles opciones.

De Almeida y otros[117] repasan la literatura relacionada con los factores de éxito y fracaso para la reutilización. A partir de esos estudios, los autores proponen un marco de trabajo amplio, organizado en dos capas; la primera contempla “buenas prácticas” que posibilitan la incorporación de la reutilización en una organización, en tanto que la segunda se centra en aspectos técnicos como los procesos, el entorno y las herramientas.

Bhuta y otros[26] proponen un marco de trabajo para la selección y evaluación de componentes y conectores en arquitecturas basadas en componentes. Este marco se orienta fundamentalmente a la evaluación de la interoperabilidad, de modo de disponer de elementos de juicio respecto del esfuerzo que implicaría lograr el trabajo conjunto de componentes diferentes; apunta específicamente a informar sobre la necesidad de conectores o de código de unión (glue code).

Bart y otros [27] aportan un enfoque diferente para la búsqueda y selección de componentes, mediante la construcción de los requisitos de un componente “ideal” o modelo, evaluando la afinidad de los componentes candidatos con ese modelo. El autor observa que los métodos habituales suelen basarse en otorgar un puntaje a los activos candidatos (por ejemplo, mediante el Método Analítico Jerárquico), pero los procedimientos no están automatizados, lo que redunda en un esfuerzo significativo para llevar adelante la tarea.

Los autores también señalan que la incorporación de un componente determinado puede alterar los requerimientos, lo que llevaría a repetir el proceso de selección bajo nuevas condiciones.

Los trabajos referidos brevemente aquí muestran que los diferentes métodos de selección presentan, en general, una sucesión de fases o etapas; en todos los casos, se incorpora el aspecto contextual de la evaluación ya sea mediante el establecimiento de pesos relativos para los indicadores a evaluar o en la asignación manual de categorías.

5.2.2 Reutilización de F/OSS

La literatura en Ingeniería de Software ha señalado muchas veces las ventajas potenciales de la reutilización en el proceso de desarrollo, especialmente bajo enfoques sistemáticos, planificados y basados en la disposición de un mercado de componentes amplio y competitivo; sin embargo, la enorme oferta de activos F/OSS no se ajusta a las necesidades de un abordaje sistemático de ese tipo, ya que no se cuenta con un sistema

de búsqueda y recuperación orientado a la reutilización[12], ni con descripciones estándares de la funcionalidad ni de mecanismos de certificación, entre otros aspectos sustanciales. En cambio, ofrece abundante software libremente disponible, herramientas de búsqueda de carácter general y un cuerpo creciente de investigaciones sobre la reutilización de este tipo de productos [118] .

Ayala y otros[118] estudiaron las dificultades de la reutilización de F/OSS en la industria, a partir del mercado de componentes de este tipo de software. Para ello entrevistaron a personas involucradas en el desarrollo basado en componentes en empresas pequeñas y medianas de España y Noruega.

De las entrevistas surge la importancia de la experiencia previa de los desarrolladores en la utilización de los componentes que eventualmente adopten.

Manuel Sojer[42], en su tesis de magister, realizó un amplio relevamiento sobre la reutilización de F/OSS, centrándose en los factores que los propios protagonistas destacan respecto de las posibilidades, las ventajas y las limitaciones que afectan este tipo de reutilización. Los principales beneficios que los desarrolladores valoran es la ayuda para alcanzar las funcionalidades esperadas, la posibilidad de concentrarse en aspectos más interesantes y la posibilidad de resolver problemas para los cuales no cuentan con la experiencia suficiente.

Aparte de estos enfoques generales, existen varios trabajos que investigan empíricamente la reutilización de F/OSS en casos concretos.

Haefliger y otros[119] estudiaron la reutilización de código en una muestra de proyectos F/OSS que muestran alta actividad de desarrollo, priorizando los que exhiben mayor número de descargas por parte de los usuarios. Por una parte, emplearon herramientas especiales para detectar la duplicación de código y detectar casos de reutilización de caja blanca (incluyendo aquí la de “caja de vidrio”) y de caja negra; por otro lado, entrevistaron a desarrolladores mediante la intervención en las listas de correo. Tres observaciones importantes que surgen de este estudio es que los criterios de selección de activos reutilizables (componentes o código) son las funcionalidades que provee, y la evaluación del código fuente y de la documentación que proveen; el segundo aspecto a destacar es que la principal fuente de información sobre posibles componentes es la propia comunidad de desarrolladores; y en tercer lugar, valoran la “popularidad” de los recursos potencialmente reutilizables, asumiendo que el uso extendido favorece a la frecuencia de la corrección de errores.

Heinemann [120] y otros analizaron una muestra de proyectos F/OSS escritos en Java. Los autores observaron que en esos proyectos el 90% utiliza software desarrollado por

terceros, siendo más habitual la reutilización de caja negra. Algunas de las aplicaciones analizadas muestran tasas de reutilización (es decir, la proporción de código reutilizado respecto del total) cercana al 0.9.

Capiluppi y Stol[121] analizaron la reutilización dentro del proyecto FFMpeg, donde observaron la evolución de “componentes a nivel de construcción”, entendidos como jerarquías de directorios que incluyen código y otros archivos que se utilizan en el momento de compilación o construcción.

El estudio muestra que varios componentes -según la definición del párrafo anterior- fueron reutilizados en otros proyectos, tanto en estrategias de caja blanca (incluyendo una copia propia de los componentes) como de caja negra (mediante enlaces dinámicos al componente).

La literatura que informa sobre casos de reutilización de F/OSS es amplia; mencionamos aquí a modo de ejemplos el desarrollo de un editor que incorpora funciones de SCORM a partir del software FCKEditor[122], el desarrollo de herramientas educativas que incorporan partes libres[123] [10], y el caso de la empresa israelí Orbotech que incluye la reutilización de F/OSS y de COTS[124]; en este último caso, la empresa dispone de personal específico para el seguimiento y la adaptación de F/OSS, por lo que su reutilización supone importantes ahorros en sucesivos desarrollos.

Los trabajos resumidos ponen de manifiesto respecto de la reutilización que:

- Es frecuente en el desarrollo de F/OSS
- Un vector habitual para adoptarla es la factibilidad de alcanzar desarrollos rápidos que ofrezcan funcionalidades básicas, lo que resulta relevante para la creación de prototipos como para publicar una “promesa creíble” en las primeras etapas de un proyecto F/OSS
- Su adopción se facilita con la experiencia previa y el conocimiento de proyectos F/OSS

5.2.3 Evaluación de F/OSS

La difusión del F/OSS y su adopción dentro de diferentes organizaciones llevó a proponer diversas metodologías para evaluar los productos de ese tipo. Entre las que alcanzaron mayor difusión se pueden mencionar los Modelos de Madurez elaborados por las compañías CapGemini y Navica, QSOS[125], SQO-OSS[126], OS BRR[127] y QualiPSO[128].

Estas propuestas apuntan en general a calificar la calidad de los productos, aunque no

necesariamente contemplan la perspectiva de la reutilización.

A continuación repasamos algunos de los modelos sugeridos y análisis de los mismos.

5.2.3.1 QSOS

La empresa Atos Origin elaboró el método denominado Qualification and Selection of Open Source software (QSOS).

QSOS ofrece en su sitio Web el detalle del método en un documento[125] distribuido bajo una licencia libre. Se han publicado las versiones 1.6 y 2.0. En el sitio también se ofrecen herramientas para realizar evaluaciones.

Los objetivos declarados de la metodología son:

- Calificar una pieza de software integrando las especificidades del software libre y de código abierto
- Comparar diferentes piezas de software dependiendo de las necesidades y ponderando criterios para tomar una decisión final.

La decisión de distribuir los documentos y las herramientas bajo licencias libres se orienta a permitir la reutilización de las evaluaciones, así como la calidad y objetividad de la documentación posibilitando la transparencia y la revisión por pares.

La metodología consta de 4 pasos que se siguen de forma iterativa:

- Definición
- Evaluación
- Calificación
- Selección

La etapa de definición busca establecer tipologías que se tendrán en consideración en etapas posteriores. Se toman en consideración las funcionalidades del software en relación a las necesidades de la organización que evalúa, las características de la licencia bajo las cuales se distribuye y la madurez del proyecto. Esta última se califica en base a una plantilla propuesta en la metodología, donde se valora la antigüedad del proyecto que desarrolla la aplicación evaluada y las características de los desarrolladores.

El resultado de esta etapa es el de establecer criterios que guiarán las etapas subsiguientes.

La etapa de evaluación comprende la recuperación de la información necesaria para

juzgar el software en función de los criterios establecidos en el paso anterior. Para ello se emplean plantillas donde se completan las calificaciones respecto de cada criterio.

La tercera etapa se centra en ajustar las evaluaciones en función del contexto específico en el que se realiza, estableciendo pesos para las distintos factores en función de los requerimientos y necesidades reales.

La última etapa consiste en la selección de la pieza o su comparación con otras. Esta selección puede ser estricta o laxa, según se determine como imperativo el cumplimiento de determinados valores para la aceptabilidad.

5.2.3.2 OpenBRR

Bussiness Readyness Rating otorga una clasificación de 5 valores posibles que van de “inaceptable” a “excelente”.

Utiliza 4 fases:

- Filtro de evaluación rápida
- Evaluación del Destino de Uso
- Recolección de datos y procesamiento
- Traducción de datos.

Deprez y Alexandre[129] comparan analíticamente las metodologías QSOS y OpenBRR.

Los autores observan que ambas metodologías parten de un conjunto predefinido de criterios. Éstos se clasifican en 3 categorías con 2 niveles jerárquicos en OpenBRR y 3 niveles en QSOS.

En ambos casos, la evaluación consiste en asignar un valor para cada criterio.

5.2.3.3 SQO-SSO

Samoladas elaboró una metodología orientada a la evaluación automática de F/OSS. Este método no toma en cuenta la funcionalidad; se centra -en cambio- en la calificación del código fuente, al que considera el producto más importante de un proyecto de desarrollo de software.

SQO-SSO también toma en cuenta la comunidad que da soporte a un proyecto, pero sólo en cuanto a las características que pueden ser medidas de manera automatizada.

Samoladas considera que OpenBRR es altamente subjetivo y señala que la especificación de una aplicación de referencia ha sido criticada en diversos trabajos.

En relación con QSOS considera que el modelo no es suficientemente flexible y resulta difícil de utilizar.

A partir de estas observaciones, se propone otra metodología con los siguientes lineamientos:

- Busca automatizar del proceso de evaluación.
- Conforma el núcleo de un sistema de monitoreo de calidad y recolección automática de métricas.
- No contempla la evaluación de la funcionalidades. Se concentra en la mantenibilidad, confiabilidad y seguridad del código.
- Se centra en el código fuente, al considera el producto simple más importante en un proyecto de desarrollo de software y cuya calidad puede jugar un rol determinante en la evaluación final de un producto.
- Toma en cuenta la comunidad de desarrollo de un proyecto, pero sólo los aspectos susceptibles de ser medidos automáticamente.
- Permite modificar los perfiles de las categorías utilizadas a fin de adaptarse al punto de vista del evaluador.

El método asume que el proceso de evaluación contempla dos fases:

- Definición del modelo de evaluación
- Definición del proceso de medición

En SQO-OSS, cada una de esas fases consta de dos pasos:

Primera fase:

- Definición de los criterios del modelo (atributos y sub-atributos)
- Definición de las métricas

Segunda Fase:

- Definición de las categorías de evaluación
- Definición de los perfiles de esas categorías

Este modelo parte de dos supuestos: que la salud y la calidad de un software libre depende de la calidad del código fuente y de la comunidad que los sostiene.

La definición de las métricas parte del modelo GQM (Goal Question Metrics) planteado

por Basili[130].

El modelo de SQO-OSS busca evaluar la mantenibilidad, la confiabilidad y la seguridad.

Basándose en el estándar ISO/IEC 9126, desglosa la mantenibilidad en:

- Analizabilidad
- Mutabilidad
- Estabilidad
- Capacidad de prueba

La comunidad en torno a un proyecto considera las listas de correo, la documentación y la base de desarrolladores.

5.2.3.4 Resumen de los métodos de evaluación de F/OSS

La siguiente tabla resume para cada una de las metodologías presentadas, las etapas, los atributos, la metodología de calificación y la escala en la que se brinda el resultado.

Este resumen sirve como referencia para definir una evaluación exhaustiva de los candidatos a la reutilización, teniendo en consideración que ninguno de ellos se orienta específicamente a esa función.

Método	Etapas	Atributos	Calificación	Resultado
QSOS[125]	Definición, evaluación, calificación, selección	Funcionalidades, madurez, licencias	Asignación manual de puntajes	Entero positivo
OpenBRR[131]	Evaluación rápida, evaluación del destino de uso, recolección de datos, traducción	Funcionalidad, aspectos operacionales del software, soporte y servicio, documentación, atributos de la tecnología de software, adopción y proceso de desarrollo	Asignación manual de puntajes	Escala en cinco categorías
SQO-SSO[126]	Definición del modelo de evaluación (criterios y métricas), definición del proceso de medición (categorías y perfiles)	Mantenibilidad, confiabilidad,seguridad (según ISO/IEC 9126)	Puntuación automatizada	Cuatro categorías
OMM[132]	Evaluación de madurez, asignación de pesos, cálculo de la madurez general	Características del producto de software, documentación, soporte, entrenamiento, integraciones de producto, y servicios profesionales	Asignación manual de puntaje	Valor entre 0 y 100

Tabla 11: Metodologías de selección de F/OSS

5.2.4 Evaluación de la reusabilidad

En el capítulo 2 repasamos trabajos relacionados con la modificación del software. La reutilización puede involucrar la modificación del activo o puede no hacerlo, según el propósito de reutilización; la facilidad de reutilizar o no un determinado recurso se relacionará, por lo tanto, con aspectos internos y externos y no sólo con la factibilidad de su modificación.

En esta sección revisamos distintos enfoques que intentan evaluar la reusabilidad a partir de características internas. Posteriormente, repasamos diversos enfoques para analizar esta característica en productos F/OSS. A continuación resumimos las categorías propuestas en como niveles de preparación para la reusabilidad, los que pueden servir como referencia para la calificar el software.

5.2.4.1 Reusabilidad y las métricas CK

Goel y Bhatia [133] analizaron la incidencia de las métricas CK como potenciales predictores de la factibilidad de reutilización de piezas de software orientado a objetos. A partir de esos indicadores propusieron 3 nuevas métricas compuestas con miras a evaluar la reusabilidad. Los autores asocian los pares de métricas según criterios analíticos.

Estas métricas son:

- Métrica 1: DIT+NOC
- Métrica 2: CBO+LCOM
- Métrica 3: WMC+RFC

Dejan planteado para el futuro la evaluación de la incidencia de esa métricas respecto de la reutilización, buscando correlaciones entre ellas con indicadores de performance respecto de las pruebas (testing), la calidad y el esfuerzo requerido para el mantenimiento.

5.2.4.2 Métricas relacionadas con la reusabilidad y los postulados de Weyuker

Elaine Weyuker [134] planteó una serie de postulados que debería cumplir una métrica que refleje la complejidad del software. Gandhi y Bhatia analizaron las métricas denominadas MTIF y ATIF, que miden respectivamente la cantidad de métodos y atributos heredados respecto del total de métodos y atributos. Consideran que ambas cumplen con la mayoría de los postulados de Weyuker (excepto uno que no se adecua a las métricas Orientadas a Objetos) y que brindan una guía para la reusabilidad ya que la herencia implica menor escritura de código.

5.2.4.3 Aproximaciones a la medición de la reusabilidad

Bhatia propuso evaluar la reusabilidad de una clase a partir de las siguientes premisas:

- Hay mayor posibilidad de reusabilidad en las clases que se encuentran más profundas en el árbol de herencia, ya que aprovechan métodos y atributos de las clases de las que desciende.
- Una cantidad moderada de NOC (cantidad de clases) refleja potencial reusabilidad, en tanto que NOC se encuentre por debajo de un umbral
- El acoplamiento excesivo indica debilidad en el encapsulamiento de una clase, por lo que un valor alto puede inhibir la reutilización

En base a estas asunciones, propone la siguiente fórmula para una métrica de

reusabilidad:

$$Reusabilidad = a * DIT + b * NOC - c * CBO$$

El autor realiza un estudio empírico sobre 5 programas orientados a objetos que apoyan la hipótesis del sentido de la incidencia de estas métricas. Es llamativo que la cantidad de clases se considere como un indicador positivo para la reusabilidad.

5.2.4.4 Evaluación de la reusabilidad de componentes F/OSS para líneas de productos

Fazal-e-amin y otros[135] abordaron la reutilización de F/OSS desde la perspectiva del desarrollo de Líneas de Productos de Software (SPL).

Este enfoque considera que la reutilización en el marco del desarrollo en líneas de productos es de carácter sistemático; además, destacan que el Instituto de Ingeniería de Software (SEI) incluye entre las formas en que ingresa un software dentro del modelo de Líneas de Productos es mediante la adquisición de código abierto.

En este trabajo, los autores indagan sobre los factores que inciden en la reusabilidad mediante un estudio exploratorio sobre la incorporación de F/OSS en un ambiente de Línea de Productos. A continuación, proponen un modelo de reusabilidad que aplican y evalúan posteriormente. La investigación adopta una técnica mixta, que incluye métodos cualitativos y cuantitativos.

Fazal-e-amin y su grupo observan que en la reutilización en el contexto de SPL es necesario tener en cuenta la variabilidad de los componentes; este concepto es de importancia esencial en SPL.

Para definir las características de F/OSS que inciden en la reusabilidad, realizaron entrevistas a expertos tanto de la academia como del ámbito industrial. La identificación de los factores por parte de los entrevistados surgió de la aplicación de técnicas de codificación axial, abierta y selectiva.

De esta forma, se identificaron a partir de las entrevistas los siguientes factores que incidirían en la reusabilidad: flexibilidad, mantenibilidad, portabilidad, cobertura de alcance, estabilidad, comprensibilidad, historia de utilización, variabilidad.

A partir de estos factores elaboraron una fórmula de reusabilidad en la que cada uno de aquéllos se pondera con igual peso (0,16). Cada factor se calcula mediante operaciones simples sobre métricas habitualmente vinculadas con cada característica; por ejemplo, la mantenibilidad se obtiene de la suma de la complejidad ciclomática y el índice de mantenibilidad, aportando cada uno de ellos un 50% del valor.

Para evaluar el indicador compuesto que proponen los autores, realizaron una encuesta a través de un cuestionario sobre 54 estudiantes de un curso de ingeniería de software; de las respuestas elaboraron un indicador para validar las cifras obtenidas mediante el polinomio postulado. Mediante un análisis del alfa de Cronbach concluyen que los resultados reflejan la reusabilidad.

5.2.4.5 Nivel de preparación para la reutilización

No existe una escala generalmente aceptada para calificar la reusabilidad de un software. En esa dirección, el Grupo de Trabajo en Reutilización de Software de la NASA Earth Science Data Systems propusieron una escala de Niveles de Preparación para la Reutilización (Reuse Readiness Level). Marshall y Downs[136] plantean esta escala como indicador de la reusabilidad.

Los autores comienzan señalando que el valor de la reutilización de software queda en evidencia en el crecimiento de la comunidad de desarrolladores y reutilizadores de software de código abierto. A partir de este fenómeno plantean que tanto desarrolladores como usuarios (reutilizadores) necesitan de herramientas que permitan evaluar los productos desde el punto de vista de la reutilización.

Observan que la reusabilidad es un aspecto generalmente descartado en las mediciones. Toman como referencia al Nivel de Preparación de la Tecnología (TRL) propuesto por el Departamento de Defensa (DoD) norteamericano, el cual tiene por objetivo la evaluación de la usabilidad.

Toman también en consideración otros emprendimientos similares como el BRL (Nivel de Preparación para el Negocio), el Modelo de Madurez del Código Abierto (OSMM) y OpenBQR, aunque destacan que la escasa adopción de estos modelos representa una limitación.

La elaboración del RRL por parte del equipo aún está en desarrollo. Los tópicos considerados son:

- Documentación
- Extensibilidad
- Aspectos de la Propiedad Intelectual
- Modularidad
- Empaquetamiento
- Portabilidad

- Cumplimiento de estándares
- Soporte
- Prueba/Verificación

Se definen 9 niveles de preparación:

Nivel	Sumario
1	El software no es reutilizable
2	Reusabilidad inicial. La reutilización del software no es práctica
3	Reusabilidad básica: el software puede ser reutilizado por personas calificadas con un esfuerzo, riesgo y costo sustanciales.
4	La reutilización es posible. El software puede ser reutilizado por cualquier usado con algún esfuerzo, costo y riesgo
5	La reutilización es práctica. El software puede ser reutilizado por la mayoría de los usuarios con costo y riesgo razonables.
6	El software es reutilizable. Puede ser reutilizado por la mayoría de los usuarios aunque puede haber costos y riesgos
7	El software es altamente reutilizable. La mayoría de los usuarios puede reutilizarlo con costos y riesgos mínimos.
8	Reusabilidad demostrada. El software fue efectivamente reutilizado por múltiples usuarios
9	Reusabilidad probada. El software fue reutilizado efectivamente por usuarios de diferentes clases y en un amplio rango de sistemas.

5.2.5 Consideraciones

Luego de repasar diferentes propuestas para seleccionar elementos reutilizables, evaluar software libre y calificar la reusabilidad de un elemento, resulta claro que los diversos enfoques presentan una serie de aspectos comunes a tener en cuenta al momento de definir un marco general para la selección de F/OSS para la reutilización.

- Los métodos para la selección de componentes suelen enfocarse en características generales de los componentes[116] o en el diseño de un componente modelo respecto del cual se califica la mayor o menor similitud de los componentes candidatos[27].
- Una búsqueda basada en un listado exhaustivo de requerimientos obligatorios devolverá una cantidad muy baja de alternativas; por el contrario, una búsqueda demasiado general puede ocasionar nuevos problemas que deberán considerarse,

como la incorporación de nuevos requerimientos, o la necesidad de escribir código de unión o la modificación del elemento (cuando el propósito de reutilización lo permita)[137].

- Las distintas metodologías para la evaluación de F/OSS coinciden en valorar positivamente la madurez de un proyecto, la amplitud de su comunidad de usuarios/desarrolladores y en calificar las características del código fuente[126] [138], si bien las pautas para esa evaluación pueden ser diferentes.
- El nivel de preparación para la reutilización (RRL) puede brindar un marco de referencia valioso, pero no hay uniformidad en cuanto a la manera de realizar la calificación. En la mayoría de los casos, es muy difícil predecir el riesgo de adoptar un determinado software para su reuso. No obstante, en muchos casos es posible considerar que un producto F/OSS presenta “reusabilidad demostrada” o “reusabilidad probada” en función de su incorporación efectiva en otros proyectos.

5.3 Marco de Trabajo Propuesto

Tomando en cuenta los trabajos resumidos más arriba y la ausencia de referencias que permitan un enfoque sistemático para la adopción de la reutilización de F/OSS, elaboramos un marco de trabajo inicial que apunta a facilitar esa adopción. Dado su carácter general, se deberá ajustar al contexto organizacional específico, a las metodologías de desarrollo y a las estrategias de reutilización habituales.

Nos planteamos que el marco propuesto:

- Sea adaptable a diferentes estrategias de reutilización y procesos de desarrollo. En particular, se busca favorecer el abordaje sistemático del reuso, aún en perspectivas de carácter oportunista.
- Contemple las especificidades de la reutilización de F/OSS
- Favorezca la toma de decisiones respecto de la reutilización en etapas tempranas del desarrollo.

El marco se organiza según tres etapas generales: Detección de oportunidades de reutilización, búsqueda de candidatos, y evaluación y selección de recursos. El esquema se muestra en la siguiente imagen.

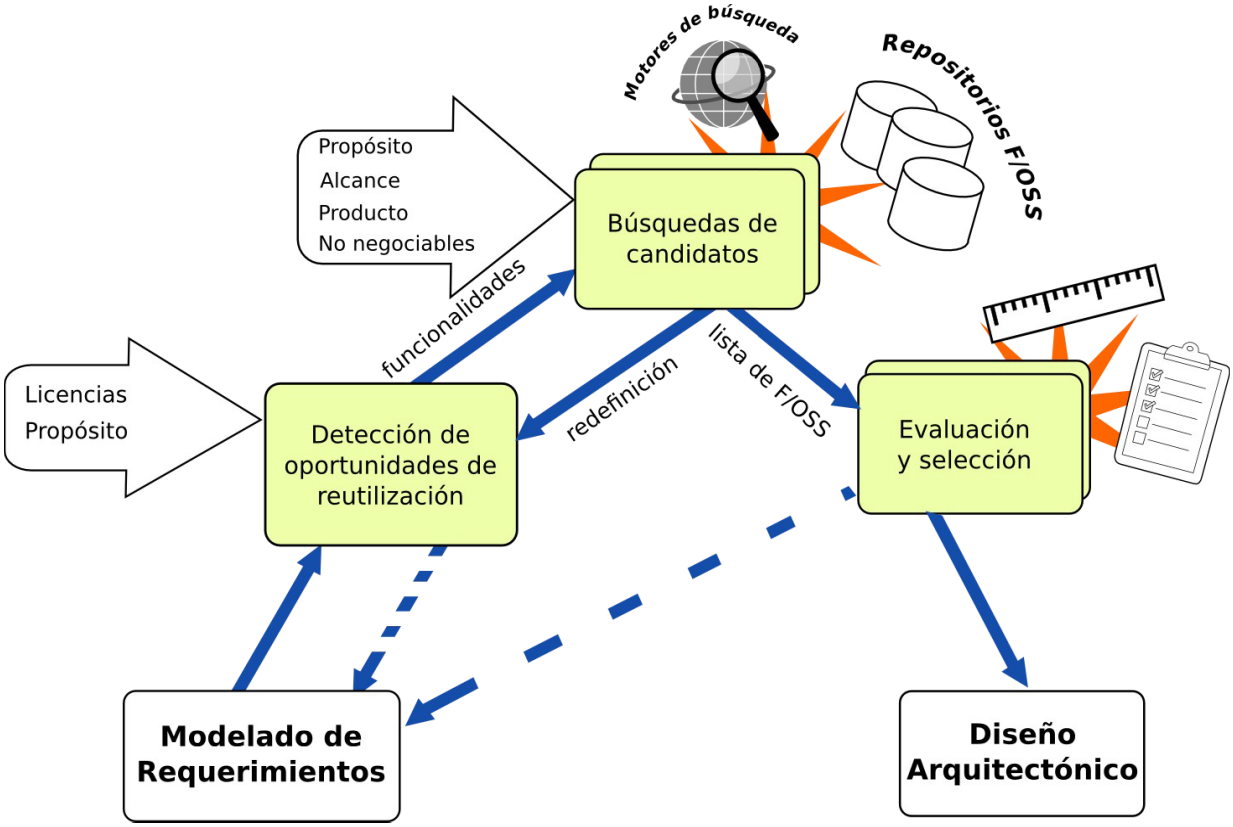

Ilustración 10: Marco para la selección de F/OSS para la reutilización

5.3.1 Etapas

5.3.1.1 Detección de oportunidades de reutilización

Para que la decisión de reutilización se adopte tempranamente en el desarrollo, se propone la elaboración de la nómina de funcionalidades pasibles de ser atendidas mediante reutilización, de manera superpuesta y paralela a los procesos de construcción de requerimientos y especificación [139].

Esa nómina orientará la búsqueda de activos pasibles de reutilizarse para atender una o varias de las funcionalidades requeridas. Para reducir el riesgo inherente a la integración de componentes, debido a los nuevos requerimientos que podrían surgir de ellos, se puede realizar un proceso iterativo, en el que se evalúen las posibles modificaciones que acarrearía la adopción para reutilización de un software determinado[137].

Bajo métodos de desarrollo tradicionales, esta etapa debería derivar directamente en la siguiente, sin modificar las especificaciones; no obstante, tanto las metodologías ágiles como el desarrollo de F/OSS asumen la modificabilidad de los requerimientos a lo largo

del proceso.

La interacción entre los procesos de elaboración de los requerimientos y el de detección de oportunidades estará condicionada por la metodología de desarrollo; en cualquier caso, es necesario establecer a priori el límite de la retroalimentación entre los procesos de desarrollo de requerimientos y la detección de oportunidades.

La salida de esta etapa es una lista de funcionalidades

5.3.1.2 Búsqueda de candidatos

A partir de las funcionalidades determinadas en el paso anterior, se conduce la búsqueda de activos potencialmente reutilizables. Otro factor que se especifica al ingresar a esta etapa es la del tipo de licencias admisibles, ya que los recursos a reutilizar deben seleccionarse entre los que se distribuyen bajo licencias compatibles.

Este proceso requiere haber establecido, de acuerdo con las facetas propuestas por Prieto-Díaz[18].:

- El propósito de la reutilización (caja negra, caja blanca, caja de vidrio)
- El producto a reutilizar (componente binario, componente en código fuente, snippet)
- El alcance de la reutilización (dentro de un dominio particular o de carácter general)
- Requerimientos no negociables

Estos aspectos definirán:

- Las herramientas de búsqueda (motores de búsqueda de carácter general, repositorios F/OSS, repositorios orientados a dominios)
- La ponderación relativa de los criterios de selección de componentes.

De esta actividad surgirá una lista de productos F/OSS candidatos a ser reutilizados para cubrir las funcionalidades determinadas en el primer paso. En esta instancia, podrá volverse al proceso anterior para una nueva definición de funcionalidades y eventualmente para modificar los criterios y el propósito.

Esta tarea aparece en principio como menos estructurada y sistemática; no obstante, existen diversas iniciativas que podrían mejorar este escenario. En el ámbito industrial, hay empresas que disponen de personal específico para el conocimiento y mantenimiento de repositorios propios de recursos F/OSS[124]; también se ha propuesto la conformación de un portal Web que permita la certificación de este tipo de software[140]; un tercer

enfoque consiste en general un catálogo de manera colaborativa, donde la información se centre específicamente en las posibilidades de reutilización[12].

Cabe destacar que la búsqueda de F/OSS para cubrir diferentes funcionalidades puede realizarse de manera paralela, en la medida en que las funcionalidades en cuestión sean independientes entre sí.

5.3.1.4 Evaluación y Selección

De la lista de candidatos generada en la instancia anterior, se realizará una comparación cuyos elementos variarán según la propósito de reutilización.

propósito	Elementos principales a evaluar	Metodologías complementarias
Caja Negra	Funcionalidades provistas, documentación de la interfaz, popularidad	OSMM, QSOS
Caja Blanca	Funcionalidades provistas, modificabilidad del código	QSOS, SQO-OSS
Caja de Vidrio	Funcionalidades provistas, indicadores de calidad del código, documentación, popularidad	OSMM, QSOS, SQO-OSS

Tabla 12: Atributos principales de acuerdo con la propósito de reutilización, y su relación con posibles métodos de evaluación de F/OSS

La selección del recurso surgirá del proceso de evaluación, que puede basarse en una de las metodologías reseñadas, teniendo en cuenta el esquema de prioridades señalados en la tabla anterior.

Los aspectos más relevantes a considerar desde la perspectiva de la reutilización serán la cobertura de las funcionalidades requeridas, la madurez de un proyecto, la documentación disponible y las características del código fuente; este conjunto de características está disponible en mayor o menor medida en todos los productos F/OSS. Otro elemento que se puede considerar aquí es la verificación de la reutilización efectiva del producto que se analiza; que un software sea efectivamente utilizado en otros proyectos es un indicio claro de su viabilidad de reutilización y al mismo tiempo brinda la posibilidad de estudiar la forma en que se reutiliza, complementando la documentación que pudiera presentar el proyecto por sí mismo.

Esta actividad sobre distintos grupos de productos F/OSS vinculados con distintas funcionalidades, puede realizarse de manera paralela para cada una de ellas.

La reutilización de caja de vidrio debería considerarse como primera opción en la

selección de F/OSS, ya que su integración es más simple y afecta en menor medida al proceso de diseño de una aplicación; por otra parte, la tarea de mantenimiento del elemento a reutilizar queda a cargo de la comunidad que lo mantiene. No obstante, la selección puede ser menos rigurosa en cuanto a la funcionalidad, lo que implicaría tener en cuenta las posibilidades de adaptación. La ventaja de la estrategia de caja de vidrio -posible por las licencias de F/OSS- es la posibilidad de evaluar aspectos de calidad a partir de la disponibilidad del código, habitualmente no disponible en COTS.

Esto supone una ventaja respecto de la reutilización tradicional, dado que no es necesario que la consideración de un recurso haya sido evaluado por un organismo de certificación, pudiéndose realizar dentro de la propia organización que se propone desarrollar con reutilización.

Capítulo 6: Desarrollo de una Herramienta de apoyo a la selección de F/OSS

6.1 Presentación

En las secciones anteriores observamos que algunas métricas podían servir como advertencia de posibles debilidades en el diseño de una aplicación, aspecto relevante al momento de decidir la reutilización de un producto F/OSS.

Posteriormente, repasamos someramente la literatura referida a la evaluación selección y evaluación referida tanto a componentes para la reutilización como a aplicaciones F/OSS, la medición de la reusabilidad y la reutilización específicamente de F/OSS.

En la sección anterior se elaboró una propuesta de marco de trabajo para la reutilización de F/OSS, con miras a sistematizar el proceso de selección correspondiente.

En esta sección describiremos el desarrollo de una sencilla herramienta para la medición de métricas de aplicaciones escritas en Java; la necesidad de contar con la herramienta surgió específicamente del proceso de investigación.

Para desarrollar la herramienta propuesta en este trabajo, nos planteamos reutilizar F/OSS que realice parte de las tareas necesarias; aplicaremos el marco de trabajo presentado anteriormente a fin de explorar la viabilidad del mismo en un caso concreto.

Si bien la herramienta en cuestión es sencilla, hay dos aspectos que la convierten en una prueba interesante para el marco: 1) Debe desarrollarse en poco tiempo y 2) Debe cumplir con una tarea surgida de necesidades reales.

6.1.1 Características esperadas de la herramienta

La herramienta que nos planteamos desarrollar debía computar una serie de métricas, almacenarlas en un formato estándar (que pudiera ser leído por otras herramientas) y dar un informe respecto de las posibles debilidades de diseño, de acuerdo a la investigación realizada en la sección correspondiente.

Más específicamente, se necesitaba una herramienta libre que computara una serie de métricas a partir del código fuente de una aplicación escrita en Java; el almacenamiento de la información se debe realizar en XML.

A partir de un directorio raíz la herramienta debería analizar los archivos .java, construir un modelo representativo de las estructura del código e informar los valores de las métricas NOC (número de clases), NOM (número de métodos, incluyendo constructores), CYCLO

(Complejidad ciclomática total), CALL (Cantidad total de llamadas diferentes a métodos en todo el código) y FOUT (Suma de los valores de fan-out[141] para todas las clases). Estas métricas fueron seleccionadas a partir de las propuestas de Lanza y Marinescu[72], con miras a estudiar la evolución del diseño de F/OSS e informar sobre la superación de umbrales en el promedio de algunas métricas[15].

La definición de los umbrales que se considerarán también deberían leerse de un archivo XML.

6.2 Aplicación del marco de trabajo

6.2.1 Detección de oportunidades de reutilización

Para iniciar esta etapa, se definió que el software se distribuiría bajo la licencia GPL 3; esto significa que las alternativas que eventualmente se consideren deben distribuirse bajo una licencia compatible con ella.

La lista de métricas a medir y las facilidades de lectura y almacenamiento en formato XML conformaron la nómina de funcionalidades.

Funcionalidades	Descripción
Lista de métricas a computar sobre código fuente en Java	NOC (número de clases), NOM (número de métodos, incluyendo constructores), CYCLO (Complejidad ciclomática total), CALL (Cantidad total de llamadas diferentes a métodos en todo el código) y FOUT (Suma de los valores de fan-out de todas las clases)
Lectura y almacenamiento en XML	Generar una estructura de árbol con los elementos y subelementos de un archivo XML Escribir un archivo XML a partir de una estructura de árbol

En una primera instancia, nos orientamos en función de una reutilización de caja blanca, bajo la hipótesis de que sería más sencillo modificar una herramienta existente que ya computara parcialmente las métricas de nuestro interés.

A continuación detallaremos sólo búsqueda de recursos para cubrir con la funcionalidad de computar el conjunto de métricas de nuestro interés.

6.2.2 Búsqueda de candidatos

La instancia de búsqueda se realizó mediante la utilización de motores generales (Google), repositorios de software libre y de código abierto (SourceForge.net y java-source.net), y foros de desarrolladores como StackOverflow. La búsqueda se orientó en primer lugar a analizar y evaluar herramientas que realicen mediciones de métricas

relacionadas con el tamaño y con el diseño de software orientado a objetos, de modo de incorporarlas o adaptarlas en nuestro desarrollo.

En primer lugar se buscaron las alternativas para el cómputo de métricas. Posteriormente, se realizó una búsqueda de recursos que permitieran operar con archivos XML.

Las aplicaciones encontradas fueron PMD, JavaNCSS, JDepend, CodeAnalyzer y JMT. En cada una de ellas se verificó la lista de métricas que obtenía y se analizó el código para evaluar la posibilidad de modificarlas e incorporar las métricas no provistas.

Herramienta	Licencia	Métricas	Sitio
PMD	PMD BSD-style license	Verifica el cumplimiento de reglas	http://pmd.sourceforge.net/pmd-5.1.3/
JNCSS	GPL	NCSS, NOC, NOP	http://www.kclee.de/clemens/java/javancss/
JDepend	BSD	NOC, AC, EC, Abstracción, Inestabilidad, Distancia de la secuencia principal	http://clarkware.com/software/JDepend.html
CodeAnalyzer	GPL	LOC, Comments Lines, Comments Ratio	http://www.codeanalyzer.teel.ws/
JMT	N/D	Métricas de Chidamber y Kemerer	http://www-ivs.cs.uni-magdeburg.de/sw-eng/agruppe/forschung/tools/

Tabla 13: Lista de herramientas analizadas con miras a su posible reutilización

En cada se analizó el método *main* de la clase que aparece como principal para determinar qué instancias de qué clases realizan el análisis de código fuente, a fin de determinar cuáles son las clases que tienen responsabilidades en el cómputo de las métricas y de qué información disponen las instancias de esas clases. El método es similar a los descritos por Kotonya[22] en lo referente a la reutilización de porciones de software descartadas; en nuestro caso, contamos con la colaboración del IDE NetBeans,

JavaNCSS y JDepend resultaron fáciles de integrar, pero las modificaciones necesarias requerirían de un conocimiento profundo de la estructura de los proyectos; en este aspecto, la escasez de documentación resultó determinante para buscar otras

alternativas.

Con estas consideraciones, se replanteó el proceso de detección de oportunidades, apuntando ahora a obtener un parser de F/OSS que facilitara extraer las métricas. La búsqueda se orientó teniendo en mente la reutilización de caja de vidrio (contemplando la posibilidad de analizar el código, pero sin modificar la funcionalidad).

En esta instancia se descartaron los generadores de parsers (ANTLR, JavaCC), ya que dependen de gramáticas específicas, con lo que la lista se redujo a dos productos: JavaParser versión 1.5 y Qdox, versión 2.0. En ambos casos, se trata de las últimas versiones encontradas al momento del desarrollo.

6.2.3 Evaluación y Selección

La siguiente tabla resume los aspectos considerados para la evaluación de los dos candidatos.

Luego de realizar pruebas con ambos, la selección recayó en el Qdox. Sobre esta biblioteca se puede considerar que muestra reusabilidad probada y ofrece código bien documentado; cabe señalar, además, que el valor de estructuración de clases está por debajo del umbral propuesto en este mismo trabajo; en cambio, el promedio de lacantidad de métodos por clase en el otro producto candidato es superior a 13.

Producto	Documentación	Reutilización comprobada	Código
JavaParser, versión 1.5	Algunos ejemplos en el sitio Web https://code.google.com/p/javaparser/wiki/UsingThisParser	No se encontró	Estructuración de clases superior a 13 (13,52)
Qdox, versión 2.0	Ejemplos de uso. JavaDoc detallado	Proyectos AspectWerkz y Cocoon	Estructuración de clases por debajo de 13 (11,02)

Tabla 14: Comparación de JavaParser y Qdox

En el caso de la selección de una biblioteca para gestionar XML, preferimos utilizar XML por la experiencia previa en su reutilización [10]. No obstante, se podría haber seguido la misma secuencia que en la selección del parser de no haberse contado con el antecedente mencionado o haber destinado más tiempo a la tarea.

6.3 Observaciones sobre el desarrollo

El desarrollo de la primera versión capaz de obtener las métricas requeridas y almacenarlas, se completó en 3 días. Midiendo el tamaño de las distintas partes con la herramienta iPlasma[89], se verifica que Qdox tiene un tamaño de 15684 LOC, en tanto

que XOM consta de 51283 LOC. El código nuevo contiene sólo 435 LOC, en 10 clases que incluyen 31 métodos (sin considerar las pruebas).

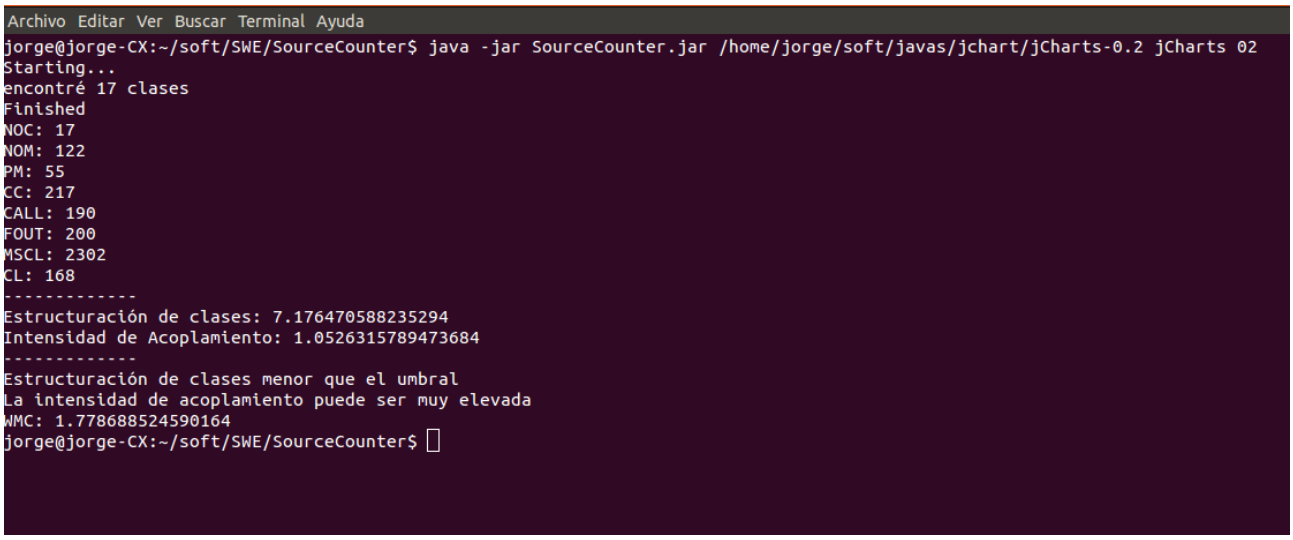

Ilustración 11: Ejecución de la Herramienta en línea de comandos

Qdox devuelve el modelo del código parseado en una estructura de árbol contenido en un objeto JavaProjectBuilder; esa clase ofrece métodos para acceder a la lista de paquetes, clases y métodos. La limitación que presenta esta biblioteca es que no analiza el código a nivel de método, por lo que fue necesario programar el cómputo de las métricas que requieren ese análisis, como la Complejidad Ciclomática o las llamadas desde y hacia las instancias de una clase.

Si bien la Complejidad Ciclomática es una métrica ampliamente conocida y utilizada, adoptamos la metodología de cálculo explicitada en la herramienta JavaNCSS[142]. En este caso, se trata de reutilización de ideas, según la clasificación de Prieto-Díaz[18].

En función de las facilidades brindadas por Qdox, y la necesidad de analizar el código de los métodos, decidimos incorporar otras dos métricas: PM (Métodos Públicos), que da cuenta de los mensajes que puede recibir una instancia; MSCL (Líneas de código no comentadas en un método) y CL (líneas comentadas).

Las decisiones de diseño priorizaron el desarrollo rápido; no obstante, se centró en una clase denominada Analyzer la responsabilidad de recibir un String con la ruta al directorio raíz del código a analizar, devolviendo una lista con las métricas computadas. Esta decisión obedece a la posibilidad de reutilizarla en un futuro, posibilitando el análisis simultáneo en varios hilos ejecutándose en forma paralela, teniendo en mente computar el conjunto de métricas para diversas versiones de un mismo software en las que el código

de cada una se encuentre bajo directorios diferentes.

6.3.1 Utilización

Para utilizar la herramienta se la invoca desde consola de acuerdo con el siguiente esquema:

```
java -jar <directorio raíz> <nombre> <version>
```

El programa analiza el código bajo el directorio raíz, y almacena una archivo XML con el nombre pasado como segundo argumento. El tercer argumento se inluye dentro del mismo archivo como atributo de un elemento que representa el nombre de la aplicación.

jCharts

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <Project name="jCharts" version="02"/>
    <metrics>
        <metric name="NOC" value="17"/>
        <metric name="NOM" value="122"/>
        <metric name="PM" value="55"/>
        <metric name="CC" value="217"/>
        <metric name="CALL" value="190"/>
        <metric name="FOUT" value="200"/>
        <metric name="MSCL" value="2302"/>
        <metric name="CL" value="168"/>
    </metrics>
</root>
```

Ilustración 12: Captura de un archivo XML generado por la herramienta para la versión 0.2 de la biblioteca jCharts

La aplicación se encuentra en https://github.com/seretur/SourceCounter y se distribuye bajo la licencia GPL v3.

6.3.2 Conclusiones sobre el desarrollo realizado

Este caso pone en evidencia la viabilidad de alcanzar desarrollos funcionales rápidos que pueden servir de base o prototipo para desarrollos posteriores.

Otro aspecto que se observa es que la búsqueda, prueba e integración de elementos F/OSS se vio facilitada por la familiaridad del autor con proyectos de este tipo escritos en

Java.

Las dos bibliotecas seleccionadas para la reutilización (Qdox y XOM) muestran interfaces de programación de la aplicación (API) simples y directos, con ejemplos claros disponibles en sus sitios web.

El hecho de que ambas bibliotecas hayan sido reutilizadas en muchos otros proyectos ofreció la oportunidad de analizar la forma en que se habían integrado a los mismos.

Capitulo 7: Conclusiones y Trabajos Futuros

7.1 Conclusiones Generales

Este trabajo muestra la viabilidad de reutilizar F/OSS para nuevos desarrollos, propone una marco de trabajo para orientar esa tarea y ofrece criterios para la selección de recursos libres para el reuso, proponiendo umbrales para proporciones de métricas que advierten sobre la posible presencia de debilidades de diseño.

La reutilización puede seguir objetivos muy distintos, entre los que podemos destacar:

- La generación rápida de un prototipo para mostrar las características de un software que se pretende desarrollar o como primera versión a partir de la cual continuar el desarrollo.
- La posibilidad de derivar en terceros el desarrollo (y eventualmente el mantenimiento) de partes del software que se encarguen de tareas complejas o que no son del interés principal para el equipo de desarrollo.

Estas perspectivas diferentes influirán en el peso relativo de los factores a considerar al momento de decidir la reutilización así como las características de la misma.

7.1.1 Umbrales como potenciales indicadores de debilidades de diseño

Se han propuesto enfoques muy diversos para emplear métricas de software para predecir aspectos difíciles de evaluar como la mantenibilidad, la evolutividad, la modificabilidad, etc. En este trabajo planteamos la utilización de umbrales de promedios de métricas que actúen como advertencias de posibles debilidades de diseño; estos indicadores se postulan luego de observar la distribución de frecuencias de distintas proporciones de métricas sobre un conjunto de versiones de productos F/OSS que se han caracterizado efectivamente por haber evolucionado (y por lo tanto, por haber sido mantenidos y modificados).

En resumen, debe prestarse atención a las posibles debilidades de diseño de una aplicación escrita en Java en la que el número de métodos por clase (estructuración de clases, según Lanza y Marinescu[72]) supere el valor de 13, la intensidad de acoplamiento (Fan-out respecto del total de llamadas[72]) sea mayor que 0,668, o los métodos ponderados por clase (WMC[64]) esté en promedio por encima de 45,68.

Los umbrales propuestos aquí corresponden a métricas de diseño, por lo que -en principio- serían valederas para software escrito en otros lenguajes orientados a objetos;

sin embargo, tal hipótesis debería contrastarse con nuevos estudios empíricos.

Los proyectos analizados en este trabajo (cuyo listado se detalla en el Anexo III) en general son de tamaño mediano a grande; en proyectos pequeños es probable que los umbrales propuestos sean menos relevantes, dadas las características de la media como medida de tendencia central. influenciada por los valores extremos.

Coincidimos con Fowler[103] en que ningún número puede rivalizar con la intuición humana; sin embargo, pueden servir como llamadas de atención que nos indiquen la necesidad de mirar con más cuidado el diseño de un producto F/OSS que nos planteamos reutilizar.

Cabe señalar que los umbrales propuestos no se vinculan directamente con atributos particulares del software; no obstante se refieren a la distribución efectiva, empíricamente analizada, de estas proporciones en aplicaciones que han producido diversas versiones durante varios años, dando muestras de la viabilidad de que sean modificados y de sus posibilidades evolutivas.

7.1.2 Utilidad del marco de trabajo

En relación con el marco de trabajo propuesto, será preciso someterlo a prueba en distintos contextos. De la experiencia realizada sugiere que es viable que este marco oriente el proceso de selección de F/OSS por lo menos en el contexto de desarrollos con fines académicos y en los que se precisan resultados en un plazo breve; no obstante, la similitudes con muchos desarrollos F/OSS así como los aspectos comunes con metodologías ágiles sugiere la posibilidad utilidad en otros contextos. Se buscó que la formulación sea lo bastante amplia como para ajustarse a escenarios y metodologías diferentes.

De todos modos, consideramos valioso disponer de una referencia para comenzar a sistematizar el proceso de selección con las características particulares de este tipo de software.

Un factor que incide en la factibilidad de reutilizar F/OSS es la experiencia y el conocimiento previo de proyectos de este tipo; una organización podría disponer de personal específico para llevar la información y realizar experiencias con F/OSS (como en el caso reportado por Morad y Kuflik sobre Orbotech[124]), o podría conformar un equipo con estas características.

La enorme cantidad de recursos disponibles bajo licencias de F/OSS lleva a plantearse la necesidad de contar con profesionales que tengan un conocimiento específico sobre el tema, comprendiendo las particularidades, las fortalezas y las debilidades del trabajo con

estos productos.

Por otra parte, la investigación empírica sobre F/OSS[38] es un campo creciente que ofrece nuevas temáticas y oportunidades para la ingeniería de software. Este aspecto, sumado al mencionado en el párrafo anterior, plantea la necesidad de incluir estas temáticas en la formación de profesionales en informática.

7.2 Trabajos futuros

De la experiencia y de los resultados de esta tesis surgen una serie de temáticas a abordar como continuación de este trabajo.

Por un lado, en lo que respecta a métricas de diseño, es necesario realizar análisis similares en otros lenguajes de programación y analizar las posibles incidencias en distintos dominios.

Dado que la media es una medida sensible a los valores extremos, sería interesante analizar su pertinencia en proyectos pequeños, donde unas pocas clases podrían producir resultados que no sean representativos. En el mismo sentido, sería interesante realizar estudios en relación a otras medidas de tendencia central u otras que consideren la distribución de las métricas al interior del código fuente de una aplicación.

Las métricas consideradas en esta tesis se basan fundamentalmente en las adoptadas por Lanza y Marinescu[72] para la caracterización del diseño de un software. Sería interesante observar el comportamiento de otras métricas, como la proporción entre comentarios y líneas de código totales, así como otras medidas relacionadas con el acoplamiento.

En lo que respecta a la reutilización de F/OSS y al marco de trabajo propuesto, nos planteamos analizar estas temáticas tanto dentro del desarrollo de F/OSS como en relación a su posible adecuación a metodologías ágiles, en los cuales la reutilización es un tema de investigación creciente[143] [144] [145]. Las interrelaciones entre metodologías ágiles y F/OSS constituyen áreas de investigación de interés, lo que se refleja en el incremento en las publicaciones sobre el tema [146] [147].

La herramienta desarrollada se utilizará como base para investigaciones empíricas relacionadas con las tendencias y etapas de la evolución de software libre y de código abierto. Las características de la herramienta posibilitan obtener un conjunto de métricas sobre el código fuente, por lo que podría extenderse para realizar esa tarea de manera paralela sobre distintas versiones de un mismo producto.

Anexo I

Métricas mencionadas en este trabajo

Métrica	Nombre	Descripción
MI	Índice de Mantenibilidad	Parte de un valor máximo al que se le van restando ponderadamente los promedios del Volumen (V), la Complejidad Ciclomática Extendida(CCE), el tamaño en cantidad de Líneas de Código (LOC) y una función del porcentaje de líneas con comentarios (PorCom)
CYCLO	Complejidad Ciclomática	Definida por McCabe como la cantidad de caminos independientes a través del programa
V	Volumen de Halstead	$V = N + \log_2(\eta)$ Donde N es la cantidad total de operadores y operandos y η es la cantidad de operadores y operandos distintos
Ca	Acoplamiento Aferente	Número de paquetes que dependen de clases dentro del paquete analizado
Ce	Acoplamiento Eferente	Cantidad de paquetes de los cuales dependen las clases de un paquete determinado
WMC	Métodos ponderados por clase	$WMC = \sum_{i=1}^{n} c_i$ Sumatoria de las complejidades de los métodos
LCOM	Falta de cohesión	Proporción de métodos disjuntos (es decir, que no comparten variables de instancia) respecto del total de métodos.
CBO	Acoplamiento entre objetos	El CBO de una clase es la cantidad de otras clases a las que está acoplada (es decir, a las que hace referencia o desde las cuales es referenciada)
DIT	Profundidad en el árbol de herencia	Cantidad de ancestros de una clase hasta la raíz
LCC	Loose Class Cohesion	Proporción de pares de métodos conectados de manera directa o transitiva (es decir, pares conectados a un atributo por sí o a través de otro método que invocan)
COF	Factor de Acoplamiento	$COF = \frac{\sum_i [\sum_j esCliente(C_j, C_i)]}{TC^2 - TC}$ La función esCliente vale 1 cuando las clases están relacionadas mediante un paso de mensaje o una referencia a método o atributo de la otra.
LOC	Líneas de código	Líneas de código incluyendo líneas en blanco y comentarios

Métrica	Nombre	Descripción
NOM	Número de métodos	Cuenta otras operaciones definidas en toda la aplicación
NOC	Cantidad de clases	
NOP	Cantidad de paquetes	
CALL	Cantidad de invocaciones a operaciones	
FOUT	Cantidad de clases referenciadas por cada clase,	Medida propuesta por Henry y Kafura, adaptada a la orientación a objetos
PM	Métodos Públicos	Cantidad total de métodos públicos de todas las clases
NCSL	Líneas de código no comentadas	Cantidad de líneas de código sin comentarios de todos los métodos
CL	Líneas comentadas	Líneas de comentarios en el código. Se cuentan también las que están a continuación de una instrucción.

Anexo II

Datos obtenidos

id_asset identifica a cada uno de los proyectos analizados en este trabajo.

id_asset	CYCLO	LOC	NOM	NOC	NOP	FOUT	CALL
1	16177	63028	5528	402	34	9138	17166
1	16071	63409	5607	410	34	9414	16932
1	16791	66357	5881	441	35	9772	17522
1	17100	68083	5960	447	35	9869	17701
1	17165	68220	5972	448	35	9899	17745
1	17208	68387	5979	448	35	9916	17786
1	17160	68334	6049	454	36	9938	17848
2	9304	48024	5079	553	62	8199	13111
2	9950	50980	5369	573	64	8582	13821
2	9952	51012	5372	573	64	8588	13824
2	10220	52055	5501	585	64	8699	14084
2	10421	53564	5592	595	66	9032	14573
2	10495	53922	5628	597	66	9074	14686
2	10903	55994	5771	609	66	9348	15231
2	15840	81263	7982	897	88	14052	22873
2	15902	81761	7989	899	88	14080	22949
2	16333	85535	8260	921	95	14576	23756
2	16361	85812	8263	921	95	14588	23775
2	16408	86173	8286	925	95	14654	23894
2	16817	88623	8578	937	97	15102	24790
2	16906	89088	8594	937	97	15164	24890
2	17102	90604	8651	949	97	15333	25339
2	17518	93073	8799	972	98	15727	26218
2	17869	95003	8900	989	98	16006	26663
2	18028	96089	8956	995	99	16011	27043
3	462	3154	259	20	2	398	729
3	462	3154	259	20	2	398	729

id_asset	CYCLO	LOC	NOM	NOC	NOP	FOUT	CALL
3	469	3276	264	22	2	416	759
3	512	3561	278	23	2	442	821
3	515	3575	278	23	2	449	827
3	519	4280	279	23	2	449	829
3	579	3944	338	24	2	560	970
4	8569	35198	5061	487	33	7050	10725
4	8584	35425	5099	494	34	7102	10804
4	8716	36012	5236	515	34	7322	10988
4	8783	36344	5301	517	35	7411	11094
4	7880	32975	4867	487	29	6869	9998
4	8002	33452	4934	494	29	6946	10129
4	8215	34263	4995	494	30	7027	10267
4	8502	35326	5153	508	30	7306	10716
4	8668	35869	5222	515	30	7387	10827
4	8748	36214	5256	517	31	7475	10948
4	8764	36209	5271	519	31	7513	10937
7	13572	75660	7024	664	53	11728	20792
7	14089	81820	7283	688	54	12496	22347
7	14681	87512	7688	723	54	13267	23728
7	14950	89854	7858	739	54	13646	24530
7	14967	89915	7868	741	54	13653	24540
7	14959	90018	7868	738	54	13683	24615
7	15022	90099	7897	740	54	13755	24731
7	15987	96071	8355	777	64	16843	27216
7	17319	104987	9284	852	67	18998	30773
7	17977	109446	9648	884	72	20008	32518
7	18361	112115	9839	895	72	20441	33373
7	18766	115760	10030	905	70	21010	34488
7	19503	120563	10380	917	69	21979	36190
7	19601	121186	10439	919	69	21934	36226
7	19622	121273	10452	922	70	21952	36251

id_asset	CYCLO	LOC	NOM	NOC	NOP	FOUT	CALL
7	19709	121845	10493	922	70	22074	36521
7	20349	125834	10904	938	71	22669	37689
7	21107	129587	11258	957	68	23229	38756
7	21329	131101	11354	951	68	23526	39312
7	22325	136083	11772	991	70	24378	40875
10	8083	38708	2478	190	4	5507	9339
10	12586	60215	3528	295	10	8064	13643
10	13674	64227	3850	313	14	8905	14857
10	13385	61078	3982	320	17	8695	14583
10	13860	63211	4110	329	17	8948	15042
10	14465	66079	4291	352	18	9335	15816
10	14967	68069	4380	358	18	9580	16298
10	15469	70154	4616	369	19	9788	16533
10	15472	70171	4617	369	19	9789	16535
10	15913	71693	4924	382	20	10964	18232
10	16189	72822	5027	386	20	11080	18497
10	16998	76418	5188	394	20	11874	19699
10	17306	78064	5323	402	20	12215	20269
10	17755	80233	5418	412	20	12511	20818
10	17806	80289	5418	411	20	12557	20869
10	19445	89356	5798	431	20	13800	23196
10	19449	89386	5798	431	22	13832	23208
10	20626	96169	6185	462	26	14628	24681
10	20685	96462	6193	463	26	14671	24766
10	20775	97759	6290	476	25	14985	25253
10	20788	97925	6300	478	25	15010	25313
10	21380	100862	6520	486	26	15796	26812
10	21382	100910	6521	486	26	15812	26829
10	22260	104306	6713	494	26	16222	27593
10	22345	104794	6725	495	26	16276	27717
10	22349	104813	6726	495	26	16279	27723

id_asset	CYCLO	LOC	NOM	NOC	NOP	FOUT	CALL
10	22614	105656	6804	497	26	16536	28123
10	22617	105685	6804	497	26	16540	28130
11	532	2268	392	28	7	599	922
11	575	2448	421	25	6	773	1194
11	706	2932	477	41	8	887	1369
11	725	3024	475	40	8	891	1406
11	998	4229	647	58	10	1190	1907
11	1010	4290	649	58	10	1205	1935
11	1271	5378	783	64	10	1512	2419
11	1569	6856	897	67	10	1758	2952
11	1750	7216	1017	69	10	1945	3272
11	1764	7264	1023	69	10	1951	3285
11	1996	8425	1092	70	10	2133	3664
11	2270	9535	1229	74	10	2412	4189
11	2733	11613	1448	92	10	2828	4972
11	4660	22033	2645	199	16	4786	8286
11	7939	41682	4982	513	30	7738	13226
11	7982	36911	4475	361	32	8507	14027
11	7160	33131	4103	336	32	7595	12551
11	7318	33886	4199	346	32	7830	12856
11	7522	34972	4300	352	32	8062	13246
11	7580	35443	4323	354	32	8135	13383
11	7620	35428	4337	355	32	8157	13429
11	7661	35638	4365	358	32	8205	13511
11	9647	45978	5583	442	47	11204	17848
11	7818	35986	4431	360	32	8761	14205
11	9736	46365	5610	442	47	11252	17945
11	7892	36324	4452	360	32	8798	14278
11	7900	36416	4457	360	32	8408	13838
11	7917	36500	4461	360	32	8432	13887
11	7923	36572	4464	360	32	8466	13943

id_asset	CYCLO	LOC	NOM	NOC	NOP	FOUT	CALL
11	7943	36681	4468	360	32	8492	13989
11	7951	36751	4473	360	32	8506	14015
11	7950	36757	4473	360	32	8507	14018
11	7950	36757	4473	360	32	8507	14018
11	7950	32757	4473	360	32	8507	14018
11	7994	36937	4492	360	32	8528	14060
11	7997	36954	4493	360	32	8529	14063
11	7998	36965	4494	360	32	8531	14066
11	8019	37010	4500	361	32	8535	14068
11	8019	37013	4500	361	32	8536	14067
12	6737	33585	2230	202	28	4094	6895
12	6775	33764	2236	203	28	4111	6935
12	9304	47745	3207	305	56	6049	10192
12	9322	47857	3216	305	56	6075	10234
12	9413	48329	3241	305	56	6126	10359
12	9565	49413	3263	306	56	6157	10428
12	9573	49498	3266	306	56	6160	10435
12	9853	51340	3344	314	57	6298	10752
12	9886	51569	3357	315	57	6331	10815
12	10100	52860	3425	321	57	6494	11109
12	10101	52862	3425	321	57	6494	11109
12	10101	52862	3425	321	57	6494	11109
12	10137	53073	3437	322	57	6512	11149
12	10181	53186	3442	322	57	6525	11185
12	10202	53224	3445	322	57	6539	11208
12	11336	59581	4096	418	64	8162	13771
12	11442	60000	4111	420	64	8205	13853
12	11562	60399	4155	425	66	8271	13984
12	11622	60738	4190	427	66	8312	14053
12	11696	61061	4203	428	66	8354	14108
12	11759	61253	4211	429	66	8362	14128

id_asset	CYCLO	LOC	NOM	NOC	NOP	FOUT	CALL
12	11870	61717	4234	431	66	8410	14229
12	11944	61925	4243	431	66	8429	14285
12	11972	61986	4337	442	66	8541	14357
12	11965	62060	4332	442	66	8518	14308
13	4208	21437	1263	62	3	2702	5501
13	4336	22397	1306	63	3	2801	5693
13	4736	24506	1403	68	3	2995	6118
13	5147	26827	1493	70	2	3145	6550
13	5168	26928	1494	70	2	3147	6562
13	6371	35295	1956	117	14	4262	8452
13	6459	36128	1980	119	14	4389	8684
13	7202	41055	2247	132	16	4802	10003
13	7223	41239	2253	131	16	4820	10045
13	7273	41552	2263	131	16	4830	10082
13	7293	41769	2168	98	4	4629	9888
14	12345	44283	5193	822	76	7465	13641
14	12327	45027	5177	678	82	7790	14098
14	12538	46023	5234	689	82	8091	14193
14	13562	50751	5867	742	88	9080	15853
14	14035	52576	6021	757	90	9298	16252
14	13586	50867	5875	742	88	9097	15883
14	13634	51078	5902	747	88	9118	15940
14	13641	51109	5905	747	88	9122	15947
14	13667	51203	5908	748	89	9136	15985
14	13813	51979	5971	754	90	9224	16116
14	14035	52576	6021	757	90	9298	16252
14	15728	60005	7178	923	101	10784	18861
14	15808	60428	7211	927	101	10852	19008
14	12345	44283	5193	822	76	7465	13641
14	12327	45027	5177	678	82	7790	14098
15	2307	8251	1032	113	13	1302	2307

id_asset	CYCLO	LOC	NOM	NOC	NOP	FOUT	CALL
15	2678	9626	1204	137	18	1564	2769
15	3106	11189	1458	165	22	1844	3211
15	3289	11811	1527	172	22	1957	3405
15	3968	14577	1876	211	31	2508	4357
15	4227	15423	1986	227	33	2684	4676
15	4454	16100	2112	236	35	2771	4791
15	5656	20935	2566	281	41	3476	6090
15	5673	20997	2573	281	41	3487	6105
15	5682	21019	2576	281	41	3490	6109
15	6339	24570	2908	324	44	4249	7494
16	406	2492	242	19	5	422	812
16	416	2945	248	20	5	449	862
16	435	3003	260	22	5	553	1025
16	442	3023	260	22	5	461	875
16	474	3171	274	24	5	487	935
16	475	3174	275	24	5	488	936
16	482	3203	276	24	5	494	951
16	502	3308	290	24	5	504	976
16	540	3514	306	25	5	531	1040
16	621	3978	369	26	5	696	1359
16	621	3962	369	26	5	694	1357
17	979	5780	1255	126	17	561	1079
17	1057	6390	1379	131	20	686	1302
17	1111	6942	1392	132	20	757	1425
17	1184	7476	1456	137	21	836	1545
17	1381	7044	1602	148	17	1711	2379
17	1398	7132	1638	155	17	1721	2409
17	2063	10939	1967	175	12	2337	3393
17	2291	11580	2029	184	13	2460	3583
17	4690	20543	4988	275	23	4828	6462
17	4928	21847	5271	282	25	5134	6895

id_asset	CYCLO	LOC	NOM	NOC	NOP	FOUT	CALL
17	5218	22844	5615	298	25	5485	7305
17	5172	22301	5819	301	24	5579	7355
17	6974	28787	6779	531	26	6472	9695
17	6995	28989	6739	531	26	6522	9760
17	7175	29992	6863	537	26	6660	9985
17	7458	31120	7175	547	26	7207	10697
17	7751	31865	7245	562	27	7282	10820
17	7253	32113	6521	628	26	7033	10629
17	7263	32119	6523	628	26	7045	10641
17	7531	33597	6653	640	26	7706	11191
17	7686	34473	5991	641	26	7843	11413
17	7689	34481	5992	641	26	7844	11416
17	7702	34646	5994	641	26	7847	11444
17	7736	34752	6028	642	27	7849	11454
17	7089	31221	5252	528	22	7099	10343
17	7137	31644	5284	529	22	7157	10473
17	8388	34865	6230	622	22	8697	12241
17	8389	34856	6230	624	22	8699	12248
17	8392	34856	6231	624	22	8704	12256
17	8401	34882	6236	624	22	8718	12270
17	9578	40154	6899	721	24	9450	13360
17	9580	40155	6899	721	24	9450	13360
17	9768	40591	6996	723	24	9578	13532
17	9835	40221	7044	731	24	8980	13319
17	9179	43589	8157	808	24	10605	15315
17	9287	44159	8376	824	24	10856	15609
17	9286	44162	8375	824	24	10853	15607
17	9660	45743	8518	842	25	11118	16052
17	11055	50928	9522	964	32	11698	17819
17	11120	51183	9560	970	32	11759	17885
17	11120	51183	9560	970	32	11759	17885

id_asset	CYCLO	LOC	NOM	NOC	NOP	FOUT	CALL
17	13231	60971	11445	1268	38	14434	21577
17	13237	60973	11447	1272	38	14434	21579
17	13245	61001	11451	1272	38	14440	21590
17	13364	61381	11706	1303	38	14640	21804
17	13440	61755	11785	1314	38	14719	21900
17	13815	62966	12384	1399	38	14969	22289
17	13858	63483	12451	1415	38	15008	22351
17	13962	63891	12595	1455	38	15072	22443
18	5298	23834	2570	115	13	6744	10628
18	5562	24788	2710	119	13	7010	11102
18	2857	12742	1389	123	13	3593	5712
18	2881	12881	1379	122	13	3593	5751
18	3064	13929	1461	135	15	3826	6081
18	3207	14481	1532	141	16	3970	6304
18	3257	14799	1553	143	16	4067	6448
18	3340	15316	1594	148	16	4211	6614
18	3376	15455	1606	149	16	4270	6704
18	3548	16107	1677	155	17	4492	7050
18	3653	16546	1705	157	17	4596	7233
18	3677	16635	1713	158	17	4632	7279
18	3692	16792	1720	158	17	4677	7343
18	3720	16993	1727	158	17	4745	7435
18	3784	17259	1759	159	17	4815	7549
18	3852	17564	1790	161	17	4902	7671
19	18388	88006	10244	1293	73	12046	19297
19	19829	94462	10771	1355	77	12899	20741
19	20471	96989	10878	1361	77	13155	21247
19	20792	101854	11045	1461	80	13535	22075
19	26464	125813	12601	1453	75	16294	26367
19	21403	104155	11263	1434	78	13478	21844
19	21434	104450	11278	1438	78	13536	21948

id_asset	CYCLO	LOC	NOM	NOC	NOP	FOUT	CALL
19	21471	103512	11301	1439	78	13572	22014
19	24875	118862	11931	1389	70	15428	24908
19	22713	108913	11615	1407	79	14076	22915
19	22999	110510	11733	1416	79	14370	23313
19	23352	112349	11843	1427	79	14586	23678
19	23356	112343	11843	1427	79	14586	23679
19	23977	116983	11593	1367	69	15152	24562
19	24202	116769	11674	1369	70	15051	24270
19	24875	118862	11931	1389	70	15428	24908
19	26464	125813	12601	1453	75	16294	26367
19	32290	150241	14643	1447	80	19860	31921
19	33408	155459	15225	1463	82	20224	33370
20	25102	112128	10055	950	100	18734	31177
20	25721	115027	10354	975	100	18057	31161
20	25453	113501	10291	984	99	17909	31071
20	25453	113501	10291	984	99	17909	31071
20	25084	112161	10097	970	98	17798	30806
20	25198	109272	10547	989	95	17095	28898
20	25369	109689	10609	998	96	17233	29063
20	25485	110121	10644	1002	97	17309	29208
20	25543	110312	10649	1002	97	17331	29266
20	25542	110302	10633	1002	97	17309	29254
20	25616	110664	10648	1003	97	17390	29363
20	25675	110872	10668	1005	97	17430	29431
20	26363	111230	10444	997	97	17254	29068
20	26541	111614	10546	998	97	18249	29908
20	24707	104611	10234	986	99	17142	29052
20	24738	104749	10238	986	99	17763	29084
20	26853	113112	10617	1010	99	18411	30309
20	25770	107756	10207	1018	113	17917	30781
20	26283	109676	10445	1046	115	18233	31531

id_asset	CYCLO	LOC	NOM	NOC	NOP	FOUT	CALL
20	26561	110870	10589	1062	119	18478	31931
20	26685	111065	10727	1079	126	18472	31949
20	26854	111938	10756	1081	127	18585	32229
20	26864	111963	10761	1081	127	18589	32237
20	27172	113306	10960	1110	128	18811	32641
20	27267	113857	11002	1114	129	18906	32823
20	28749	123478	11557	1115	129	19354	33948
20	29483	126516	11833	1150	129	19842	34959
20	30322	131135	12415	1206	136	20256	36101
20	28015	120276	11986	988	132	18990	33154
20	30751	132227	12848	1253	132	20556	36186
20	29308	124920	12462	1027	132	19699	34129
20	43622	253333	22953	2399	135	53050	97943
20	41927	282691	23070	2190	134	54266	99851
20	45921	302435	24518	2448	136	57708	107677
20	44200	286050	23989	2411	135	54909	103237
20	44281	286868	24023	2413	135	55044	103500
21	11877	53787	6661	697	83	14284	22022
21	12034	54499	6729	702	84	14395	22245
21	12067	54634	6729	701	84	14426	22294
21	12067	54634	6729	701	84	14426	22294
21	17639	80789	9691	966	100	21140	32627
21	17679	80970	9720	967	101	21184	32696
21	17797	81340	9811	970	101	21326	32867
21	20708	94181	11513	1181	128	24548	37913
21	20727	94258	11522	1181	128	24565	37942
21	20712	94188	11520	1179	128	24564	37937
22	69298	493317	35117	3262	481	52479	91559
22	69174	422955	35135	3257	480	52504	91560
22	69180	493045	35135	3257	480	52510	91576
22	69210	493188	35143	3258	480	52540	91625

id_asset	CYCLO	LOC	NOM	NOC	NOP	FOUT	CALL
22	70771	501949	36573	3276	482	53181	93548
22	70804	501983	36565	3275	482	53169	93566
22	70818	502103	36568	3275	482	53179	93580
22	70949	503219	36631	3275	482	53309	93775
22	71520	507039	36869	3282	483	54278	94570
22	71975	510821	36986	3297	483	53914	94862
22	72186	512408	37088	3300	483	54079	95119
22	72675	516239	37257	3299	483	55589	96142
22	71873	505550	36734	3250	474	53566	94296
22	71972	506498	36774	3252	474	53654	94448
22	74344	523815	37594	3277	477	55137	97383
22	74353	523873	37595	3277	477	55142	97392
22	75111	530976	37846	3287	478	56732	98637
22	75650	536418	38133	3301	478	57196	99467
22	76219	542513	38338	3315	481	57427	100409
22	76233	542619	38342	3315	481	57430	100422
22	77041	550871	38679	3340	486	57953	101386
22	78565	562578	39316	3379	488	59016	103201
22	79894	572998	39955	3438	493	60508	105097
22	80557	581794	40146	3445	488	61104	106231
22	81239	588723	40397	3459	489	61631	107179
22	82206	598242	40709	3477	490	62277	108381
22	83490	610621	41268	3507	494	63285	110050
22	84784	624158	41655	3529	495	64291	111940
23	2735	19419	1582	112	7	3795	6682
23	3007	21106	1704	115	7	4019	7241
23	3269	22926	1820	126	7	4357	7754
23	3271	22938	1820	126	7	4359	7759
23	3372	23572	1848	127	7	4476	7975
23	3372	23574	1848	127	7	4475	7974
23	3826	27769	2040	146	7	5162	9149

id_asset	CYCLO	LOC	NOM	NOC	NOP	FOUT	CALL
23	5324	37189	2760	174	10	6742	12065
23	5566	38477	2820	181	11	6774	12203
23	5821	39849	2905	189	11	6897	12605
23	5968	41006	2962	191	11	7099	12976
23	6499	44585	3145	197	11	7752	14213
23	6526	44983	3145	197	11	7906	14330
23	6675	45636	3192	197	11	7980	14514
23	7466	51464	3537	205	11	8910	16301
23	7843	53163	3611	208	11	9072	16797
23	7914	53466	3629	208	11	9153	16921
23	7855	53360	3599	208	11	9184	16808
23	8491	54072	3763	183	10	9040	16502
23	8581	54505	3783	185	10	9121	16770
23	8881	60176	3975	211	11	10087	18688
23	8892	60239	3976	211	11	10093	18696
23	9753	65388	4261	219	11	11323	20339
23	9976	66422	4338	222	11	11500	20717
23	10132	67259	4423	222	11	11685	21031
23	10252	68006	4450	222	11	11798	21260
23	11216	74701	4822	265	11	12872	23192
23	11258	75062	4838	237	11	12943	23309
23	11417	76394	4876	236	11	13053	23573
23	11466	76603	4880	236	11	13056	23590
23	12000	80056	5058	245	11	13597	24703
23	12295	81677	5153	246	11	13822	25163
23	12331	81908	5168	246	11	13862	25240
24	2571	13371	866	42	3	1679	2838
24	5743	33118	1904	124	6	3887	6630
24	6252	36615	2070	140	6	4182	7202
24	11929	71939	4106	272	14	7855	13666
24	11960	72071	4106	272	14	7862	13677

id_asset	CYCLO	LOC	NOM	NOC	NOP	FOUT	CALL
24	14685	87793	5064	309	15	9900	17271
24	14825	88626	5092	310	15	9946	17361
24	28289	164936	9395	507	27	18377	30954
24	30489	178561	9786	520	28	19538	33083
24	30506	178657	9795	520	28	19551	33103
24	30535	178843	9793	520	28	19566	33134
24	30665	179700	9803	520	28	19638	33313
24	30668	179732	9803	520	28	19642	33321
24	30725	179996	9809	520	28	19652	33343
24	31398	182824	9869	521	28	19869	33833
24	31494	183277	9904	526	28	19907	33905
24	31563	183600	9941	538	28	19947	33969
24	32350	187715	10216	537	28	20422	34735
24	33412	193515	10576	547	28	21019	35743
24	33441	193615	10581	547	28	21030	35757
24	33513	194113	10605	551	28	21068	35830
25	326	1908	251	21	3	491	719
25	341	1990	263	22	3	503	740
25	309	1874	237	20	3	471	688
25	353	2094	259	21	3	521	769
25	454	2264	313	23	5	499	827
25	462	2270	324	22	5	502	833
25	468	2295	328	23	5	504	831
25	468	2301	329	23	5	507	833
25	469	2304	330	23	5	508	834
25	470	2314	332	23	5	511	837
25	478	2348	336	23	5	516	852
25	478	2347	336	23	5	516	852
25	483	2363	339	23	5	520	859
25	495	2408	342	23	5	527	880
26	382	1493	250	52	4	195	390

id_asset	CYCLO	LOC	NOM	NOC	NOP	FOUT	CALL
26	2625	13006	662	128	8	923	1936
26	2676	13345	715	137	8	1011	2137
26	3660	18468	1004	150	9	1419	2935
26	3855	17277	1138	157	10	1633	3338
26	3901	17403	1168	158	10	1671	3389
26	4694	21357	1395	169	10	2079	4288
26	12115	46690	3052	231	10	3790	8608
26	11978	46199	3029	224	10	3969	8514
26	11978	46209	3029	224	10	3969	8514
27	326	1908	251	21	3	491	719
27	341	1990	263	22	3	503	740
27	309	1874	237	20	3	471	688
27	353	2094	259	21	3	521	769
27	454	2264	313	23	5	499	827
27	462	2270	324	22	5	502	833
27	468	2295	328	23	5	504	831
27	468	2301	329	23	5	507	833
27	469	2304	330	23	5	508	834
27	470	2314	332	23	5	511	837
27	478	2348	336	23	5	516	852
27	478	2347	336	23	5	516	852
27	483	2363	339	23	5	520	859
27	495	2408	342	23	5	527	880
28	2471	9163	905	61	8	1246	2101
28	2091	5433	425	33	5	643	1342
28	2389	6806	576	38	6	811	1604
28	2686	8136	725	42	7	1066	1859
28	2218	8331	767	47	8	1159	1950
28	2662	9816	1026	74	9	1455	2397
28	2396	8718	883	60	8	1227	2048
28	2429	8916	904	61	8	1232	2052

id_asset	CYCLO	LOC	NOM	NOC	NOP	FOUT	CALL
28	2470	9157	905	61	8	1246	2101
28	2471	9163	905	61	8	1246	2101
28	6787	38788	3840	230	8	7382	16250
28	2405	8800	899	61	40	1226	2035
28	6869	39302	3867	234	41	7384	16497
28	6876	39397	3872	234	41	7396	16534
28	7727	40921	3963	247	43	7630	16800
28	7761	41061	3393	250	43	7556	16858
29	2632	8157	1362	140	13	611	995
29	8215	26571	1330	67	6	1350	2547
29	10093	28469	1398	70	7	1540	2824
29	10103	28521	1401	71	7	1534	2822
29	10094	28522	1388	71	7	1544	2838
29	13977	38150	1304	77	7	1581	3296
29	14005	38167	1304	77	7	1581	3308
29	14013	38162	1305	77	7	1583	3288
29	14020	38198	1306	77	7	1587	3298
29	13997	38147	1301	77	7	1582	3292
30	3687	17111	1904	220	24	4075	5999
30	3717	17268	1907	220	24	4108	6072
30	4137	19533	2174	238	24	4658	6904
30	1476	9978	534	37	2	1431	2472
30	5442	24598	2689	261	27	5652	8644
30	2092	11693	965	143	18	2187	3914
30	5793	25733	2779	266	27	5884	9115
30	6153	27597	2928	268	27	6257	9729
30	6561	29402	3090	276	27	6574	10350
30	6593	29539	3118	276	27	6589	10367
30	6584	29541	3118	276	27	6598	10380
30	6595	29544	3118	276	27	6598	10380
30	6660	30639	3211	303	28	6288	10552

id_asset	CYCLO	LOC	NOM	NOC	NOP	FOUT	CALL
30	9194	45220	4368	436	35	8090	13842
30	9191	45225	4365	435	35	8084	13845
30	9197	45250	4366	435	35	8086	13850
30	9197	45252	4366	435	35	8085	13850
31	323	1541	114	10	5	212	360
31	1223	7372	443	35	8	1075	1892
31	1242	7594	451	35	8	1107	1941
31	1242	7594	451	35	8	1106	1940
31	1250	7667	454	35	8	1115	1960
31	1252	7686	454	35	8	1115	1962
31	1355	8320	513	38	9	1276	2215
31	1370	8344	525	39	9	1286	2227
31	1629	9881	678	55	12	1648	2828
31	1630	9885	678	55	12	1648	2831
31	1632	9886	678	55	12	1649	2833
32	3687	17111	1904	220	24	4075	5999
32	3717	17268	1907	220	24	4108	6072
32	4137	19533	2174	238	24	4658	6904
32	1476	9978	534	37	2	1431	2472
32	5442	24598	2689	261	27	5652	8644
32	2092	11693	965	143	18	2187	3914
32	5793	25733	2779	266	27	5884	9115
32	6153	27597	2928	268	27	6257	9729
32	6561	29402	3090	276	27	6574	10350
32	6593	29539	3118	276	27	6589	10367
32	6584	29541	3118	276	27	6598	10380
32	6595	29544	3118	276	27	6598	10380
32	6660	30639	3211	303	28	6288	10552
32	9194	45220	4368	436	35	8090	13842
32	9191	45225	4365	435	35	8084	13845
32	9197	45250	4366	435	35	8086	13850

id_asset	CYCLO	LOC	NOM	NOC	NOP	FOUT	CALL
32	9197	45252	4366	435	35	8085	13850

Anexo III

Lista de Software Analizado

Id_asset	Nombre	Descripción	Cantidad de Versiones
1	iText	Librería para gestionar formato PDF	17
2	Apache PDF Box	Bilbioteca de manipulación de PDF	18
3	GNUJpdf	Biblioteca para gestionar PDF	7
4	jpod	Framework de manipulación de archivos PDF	11
5	PDF Clown	Biblioteca de manipulación de PDF	
6	JOpenChart	Biblioteca para incluir gráficas	
7	JfreeChart	Librería para gráficas	20
10	Art of Illusions	Modelador gráfico 3D	28
11	FreeMind	Herramienta para crear mapas mentales	41
12	ICEPdf	Biblioteca para manipulación de PDF	25
13	Hodoku	Generador de Sudoku	12
14	PMD	Analizador de código fuente	15
15	JTS	API que proporciona un modelo de objetos espaciales y funciones fundamentales geométricas 2D	11
16	JFreeSVG	Implementación liviana y rápida de SVG para Java	11
17	lwjgl	Biblioteca liviana para desarrollo de juegos	49
18	SQLeonardo	Herramienta para realizar consultas a bases de datos	16
19	Batik	Librería para gestionar formato SVG	19
20	Chemistry Development Kit	Librería orientada a bioquímica y química	36
21	FreePlane	Herramienta para crear mapas mentales	10
22	Vuze	Cliente Bittorrent	28

23	Sweethome3D	Aplicación para diseño de interiores	33
24	HSQLDB	Motor de base de datos	21
25	DynamicReports	Diseño de informes	48
26	JavaCC	Generador de parsers Compilador-Compilador	10
27	JFig	Solución de configuración de aplicaciones Java	14
28	JDOM	Parser XML	15
29	CSSParser	Parser para hojas de estilo	10
30	jHotDraw	Framework para gráficos estructurados y técnicos	19
31	Plotdigitizer	Digitalización de información gráfica	19
32	jPicEdt	Editor de dibujo vectorial	18

Anexo IV

Términos, expresiones y siglas mencionadas en esta tesis

ACM	Acrónimo de Association for Computing Machinery, sociedad científica y educativa sobre computación, fundada en 1947 en Estados Unidos
Code Smells	En un código fuente, síntoma de un posible problema en el diseño del código o de la aplicación.
Componente de Software	Recursos de software que pueden integrarse a diversos desarrollos y que puede desplegarse de manera independiente
Evolución del Software	El software que modela aspectos de la vida real debe ser adaptado continuamente para seguir cumpliendo la función que cumple
ISO 9126	Estándar interncional referido a la evaluación de la calidad de software.
Mantenibilidad	Factibilidad de identificar una falla y corregirla
Mantenimiento adaptativo	Actividades de modificación del software para hacer frente a cambios en el entorno
Mantenimiento correctivo	Actividades de búsqueda y corrección de errores
Mantenimiento perfectivo	Actividades tendientes a mejorar el desempeño o a incrementar las funcionalidades de un sistema de software
Mantenimiento preventivo	Actividades de modificación orientadas a evitar futuros fallos
Modificabilidad	Atributo que caracteriza el esfuerzo necesario para modificar un sistema de software
Reusabilidad	Atributo que refiere a la facilidad de incorporar un recurso de software en diferentes desarrollos
Reutilización	La utilización de un recurso de software (requerimientos, diseños, código, etc.) en varios proyectos de desarrollo
Reutilización oportunista	Reutilización ocasional, a nivel de proyecto individual, de carácter no sistemática.
Reutilización pragmática	Reutillización de código fuente no desarrollado necesariamente con fines de ser reutilizado; puede realizarse de manera planificada y sistemática.
Reutilización sistemática	Reutilización planificada y formal, con lineamientos y procesos claramente definidos.
Umbral	Valor de referencia para una métrica de software que indica un límite por encima del cual (o por debajo del cual) los resultados serían no deseados, o serían indicativos de posibles problemas en relación con el atributo medido.

Índices de Tablas e ilustraciones

Índice de tablas

Índice de ilustraciones

Referencias

[1] F. P. Brooks, “No Silver Bullet Essence and Accidents of Software Engineering,” *Computer*, vol. 20, pp. 10–19, Apr. 1987.

[2] E. S. de Almeida, A. Alvaro, V. C. Garcia, J. Cordeiro Pires, V. A. Burégio, L. M. Nascimento, D. Lucrédio, and S. Lemos Meira, *C.R.U.I.S.E: Component Reuse in Software Engineering*, 1st ed. Recife: C.E.S.A.R e-book, 2007.

[3] J. Sametinger, *Software engineering with reusable components*. New York, NY, USA: Springer-Verlag New York, Inc., 1997.

[4] M. D. McIlroy, “Mass Produced Software Components,” in *Software Engineering Concepts and Techniques*, vol. 1, P. Naur and Be. Randell, Eds. NATO Science Committee, 1968, pp. 138–150.

[5] W. B. Frakes and S. Isoda, “Success Factors of Systematic Reuse,” *IEEE Softw*, vol. 11, pp. 14–19, Sep. 1994.

[6] M. Morisio, M. Ezran, and C. Tully, “Success and Failure Factors in Software Reuse,” *IEEE Trans Softw Eng*, vol. 28, pp. 340–357, Apr. 2002.

[7] R. Holmes and R. J. Walker, “Systematizing Pragmatic Software Reuse,” *ACM Trans Softw Eng Methodol*, vol. 21, no. 4, pp. 20:1–20:44, Feb. 2013.

[8] J. Ramirez, G. Gil, G. Romero, and L. Gimson, “Indicadores de la utilización del bug traker en proyectos F/OSS,” in *Congreso Argentino de Ciencias de la Computación 2009*, Jujuy, 2009, pp. 713–721.

[9] J. Ramirez, L. Gimson, and G. Gil, “Evaluación de la Evolución del Diseño en F/OSS:un Caso de Estudio,” in *VII Workshop Ingeniería de Software - XVI Congreso Argentino de Ciencias de la Computación*, 2010.

[10] J. Ramirez, G. Gil, and L. Gimson, “El Nacimiento de una Herramienta Educativa Libre,” in *40 Jornadas Argentinas de Informática*, Córdoba, 2011.

[11] J. Ramirez, “Mantenibilidad y Evolutividad en el Software Libre y de Código Abierto,” Trabajo Integrador Especialidad en Ingeniería de Software, Universidad Nacional de La Plata, La Plata, 2011.

[12] J. Ramirez, G. Gil, and M. L. Massé Palermo, “Hacia un catálogo práctico de componentes de F/OSS para la reutilización,” in *41 Jornadas Argentinas de Informática*, La Plata, 2012, pp. 45–52.

[13] J. Ramirez, C. Reyes, and G. Gil, “Métricas de Código fuente y Evolución de F/OSS: un estudio exploratorio,” in *42 Jornadas Argentinas de Informática*, Córdoba, 2013, pp. 198–209.

[14] J. Ramirez, C. Reyes, G. Gil, and F. Durgam, “Evolución y reusabilidad en F/OSS,” in *XVII Workshop de Investigadores en Ciencias de la Computación*, Salta, 2015.

[15] J. Ramirez, G. Gil, and C. Reyes, “Umbrales sugeridos para promedios de métricas de diseño de una aplicación en Java,” 2015.

[16] B. Jalender, A. Govardhan, and P. Premchand, “A pragmatic approach to Software Reuse,” *J. Theor. Appl. Inf. Technol.*, vol. 14, no. 2, pp. 87–96, Jun. 2010.

[17] H. Apperly, “Component-based software engineering,” G. T. Heineman and W. T. Councill, Eds. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001, pp. 513–526.

[18] R. P. Prieto-Díaz, “Status Report: Software Reusability.,” *IEEE Softw.*, vol. 10, no. 3, pp. 61–66, 1993.
[19] R. Pressman, *Ingeniería del Software. Un enfoque Práctico*. México: McGraw-Hill, 2010.
[20] R. Holmes, “Pragmatic software reuse,” University of Calgary, Canadá, 2008.
[21] S. Jansen, S. Brinkkemper, I. Hunink, and C. Demir, “Pragmatic and Opportunistic Reuse in Innovative Start-up Companies,” *IEEE Softw*, vol. 25, no. 6, pp. 42–49, Nov. 2008.
[22] G. Kotonya, S. Lock, and J. Mariani, “Opportunistic reuse: Lessons from scrapheap software development,” in *Component-Based Software Engineering*, Springer, 2008, pp. 302–309.
[23] G. Caldiera and V. R. Basili, “Identifying and Qualifying Reusable Software Components,” *IEEE Comput.*, vol. 24, no. 2, pp. 61–70, 1991.
[24] E. Selim, Y. Ghanam, C. Burns, T. Seyed, and F. Maurer, “A Test-Driven Approach for Extracting Libraries of Reusable Components from Existing Applications,” in *Agile Processes in Software Engineering and Extreme Programming*, Springer, 2011, pp. 238–252.
[25] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.
[26] J. Bhuta, C. A. Mattmann, N. Medvidovic, and B. Boehm, “A Framework for the Assessment and Selection of Software Components and Connectors in COTS-Based Architectures,” in *WICSA ’07: Proceedings of the Sixth Working IEEE/IFIP Conference on Software Architecture*, Washington, DC, USA, 2007, p. 6.
[27] G. Bart, R. Fleurquin, and S. Sadou, “A Component Selection Framework for COTS Libraries,” in *Component-Based Software Engineering*, vol. 5282, M. V. Chaudron, C. Szyperski, and R. Reussner, Eds. Springer Berlin Heidelberg, 2008, pp. 286–301.
[28] J. M. Voas, “Certifying off-the-shelf software components,” *Computer*, vol. 31, no. 6, pp. 53–59, 1998.
[29] C. Wohlin and P. Runeson, “Certification of software components,” *Softw. Eng. IEEE Trans. On*, vol. 20, no. 6, pp. 494–499, 1994.
[30] J. Morris, G. Lee, K. Parker, G. A. Bundell, and C. P. Lam, “Software component certification,” *Computer*, vol. 34, no. 9, pp. 30–36, Sep. 2001.
[31] N. Budhija and S. P. Ahuja, “Review of software reusability,” in *International conference on Computer Science and Information Technology (ICCSIT’2011), Pattaya Dec*, 2011.
[32] J. González-Barahona, J. Seoane Pascual, G. Robles, J. Mas Hernández, and D. Megías Giménez, *Introduction to Free Software*, Tercera. Free technology Academy, 2009.
[33] S. Hissam, C. B. Weinstock, D. Plakosh, and J. Asundi, “Perspectives on Open Source Software,” SEI/CMU, 2001.
[34] F. S. Foundation, *Free Software Definition*. Free Software Foundation, 2009.
[35] O. S. Initiative, *Open Source Definition*. 2006.
[36] R. Stallman, “Viewpoint: Why ‘Open Source’ Misses the Point of Free Software,” *Commun ACM*, vol. 52, no. 6, pp. 31–33, Jun. 2009.
[37] J. Howison and K. Crowston, “The perils and pitfalls of mining SourceForge,” in *Proc. of Mining Software Repositories Workshop at the International Conference on Software Engineering (ICSE)*, Edinburgh, Scotland, 2004.
[38] F. Kon, P. Meirelles, N. Lago, A. S. Terceiro, C. Chavez, and M. G. Mendonça, “Free and Open Source Software Development and Research: Opportunities for Software Engineering,” in *SBES*, 2011, pp. 82–91.
[39] I. Samoladas, I. Stamelos, L. Angelis, and A. Oikonomou, “Open source software

development should strive for even greater code maintainability," *Commun ACM*, vol. 47, no. 4, pp. 83–87, 2004.
[40] C. M. Schweik, "Sustainability in Open Source Software Commons: Lessons Learned from an Empirical Study of SourceForge Projects," *Technol. Innov. Manag. Rev.*, vol. 3, pp. 13–19, Jan. 2013.
[41] J. Ramirez, G. Gil, G. Romero, and L. Gimson, "Indicadores de la Utilización del Bug Tracker en Proyectos F/OSS.," in *XV Congreso Argentino de Ciencias de la Computación*, 2009.
[42] M. Sojer, "Reusing open source code: value creation and value appropriation perspectives on knowledge reuse," Technical University Munich, 2011.
[43] L. Jaccheri and T. Osterlie, "Open Source Software: A Source of Possibilities for Software Engineering Education and Empirical Software Engineering," in *FLOSS '07: Proceedings of the First International Workshop on Emerging Trends in FLOSS Research and Development*, Washington, DC, USA, 2007, p. 5.
[44] G. von Krogh and S. Spaeth, "The Open Source Software Phenomenon: Characteristics That Promote Research," *J Strateg Inf Syst*, vol. 16, no. 3, pp. 236–253, Sep. 2007.
[45] H. Pei-Breivold, M. A. Chauhan, and M. A. Babar, "A Systematic Review of Studies of Open Source Software Evolution," in *17th Asia Pacific Software Engineering Conference (APSEC), IEEE*, 2010.
[46] J. González-Barahora, G. Robles, Gregorio, I. Herraiz, Israel, and F. Ortega, Felipe, "Studying the laws of software evolution in a long-lived FLOSS project," *J. Softw. Evol. Process*, vol. 26, no. 7, pp. 589–612, Jul. 2014.
[47] A. Bauer and M. Pizka, "The Contribution of Free Software to Software Evolution," in *In IEEE International Workshop on Principles of Software Evolution (IWPSE03*, 2003, pp. 170–179.
[48] I. Sommerville, *Ingeniería del Software: un enfoque práctico*. Pearson Educación, 2005.
[49] K. H. Bennett and V. T. Rajlich, "Software maintenance and evolution: a roadmap," in *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, New York, NY, USA, 2000, pp. 73–87.
[50] T. Mens and S. Demeyer, *Software Evolution*. Springer, 2008.
[51] J. Adell and Y. Bernabé, *Software Libre en Educación*. McGraw-Hill, 2007.
[52] P. A. Grubb and A. A. Takang, *Software maintenance - concepts and practice (2. ed.)*. World Scientific, 2003.
[53] J. Martin and C. McClure, *Software Maintenance: The Problem and its Solutions*. London, 1983.
[54] K. Erdil, E. Finn, K. Keating, J. Meattle, S. Park, D. Yoon, and P. Stafford, *Comp180: Software Engineering*. Tufts University, 2003.
[55] B. Lientz, E. Swanson, and G. Tompkins, "Characteristics of application software maintenance," *Commun ACM*, vol. 21, no. 6, pp. 466–471, 1978.
[56] N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*. Boston, MA, USA: PWS Publishing Co., 1998.
[57] M. Fowler and K. Beck, "Bad Smells in Code," in *Refactoring: Improving the design of existing code*, Addison-Wesley, 1999, pp. 75–88.
[58] D. E. Peercy, "A Software Maintainability Evaluation Methodology," *IEEE Trans Softw Eng*, vol. 7, no. 4, pp. 343–351, 1981.

[59] R. Banker, S. Datar, and D. Zweig, “Software complexity and maintainability,” in *ICIS ’89: Proceedings of the tenth international conference on Information Systems*, New York, NY, USA, 1989, pp. 247–255.
[60] M. Kajko-Mattsson, G. Canfora, D. Chiorean, A. van Deursen, T. Ihme, M. M. Lehman, R. Reiger, T. Engel, and J. Wernke, “A Model of Maintainability - Suggestion for Future Research,” in *Software Engineering Research and Practice*, 2006, pp. 436–441.
[61] D. Coleman, D. Ash, B. Lowther, and P. Oman, “Using Metrics to Evaluate Software System Maintainability,” *Computer*, vol. 27, no. 8, pp. 44–49, 1994.
[62] I. Heitlager, T. Kuipers, and J. Visser, “A Practical Model for Measuring Maintainability,” *Qual. Inf. Commun. Technol. Int. Conf. On*, vol. 0, pp. 30–39, 2007.
[63] T. McCabe, “A complexity measure,” in *ICSE ’76: Proceedings of the 2nd international conference on Software engineering*, Los Alamitos, CA, USA, 1976, p. 407.
[64] S. R. Chidamber and C. F. Kemerer, “A Metrics Suite for Object Oriented Design,” *IEEE Trans Softw Eng*, vol. 20, no. 6, pp. 476–493, 1994.
[65] D. I. K. Sjøberg, B. Anda, and A. Mockus, “Questioning Software Maintenance Metrics: A Comparative Case Study,” in *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, New York, NY, USA, 2012, pp. 107–110.
[66] M. J. Munro, “Product Metrics for Automatic Identification of Bad Smell Design Problems in Java Source-Code,” in *METRICS ’05: Proceedings of the 11th IEEE International Software Metrics Symposium*, Washington, DC, USA, 2005, p. 15.
[67] H. Kabaili, R. K. Keller, and F. Lustman, “Cohesion as Changeability Indicator in Object-Oriented Systems,” in *Fifth Conference on Software Maintenance and Reengineering, CSMR 2001, Lisbon, Portugal, March 14-16, 2001*, 2001, pp. 39–46.
[68] Y. Ayalew and K. Mguni, “An Assessment of Changeability of Open Source Software,” *Comput. Inf. Sci.*, vol. 6, no. 3, pp. 68–79, 2013.
[69] L. Rosenberg, “Applying and Interpreting Object Oriented Metrics,” presented at the Software Technology Conference, 1998.
[70] K. Erni and C. Lewerentz, “Applying Design-metrics to Object-oriented Frameworks,” in *Proceedings of the 3rd International Symposium on Software Metrics: From Measurement to Empirical Results*, Washington, DC, USA, 1996, p. 64–.
[71] S. R. Chidamber, D. P. Darcy, and C. F. Kemerer, “Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis,” *IEEE Trans Softw Eng*, vol. 24, no. 8, pp. 629–639, Aug. 1998.
[72] M. Lanza, R. Marinescu, and S. Ducasse, *Object-Oriented Metrics in Practice*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.
[73] K. A. M. Ferreira, M. A. S. Bigonha, R. S. Bigonha, L. F. O. Mendes, and H. C. Almeida, “Identifying Thresholds for Object-oriented Software Metrics,” *J Syst Softw*, vol. 85, no. 2, pp. 244–257, Feb. 2012.
[74] S. Benlarbi, K. E. Emam, N. Goel, and S. N. Rai, “Thresholds for Object-Oriented Measures.,” in *ISSRE*, 2005, pp. 24–39.
[75] R. Shatnawi, W. Li, J. Swain, and T. Newman, “Finding software metrics threshold values using ROC curves,” *J. Softw. Maint. Evol. Res. Pract.*, vol. 22, no. 1, pp. 1–16, 2010.
[76] R. Shatnawi, “A Quantitative Investigation of the Acceptable Risk Levels of Object-Oriented Metrics in Open-Source Systems,” *IEEE Trans. Softw. Eng.*, vol. 99, no. RapidPosts, pp. 216–225, 2010.

[77] S. Kaur, S. Singh, and H. Kaur, “A Quantitative Investigation Of Software Metrics Threshold Values At Acceptable Risk Level,” *Int. J. Eng. Res. Technol.*, vol. 2, no. 3, Mar. 2013.
[78] F. Brito e Abreu and R. Carapuca, “Object-Oriented Software Engineering: Measuring and Controlling the Development Process,” in *Proc. Int’l Conf. Software Quality (QSIC)*, 1994.
[79] R. Bender, “Quantitative Risk Assessment in Epidemiological Studies Investigating Threshold Effects,” vol. 41, no. 3, pp. 305–319, 1999.
[80] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” *Empir. Softw Engg*, vol. 14, no. 2, pp. 131–164, 2009.
[81] G. Robles, “Empirical Software Engineering Research on Libre Software: Data Sources, Methodologies and Results,” Universidad Rey Juan Carlos, 2006.
[82] E. Fernández, O. Dieste, P. Pesado, and R. García-Martinez, “La Importancia del Uso de la Evidencia Empírica en Ingeniería de Software,” in *CACIC 2010*, Buenos Aires, 2010, pp. 206–215.
[83] K. Crowston, K. Wei, J. Howison, and A. Wiggins, “Free/Libre Open-source Software Development: What We Know and What We Do Not Know,” *ACM Comput Surv*, vol. 44, no. 2, pp. 7:1–7:35, Mar. 2008.
[84] L. Gasser and W. Scacchi, “Towards a Global Research Infrastructure for Multidisciplinary Study of Free/Open Source Software Development,” in *Open Source Development, Communities and Quality*, vol. 275, B. Russo, E. Damiani, S. Hissan, B. Lundell, and G. Succi, Eds. Boston: Springer, 2008, pp. 143–158.
[85] S. H. Kan, *Metrics and Models in Software Quality Engineering*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.
[86] R. K. Bandi, V. K. Vaishnavi, and D. E. Turk, “Predicting Maintenance Performance Using Object-Oriented Design Complexity Metrics,” *IEEE Trans Softw Eng*, vol. 29, no. 1, pp. 77–87, 2003.
[87] M. Dagpinar and J. H. Jahnke, “Predicting Maintainability with Object-Oriented Metrics - An Empirical Comparison.,” in *WCRE*, 2004, pp. 155–164.
[88] Gandh, Parul and Bhatia,Pradep, “Reusability Metrics for Object-Oriented System: An Alternative Approach,” *Int. J. Softw. Eng.*, vol. 1, no. 4, pp. 63–72, 2010.
[89] C. Marinescu, R. Marinescu, P. F. Mihancea, D. Ratiu, and R. Wettel, “iPlasma: An Integrated Platform for Quality Assessment of Object-Oriented Design,” in *ICSM (Industrial and Tool Volume)*, 2005, pp. 77–80.
[90] D. C. Ince, L. Hatton, and J. Graham-Cumming, “The case for open computer programs,” *Nature*, vol. 482, no. 7386, pp. 485–488, 2012.
[91] D. Wheeler, “SLOCcount,” 2009. [Online]. Available: http://www.dwheeler.com/sloccount/.
[92] M. M. Lehman and J. F. Ramil, “An approach to a theory of software evolution,” in *IWPSE ’01: Proceedings of the 4th International Workshop on Principles of Software Evolution*, New York, NY, USA, 2001, pp. 70–74.
[93] M. Lehman and J. C. Fernández-Ramil, “Software Evolution,” in *Software Evolution and Feedback*, D. E. P., Juan C Fernández-Ramil Nazim H Madhavji, Ed. 2006, pp. 7–40.
[94] M. Lehman, “Laws of Software Evolution Revisited,” in *EWSPT ’96: Proceedings of the 5th European Workshop on Software Process Technology*, London, UK, 1996, pp. 108–124.
[95] M. W. Godfrey and Q. Tu, “Evolution in Open Source Software: A Case Study,” in *In Proceedings of the International Conference on Software Maintenance*, 2000, pp. 131–142.
[96] A. Capiluppi, M. Morisio, and J. F. Ramil, “Structural Evolution of an Open Source System:

A Case Study," in *IWPC*, 2004, pp. 172–182.
[97] A. Terceiro and C. Chavez, "Structural Complexity Evolution in Free Software Projects: A Case Study," in *Quality and Architectural Concerns in Open Source Software*, 2009.
[98] G. Xie, J. Chen, and I. Neamtiu, "Towards a better understanding of software evolution: An empirical study on open source software.," in *ICSM*, 2009, pp. 51–60.
[99] I. Herraiz, G. Robles, and J. M. Gonzalez-Barahona, "Comparison between SLOCs and number of files as size metrics for software evolution analysis," *2011 15th Eur. Conf. Softw. Maint. Reengineering*, vol. 0, pp. 206–213, 2006.
[100] I. Herraiz, "A statistical examination of the evolution and properties of libre software," in *ICSM*, 2009, pp. 439–442.
[101] F. Shull, J. Singer, and D. I. K. Sjøberg, *Guide to Advanced Empirical Software Engineering*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.
[102] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski, "Metrics and Laws of Software Evolution - The Nineties View," in *Proceedings of the 4th International Symposium on Software Metrics*, Washington, DC, USA, 1997, p. 20–.
[103] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*, 1st ed. Addison-Wesley Professional, 1999.
[104] K. Hashim and E. Key, "A Software Maintainability Attributes Model," *Malysian J. Comput. Sci.*, vol. 9, pp. 92–97, 1996.
[105] J. L. Devore, *Probabilidad y estadística para Ingeniería y ciencias*. CENGAGE Learning, 2005.
[106] S. Herbold, J. Grabowski, and S. Waack, "Calculation and Optimization of Thresholds for Sets of Software Metrics," *Empir. Softw Engg*, vol. 16, no. 6, pp. 812–841, Dec. 2011.
[107] J. Bosch and C. Krueger, *Software reuse methods, techniques, and tools*, vol. 3107. Berlin , New York: Springer, 2004.
[108] D. I. K. Sjoberg, T. Dyba, and M. Jorgensen, "The Future of Empirical Methods in Software Engineering Research," in *2007 Future of Software Engineering*, Washington, DC, USA, 2007, pp. 358–378.
[109] E. Raymond, *The Cathedral and the Bazaar*. Eric Raymond, 2000.
[110] S. Krishnamurthy, "Cave or Community?: An Empirical Examination of 100 Mature Open Source Projects," *SSRN ELibrary*, 2002.
[111] A. Capiluppi and M. Michlmayr, "From the Cathedral to the Bazaar: An Empirical Study of the Lifecycle of Volunteer Community Projects," in *Open Source Development, Adoption and Innovation*, 2007, pp. 31–44.
[112] A. Senyard and M. Michlmayr, "How to Have a Successful Free Software Project," in *Proceedings of the 11th Asia-Pacific Software Engineering Conference*, Busan, Korea, 2004, pp. 84–91.
[113] K. Fogel, *Producing Open Source Software: How to Run a Successful Free Software Project*. O'Reilly Media, Inc., 2005.
[114] W. Scacchi, "Understanding the requirements for developing open source software systems," *IEE Proc. - Softw.*, vol. 149, no. 1, pp. 24–39, 2002.
[115] A. Cechich and M. Piattini Velthuis, "Methods for measurement-based COTS assessments and selection," in *IV Workshop de Investigadores en Ciencias de la Computación*, 2002.
[116] A. Cechich and K. Taryano, "Selecting COTS components: a comparative study on E-Payment systems," in *IX Congreso Argentino de Ciencias de la Computación*, 2003.

[117] S. Romero, E. Santana de Almeida, A. Alexandre, D. Lucredio, and V. Cardoso García, "RiSE Project: Towards Robust Framework for Software Reuse," presented at the Conference on Information Reuse and Integration, 2004. IRI 2004. Proceedings of the 2004 IEEE International, 2004, pp. 48–53.
[118] C. P. Ayala, O. Hauge, R. Conradi, X. Franch, J. Li, and K. S. Velle, "Challenges of the Open Source Component Marketplace in the Industry," in *OSS*, 2009, pp. 213–224.
[119] S. Haefliger, G. von Krogh, and S. Spaeth, "Code Reuse in Open Source Software," *Manage Sci*, vol. 54, no. 1, pp. 180–193, 2008.
[120] L. Heinemann, F. Deissenboeck, M. Gleirscher, B. Hummel, and M. Irlbeck, "On the Extent and Nature of Software Reuse in Open Source Java Projects.," in *ICSR*, 2011, vol. 6727, pp. 207–222.
[121] A. Capiluppi, K.-J. Stol, and C. Boldyreff, "Software Reuse in Open Source A Case Study," *Open Source Softw. Dyn. Process. Appl.*, p. 151, 2013.
[122] J. Díaz, A. Schiavoni, and N. Banchero, "Extendiendo un editor de software libre para soportar contenido SCORM," in *CACIC 2009*, 2009, pp. 479–488.
[123] M. E. Ascheri, G. J. Astudillo, P. García, R. A. Pizarro, and M. E. Culla, "Elaboración de un software educativo usando herramientas gratuitas," in *V Congreso de Tecnología en Educación y Educación en Tecnología*, 2010.
[124] S. Morad and T. Kuflik, "Conventional and open source software reuse at Orbotech-an industrial experience," in *Software-Science, Technology and Engineering, 2005. Proceedings. IEEE International Conference on*, 2005, pp. 110–117.
[125] "Method for Qualification and Selection of Open Source Software (QSOS)." 2013.
[126] I. Samoladas, G. Gousios, D. Spinellis, and I. Stamelos, "The SQO-OSS Quality Model: Measurement Based Open Source Software Evaluation," in *OSS*, 2008, pp. 237–248.
[127] J.-C. Deprez and S. Alexandre, "Comparing Assessment Methodologies for Free/Open Source Software: OpenBRR and QSOS," in *PROFES'08*, 2008, pp. 189–203.
[128] K. Malanga, J. Mehat, I. Ganchev, J. Wandeto, and C. Shikali, "Evaluation of Open Source Software with QualiPSO OMM: a case for Bungeni and AT4AM for All," presented at the Free and Open Source Software Conference FOSSC-15, 2015, pp. 41–46.
[129] J.-C. Deprez, F. F. Monfils, M. Ciolkowski, and M. Soto, "Defining Software Evolvability from a Free/Open-Source Software," *Softw. Evolvability IEEE Int. Workshop On*, vol. 0, pp. 29–35, 2007.
[130] V. R. Basili, "Software modeling and measurement: the Goal/Question/Metric paradigm," Techreport UMIACS TR-92-96, University of Maryland at College Park, College Park, MD, USA, 1992.
[131] A. Wasserman, M. Pal, and C. Chan, "The Business Readiness Rating Model: an Evaluation Framework for Open Source," Como, Italy, 2006.
[132] B. Golden, "Making open source ready for the enterprise: The open source maturity model," *Open Source Bus. Resour.*, no. May 2008, 2008.
[133] B. M. Goel and P. K. Bhatia, "Analysis of Reusability of Object-oriented Systems Using Object-oriented Metrics," *SIGSOFT Softw Eng Notes*, vol. 38, no. 4, pp. 1–5, Jul. 2013.
[134] E. Weyuker, "Evaluating Software Complexity Measures," *IEEE Trans. Softw. Eng.*, vol. 14, pp. 1357–1365, 1988.
[135] Fazal-e-Amin, A. K. Mahmood, and A. Oxley, "Reusability Assessment of Open Source Components for Software Product Lines," *Int. Conf. Comput. Inf. Sci. ICCIS*, vol. 3, 2012.

[136] J. J. Marshall and R. R. Downs, “Reuse Readiness Levels as a Measure of Software Reusability,” in *Geoscience and Remote Sensing Symposium, 2008. IGARSS 2008. IEEE International*, 2008, vol. 3.
[137] C. Ncube and N. A. Maiden, “PORE: Procurement-oriented requirements engineering method for the component-based systems engineering development paradigm,” in *International Workshop on Component-Based Software Engineering*, 1999, pp. 130–140.
[138] J. Merilinna and M. Matinlassi, “State of the Art and Practice of OpenSource Component Integration.,” in *EUROMICRO-SEAA*, 2007, pp. 170–177.
[139] P. Loucopoulos and V. Karakostas, *System requirements engineering*. McGraw-Hill, Inc., 1995.
[140] P. Martins, J. P. Fernandes, and J. Saraiva, “A web portal for the certification of open source software,” in *Information Technology and Open Source: Applications for Education, Innovation, and Sustainability*, Springer, 2014, pp. 244–260.
[141] S. Henry and D. Kafura, “Software Structure Metrics Based on Information Flow,” *IEEE Trans Softw Eng*, vol. 7, no. 5, pp. 510–518, 1981.
[142] C. Lee, “JavaNCSS-a source measurement suite for Java, 2005,” *URL Httpjavancss Codehaus Org*.
[143] S. Thakur and H. Singh, “FDRD: Feature driven reuse development process model,” in *Advanced Communication Control and Computing Technologies (ICACCCT), 2014 International Conference on*, 2014, pp. 1593–1598.
[144] S. Singh and I. Chana, “Enabling Reusability in Agile Software Development,” *CoRR*, vol. abs/1210.2506, 2012.
[145] D. Turk, R. France, and B. Rumpe, “Limitations of agile software processes,” *ArXiv Prepr. ArXiv14096600*, 2014.
[146] T. J. Gandomani, H. Zulzalil, A. A. A. Ghani, and A. B. M. Sultan, “A Systematic Literature Review on relationship between agile methods and Open Source Software Development methodology,” *CoRR*, vol. abs/1302.2748, 2013.
[147] B. Russo, *Agile technologies in open source development*. IGI Global, 2009.