Evaluación de la Evolución del Diseño en F/OSS:un Caso de Estudio

Jorge Ramirez, Loraine Gimson, Gustavo Gil

Centro de Investigación y Desarrollo en Informática Aplicada (CIDIA),
Universidad Nacional de Salta.
{ramirezj, loraine, gdgil}@cidia.unsa.edu.ar

Abstract: En los últimos años El F/OSS se convirtió en una alternativa viable tanto para la provisión de software como para la reutilización en la creación de nuevos productos. Dada la enorme heterogeneidad de los productos F/OSS, se vuelve necesaria la búsqueda de métodos para evaluar estos productos. Nos proponemos aquí explorar la información que puede obtenerse de un producto F/OSS utilizando una serie de métricas que permitan sacar conclusiones respecto de la evolución del código fuente. Tomamos como caso de estudio la aplicación Sweet Home 3D, una aplicación para diseño de interiores escrita en Java y alojada en el repositorio SourceForge, Realizamos las mediciones y el procesamiento estadístico mediante herramientas de libre disponibilidad, a fin de facilitar la replicación del estudio o comparar con estudios similars. Posteriormente efectuamos el análisis de los datos obtenidos, observando una correlación lineal entre las variables consideradas para los valores registrados en este caso. Esto nos permitirá a futuro poder cotejar los resultados con experiencias similares.

Palabras clave: Software Libre y de CódigoAbierto, métricas, evolución, caso de estudio

1.- Introducción

El Software Libre y de Código Abierto (F/OSS, por sus siglas en inglés) alcanzó una amplia difusión en los últimos años, convirtiéndose en una alternativa viable tanto para la provisión de software como para la reutilización en la creación de nuevos productos[1].

La enorme heterogeneidad de los productos F/OSS, tanto en lo que respecta a la calidad como a los procesos de desarrollo, vuelven necesaria la búsqueda de métodos para evaluar estos productos.

El presente trabajo se orienta a la evaluación de F/OSS con miras a la reutilización del código, sea mediante la adaptación a circunstancias particular o como componentes de nuevos desarrollos.

Uno de los aspectos más relevantes para ese fin, es la mantenibilidad, entendida como la facilidad de realizar modificaciones sobre el software.

Nos proponemos aquí explorar la información que puede obtenerse de un producto F/OSS a partir de la información disponible sobre el mismo, centrándonos en la disponibilidad del código fuente y utilizando una serie de métricas que permitan sacar conclusiones respecto de la evolución del código fuente.

Las métricas seleccionadas apuntan a caracterizar el diseño de una aplicación escrita en Java en lo referente a tamaño y complejidad, y el acoplamiento del sistema. Nos orientamos a una aplicación Orientada a Objetos, basándonos en la posibilidad de obtener información sobre el diseño de la aplicación a partir del análisis del código fuente.

Mediante el análisis de los datos buscaremos elementos que nos permitan realizar afirmaciones sobre la manera en que evolucionó la aplicación analizada, con miras a cotejar los resultados con experiencias similares.

Específicamente, nos proponemos observar:

- 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.

Las dos primeras cuestiones se vinculan con leyes II, VI y VII de la Evolución de los sistemas y postuladas por Lehman [2][3] según la cuales los sistemas de software tienden con el tiempo a incrementar su complejidad, aumentar sus prestaciones y a declinar su calidad.

El resto de este trabajo se organiza de la siguiente forma: en la segunda sección repasamos una serie de estudios vinculados con la evolución del software y los proyectos de F/OSS, destacando especialmente aquellos que contemplan aspectos estructurales y se refieren a desarrollos orientados a objetos. En la tercera sección detallamos la metodología seguida en este estudio; en la sección 4, presentamos diferentes vistas de los datos obtenidos; a continuación analizamos dichos datos y finalmente exponemos nuestras conclusiones y los futuros trabajos a realizar.

2.- Evolución del Software y F/OSS

El F/OSS se caracteriza por la disponibilidad pública del código fuente; con frecuencia, también se puede acceder libremente a información abundante sobre el desarrollo.

Estas propiedades alentaron un número creciente de investigaciones sobre el desarrollo F/OSS, cubriendo aspectos técnicos o tecnológicos, económicos y sociales[4]. En particular, la disponibilidad pública de información favoreció el desarrollo de investigaciones empírica en el área de la Ingeniería de Software[5][6]. Stol y otros [7] realizaron una revisión sistemática de los trabajos sobre F/OSS, detectando las temáticas más comunes abordadas en este tipo de estudios, los métodos y técnicas de investigación utilizados, así como el crecimiento de trabajos expuestos en las conferencias de código abierto (OSS Conferences) entre los años 2005 y 2008.

Uno de los aspectos que concitó mayor atención por parte de diversos investigadores es la evolución del software de este tipo, en particular las similitudes y diferencias respecto del software desarrollado por métodos "tradicionales" o de código cerrado[8][9][10].

Algunas investigaciones realizadas sobre productos F/OSS de gran tamaño sugieren que la evolución de este tipo de productos -en lo que se refiere al tamaño de la aplicación en función del tiempo- sigue en ocasiones tasas super-lineales[11][12], aunque la mayoría parece seguir una tendencia de crecimiento lineal [13], medida en términos de líneas de código. Algunos autores consideran que las altas tasas de crecimiento y la complejidad creciente constituyen una amenaza para la mantenibilidad del F/OSS.

Otros autores trabajaron sobre la evolución de F/OSS de menor escala, en la búsqueda de características y comportamientos evolutivos que pudieran compararse y que condujeran eventualmente a propuestas más generales. Los trabajos de Lee y otros[14] analizando la librería JFreeChart, y de Terceiro y Chavez[15] sobre el proyecto Ristretto, centraron su interés en la evolución del acoplamiento en sucesivas versiones de las aplicaciones analizadas. Capiluppi[16] estudió un conjunto de 15 programas de este tipo de tamaños pequeño a mediano buscando modelos de evolución, considerando el tamaño, los módulos y los desarrolladores intervinientes en cada proyecto; este autor observó en varios casos una tendencia de crecimiento lineal, advirtiendo también que frecuentemente los módulos incrementan su tamaño de manera asintótica hacia un valor tope, al menos en los casos estudiados.

Una dificultad que aparece al momento de intentar comparar los resultados obtenidos en los diferentes estudios es la diversidad de técnicas y medidas utilizadas. La medición del tamaño tampoco muestra una coincidencia absoluta, pero las métricas utilizadas (Líneas de Código, líneas de código no comentadas, líneas de código físicas, cantidad de módulos, número de clases) frecuentemente están relacionadas entre sí o presentan correlaciones significativas. Los estudios sobre las relaciones entre complejidad, crecimiento y evolución apelan a medidas muy dispares que aún esperan consenso más firme de la comunidad académica.

En el ámbito específico de las aplicaciones orientadas a objetos, el análisis de la evolución estructural presentan mayores similitudes. Aquí cabe considerar los intentos de validar diferentes métricas estructurales en relación con la mantenibilidad[17], algunos de los cuales utilizaron F/OSS como casos de estudio[18].

Evaluación del diseño

Lanza y Marinescu[19] proponen la caracterización del diseño de una aplicación desarrollada bajo el paradigma de orientación a objetos mediante una serie de proporciones entre métricas. Las medidas utilizadas son las siguientes:

CYCLO: complejidad ciclomática, según la definición de McCabe[20]
LOC: Líneas de código, incluyendo líneas en blanco y comentarios.
NOM: número de operaciones. Contabiliza todos los métodos y operaciones definidas en la aplicación.
NOC: Número de clases
NOP: Número de paquetes
CALLS: Número de llamadas a operaciones. Cuenta la cantidad total de invocaciones a operaciones en todo el proyecto.
FANOUT: Cantidad de clases llamadas
Lanza y Marinescu utilizan una serie de proporciones entre estas métricas, con las caracterizan diferentes "estructuraciones" relativas al tamaño y la complejidad del sistema de software:

- Estructuración de alto nivel (NOC/NOP)
- Estructuración de clases (NOM/NOC)
- Estructuración de operaciones (LOC/NOM)
- Complejidad de operación intrínseca (CYCLO/LOC)

El acoplamiento del sistema se refleja mediante otras dos proporciones:

- Intensidad de acoplamiento (CALLS/NOM)
- Dispersión del acoplamiento (FANOUT/CALLS)

3.- Metodología

Para el presente trabajo descargamos todas las versiones publicadas de un producto F/OSS desde su versión 1.0 hasta la más reciente designada como 2.4. La secuencia de versiones abarca más de dos años de desarrollo.

Para cada una de ellas obtuvimos la serie de métricas mencionadas más arriba y calculamos las proporciones referidas por Lanza y Marinescu como "Estructuración de alto nivel", "Estructuración de clases", "Estructuración de operaciones" y "Complejidad de operación intrínseca". Adicionalmente, consideramos otra medida del tamaño basado en el código fuente, la cantidad de líneas de código físicas, que descuenta las líneas en blanco y comentadas.

A continuación, observamos la evolución de esas métricas a lo largo del tiempo, prestando atención a posibles tendencias y evaluando posibles correlaciones entre las versiones, el tamaño y las proporciones que caracterizan (en este modelo) a las cuatro estructuraciones mencionadas.

Aplicación estudiada

Tomamos como caso de estudio la aplicación Sweet Home 3D, una aplicación para diseño de interiores escrita en Java y alojada en el repositorio SourceForge (http://sourceforge.net/projects/sweethome3d/)

Elegimos esta aplicación por la disponibilidad pública del código fuente desde antes incluso de su versión 1.0 hasta la versión 2.4, las que abarcan un período que va de diciembre de 2007 a mayo de 2010. Puede observarse en el repositorio mencionado que este proyecto continúa desarrollándose regularmente.

Otro aspecto que favoreció la adopción de este caso es el lenguaje de desarrollo, que facilita extraer conclusiones referidas al diseño mediante herramientas disponibles libremente.

Herramientas utilizadas

En el presente trabajo, las mediciones y el trabajo estadístico se realizaron mediante herramientas de acceso libre, de modo de facilitar la replicación independiente de este trabajo, y llevar adelante otros similares cuyos resultados sean comparables.

Las mediciones se realizaron mediante la aplicación iPlasma, un entorno integrado para el análisis de calidad de sistemas de software orientados a objetos. La aplicación está disponible en http://loose.upt.ro/iplasma/index.html.

Esta herramienta ofrece también consideraciones generales sobre las características generales del diseño, remarcando los valores de las proporciones que podrían sugerir problemas de diseño. Estas indicaciones se basan en “umbrales” determinados estadísticamente por Lanza y Marinescu[19].

Cabe señalar que la herramienta en cuestión también se utiliza en entornos industriales para el seguimiento y administración de las actividades de aseguramiento de calidad.

El cómputo de líneas de código físicas se realizó con SLOCCount, una aplicación desarrollada por David Wheeler ampliamente utilizada y probada[5][21][22], que puede descargarse de http://www.dwheeler.com/sloccount/.

El tratamiento estadístico se realizó con una planilla de cálculo OpenOffice Calc.

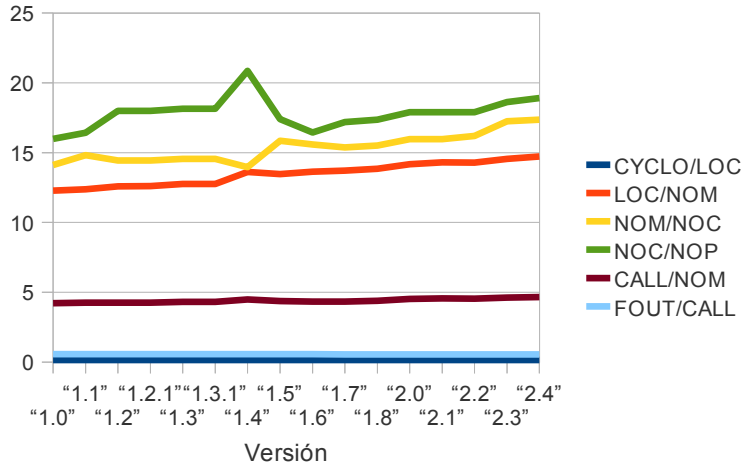
4.- Datos obtenidos

Por razones de espacio, no presentamos la totalidad de los valores obtenidos y calculados. En la tabla 1 mostramos los valores obtenidos para las proporciones sugeridas para evaluar el tamaño y la complejidad del sistema.

Versión	Complejidad operacional intrínseca	Estructuración de operación	Estructuración de clases	Estructuración de alto nivel
“1.0”	0,14	12,27	14,13	16
“1.1”	0,14	12,39	14,82	16,43
“1.2”	0,14	12,6	14,44	18
“1.2.1”	0,14	12,6	14,44	18
“1.3”	0,14	12,76	14,55	18,14
“1.3.1”	0,14	12,76	14,55	18,14
“1.4”	0,14	13,61	13,97	20,86
“1.5”	0,14	13,47	15,86	17,4
“1.6”	0,14	13,64	15,58	16,45
“1.7”	0,15	13,72	15,37	17,18
“1.8”	0,15	13,84	15,51	17,36
“2.0”	0,15	14,18	15,96	17,91
“2.1”	0,15	14,3	15,96	17,91
“2.2”	0,15	14,3	16,2	17,91
“2.3”	0,15	14,55	17,25	18,64
“2.4”	0,15	14,72	17,36	18,91

Tabla 1: valores computados con la herramienta iPlasma

En la gráfica 1 podemos observar el crecimiento de las distintas proporciones recopiladas respecto de la versión de la aplicación.



Grráfica 1: Evolución de las proporciones computadas en diferentes versiones, ordenadas cronológicamente

5.- Análisis de los datos

Observamos que todas las proporciones muestran una suave tendencia a crecer en el tiempo. Algunas de ellas, especialmente las relacionadas con el acoplamiento a nivel del sistema, muestran un comportamiento casi constante.

La complejidad de operación intrínseca también permanece casi estable, advirtiéndose un leve incremento a partir de la versión 1.8

Los datos también sugieren 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.

La gráfica muestra un pequeño salto en la estructuración de alto nivel en la versión 1.4, indicando un incremento de la cantidad de clases por paquete. A partir de la versión siguiente, esta proporción recupera la tendencia de crecimiento lineal anterior.

A pesar de esa variación, el coeficiente de Pearson obtenido a partir de las líneas de código y la estructuración de alto nivel es de 0,93, lo que sugiere una correlación lineal entre las variables consideradas para los valores registrados en este caso.

La herramienta iPlasma ofrece una valoración de estas características, considerando como potenciales problemas de diseño la alta cantidad de métodos que presentan las clases, así como la longitud de los métodos. En la versión 1.4, la herramienta destacaba que el software tiende a organizarse en paquetes demasiado grandes; esta advertencia no se remarca en otras versiones.

5.-Conclusiones y trabajos futuros

El presente trabajo es de carácter exploratorio, y se orienta a detectar posibles indicadores de la evolución de un producto F/OSS a partir de los datos extraídos de su código fuente.

La información relevada sugiere que en el caso estudiado el desarrollo tiende al crecimiento lineal, con un incremento de la complejidad verificable y de comportamiento lineal, pero con una pendiente que indica que tal crecimiento es poco pronunciado a lo largo del tiempo.

La modificación que se observa luego de la versión 1.4 respecto de la estructuración de alto nivel, según la definen Lanza y Marinescu, sugiere que en el proyecto se han tomado medidas tendientes a reducir posibles problemas de diseño.

En casi dos años y medio de desarrollo, el proyecto exhibe un comportamiento bastante estable en cuanto al crecimiento en tamaño y complejidad.

Entendemos que la información que ofrece este conjunto de medidas brinda un primer panorama respecto de las características de la evolución de este producto en particular; no obstante, sería necesario vincular esta información con otros aspectos del desarrollo (desempeño respecto de la corrección de errores, cantidad y

organización de los desarrolladores, evolución en cuanto al número de descargas, etc.)

Por otra parte, será preciso realizar otros estudios similares que permitan apreciar los aspectos específicos y las posibles enseñanzas que estos casos pueden ofrecer para otros proyectos F/OSS y para la evaluación de los mismos en función de las características del diseño y del desarrollo.

Referencias:

1. Hissam, S.A.; Weinstock, C.B.. Open Source Software: The Other Commercial Software. In *In 1st Workshop on Open Source Software at ICSE*. (2001)

2. Lehman, M.M.; Ramil, J.F.. An approach to a theory of software evolution. In *IWPSE '01: Proceedings of the 4th International Workshop on Principles of Software Evolution*. (2001)

3. Lehman, M.; Fernández-Ramil, J.C.: *Software Evolution*, en *Software Evolution and Feedback*, . (2006)

4. Amant, K.; Still, B.: *Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives*. .IGI Global(2007)

5. Robles, G.:Empirical Software Engineering Research on Libre Software: Data Sources, Methodologies and Results. Universidad Rey Juan Carlos (2006)

6. Jaccheri, L.; Osterlie, T.. 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*. (2007)

7. Stol, K.; Babar,M.A.; Russo,B.; Fitzgerald, B.. The use of empirical methods in Open Source Software research: Facts, trends and future directions. In *FLOSS '09: Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*. (2009)

8. Scacchi, W.. Understanding Open Source Software Evolution: Applying, Breaking, and Rethinking the Laws of Software Evolution. In *Applying, Breaking and Rethinking the Laws of Software Evolution*. (2003)

9. Wu, J.:Open Source Software Evolution and Its Dynamics. University of Waterloo (2006)

10. Capiluppi, A.; Morisio,M.; Ramil, J.F.. Structural Evolution of an Open Source System: A Case Study. In *IWPC*. (2004)

11. Godfrey, M.W.; Tu, Q.. Evolution in Open Source Software: A Case Study. In *In Proceedings of the International Conference on Software Maintenance*. (2000)
12. Koch, S.:Software evolution in open source projects---a large-scale investigation.*J. Softw. Maint. Evol.*, 19 (2007)
13. Robles, G.; Amor,J.J.; Gonzalez-Barahona,J.M.; Herraiz, I.. Evolution and Growth in Large Libre Software Projects. In *IWPSE '05: Proceedings of the Eighth International Workshop on Principles of Software Evolution*. (2005)
14. Lee, Y.; Yang,J.; Chang, K.H.. Metrics and Evolution in Open Source Software. In *QSIC '07: Proceedings of the Seventh International Conference on Quality Software*. (2007)
15. Terceiro, A.; Chavez, C.. Structural Complexity Evolution in Free Software Projects: A Case Study. In *Quality and Architectural Concerns in Open Source Software* . (2009)
16. Capiluppi, A.. Models for the evolution of OS projects. In *ICSM '03: Proceedings of the International Conference on Software Maintenance*. (2003)
17. Bandi, R.K.; Vaishnavi,V.K.; Turk, D.E.:Predicting Maintenance Performance Using Object-Oriented Design Complexity Metrics.*IEEE Trans. Softw. Eng.*, 29 (2003)
18. Olague, H.M.; Etzkorn,L.H.; Gholston,S.; Quattlebaum, S.:Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes.*IEEE Trans. Softw. Eng.*, 33 (2007)
19. Lanza, M.; Marinescu,R.; Ducasse, S.: *Object-Oriented Metrics in Practice*. .Springer-Verlag New York, Inc.(2005)
20. McCabe, T.. A complexity measure. In *ICSE '76: Proceedings of the 2nd international conference on Software engineering*. (1976)
21. Herraiz, I.; González-Barahona,J.M.; Robles,G.; Germán, D.M.. On the prediction of the evolution of libre software projects. In *ICSM*. (2007)
22. Wheeler, D.:How to Evaluate Open Source Software/Free Software (OSS/FS) Programs. http://www.dwheeler.com/oss_fs_eval.html.