

UNIVERSIDAD
NACIONAL
DE COLOMBIA

Facultad de Ingeniería

ShoggothViz

Herramienta para la visualización evolutiva de código fuente de proyectos hospedados en repositorios abiertos de software

Por

Joan Sebastian Lopez Riaño

Ingeniero de Sistemas - Universidad Nacional de Colombia
Especialista en Multimedia - Universidad de los Andes

Documento presentado para la Maestría de Ingeniería de Sistemas y Computación

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ

Noviembre 2016

CONTENIDO

1. INTRODUCCIÓN.....	3
2. MARCO CONCEPTUAL Y ESTADO DEL ARTE.....	4
2.1 Marco Conceptual.....	4
2.1.1 Código Fuente.....	4
2.1.2 Software Libre.....	4
2.1.3 Categoría F.L.O.S.S.....	5
2.1.4 Comunidades del Software Libre y Código Abierto.....	5
2.1.5 Naturaleza evolutiva del Software.....	6
2.1.6 Minería de Repositorios de Software.....	7
2.1.7 Visualización de Software.....	7
2.1.8 Data Art.....	8
2.1.9 Métricas para proyectos FLOSS.....	9
2.2 Estado del Arte.....	11
2.2.3 Proyectos más cercanos a la propuesta de <i>ShoggothViz</i>	11
2.2.3.1 Malwarez.....	11
2.2.3.2 Code_Swarm.....	11
2.2.3.3 Gource.....	12
2.2.3.4 Software Feathers.....	12
2.2.3.5 Codeology.....	13
2.3 Conclusiones del estado del arte.....	14
3. DEFINICIÓN Y JUSTIFICACIÓN DEL PROBLEMA.....	15
4. OBJETIVOS.....	16
5. METODOLOGÍA.....	17
6. BIBLIOGRAFÍA.....	19

1. INTRODUCCIÓN

Partiendo de la idea de aportar nuevas metáforas en el campo de la visualización de código fuente y la relación que este campo puede tener con las artes visuales, se presenta el proyecto ***ShoggothViz***¹, una herramienta de software complementada con una experiencia interactiva (instalación artística) que permita acercar a la audiencia al entorno del desarrollo y evolución de los proyectos de Software Libre (o de Código Abierto) y que permita “dar un rostro” a artefactos compuestos por código que también pueden tener una representación estética y funcional.

Se implementaran distintas métricas para el análisis de código fuente así como también se aplicara una minería de repositorios de software para ofrecer una experiencia no solo visual de las formas generadas a partir del análisis sintáctico del código sino que también se pueda visualizar la evolución de la pieza de software que se recibe como entrada en un orden cronológico dando cuenta de la relación entre la calidad y la participación de las comunidades a lo largo del tiempo de vida del mismo.

Palabras clave: Visualización de código, Minería de repositorios de software, Software Libre, Comunidades, Arte Interactivo, Metáforas en visualización, Gource, Codeology, Code_Swarm.

1. El nombre ShoggothViz, es un juego de palabras para resumir la propuesta de metáfora *Shoggoth Visualization*. Los Shoggoth son criaturas ficticias monstruosas y amorfas creadas por la imaginación del autor estadounidense de horror Howard P. Lovecraft(1890-1937).

2. MARCO CONCEPTUAL Y ESTADO DEL ARTE

2.1 Marco Conceptual

2.1.1 Código Fuente

En la actualidad el uso del software es algo generalizado en la sociedad, sin embargo muchas personas ignoran que el proceso por el cual se construyen las piezas de software es en realidad un proceso creativo que combina el pensamiento abstracto con la escritura.

Por código fuente se entiende la serie de instrucciones escritas con un sentido algorítmico que sirven para indicar a un computador qué operaciones hacer en un orden específico. Dependiendo de la época y la tecnología, el código fuente es escrito siguiendo una sintaxis previamente definida, también conocida como *lenguaje de programación*. Ejemplos de lenguajes de programación son : *Kobold*, *C++*, *Java*, *Python*, etc.

El código fuente es la parte esencial de toda pieza de software pues indica como está hecho y la manera en que procesa las entradas y salidas con que fue diseñada. Sus símiles mas cercanos en la cotidianidad podrían ser los planos de una construcción, o el plano esquemático de una tarjeta electrónica.

2.1.2 Software Libre

Una definición aproximada de la expresión *Software Libre* se refiere a la categoría que agrupa los productos de software que garantizan las libertades esenciales de quienes los usan para cualquier propósito , así como también propenden por acceso sin restricciones al código fuente del programa y a la posterior distribución tanto de ejecutables como de copias del mismo.

La expresión “Software Libre”, donde el adjetivo *Libre* denota la condición de libertad, fue acuñada por la Free Software Foundation (FSF) en cabeza de su principal representante Richard M. Stallman en el año 1985 en U.S.A y hace hincapié en la diferencia del término en inglés *free* en referencia a la libertad como concepto con la definición mercantil que solo se limita a acotar el precio de un producto.

En sus orígenes la FSF se planteó proteger jurídicamente aquellos programas de Software Libre mediante la implementación de un acuerdo entre quienes desarrollaban el software y quienes lo usasen, a este documento se le conoce usualmente como licencia de uso. Para tales efectos la FSF creó la *General Public Licence* (GPL por sus siglas en inglés) la cual establece los pilares éticos y filosóficos del Movimiento por el Software Libre y que en su esencia garantiza las cuatro libertades básicas enunciadas por Richard Stallman[1]:

0. *Libertad para ejecutar el programa con cualquier propósito.*
1. *Libertad para estudiar el programa y adaptarlo*
2. *Libertad de redistribución de copias del programa.*
3. *Libertad para modificar el programa y publicar tales mejoras.*

Como eje fundamental de la filosofía con que fue concebida la GPL está el acceso al código fuente del programa. El libre acceso al código fuente permite garantizar las libertades 1 y 3 pero por sí mismo no garantiza que una pieza de software pueda ser considerada como *Software Libre* en la definición de la FSF pues obstáculos sobre las libertades 0 y 2 restringen las libertades de uso.

En el año 1998 surge en U.S.A la “Iniciativa del Código Abierto” u *Open Source Initiative*[2] por sus siglas en inglés. Dicha iniciativa surgió con la filosofía de promover una metodología en la industria del desarrollo de software donde se permitiera el libre acceso al código fuente bajo la premisa de que así se crearían productos de software más eficientes y potentes.

Desde que surgió la *Open Source Initiative* se inició un intenso debate entre los defensores de la ahora denominada comunidad *Open Source* y la FSF, puesto que esta última invitaba a quienes quisieran desarrollar software libre a desligarse de la expresión *Open Source*, puesto que la filosofía del Software Libre iba mas allá de una metodología en el desarrollo del software y se constituía mas como un movimiento social por las libertades del uso de la tecnología y el acceso al conocimiento, algo de lo cual quienes defendían la practica industrial del *Open Source* no estaban muy preocupados.

El trasfondo de la discusión gira en torno a la idea del “para que” o “para quienes” se produce el software. Respecto este debate dice Dávila[3]: “*La Open Source Initiative hace énfasis en la producción de mejoras al software para crear nuevos productos beneficiando mas a los desarrolladores que a los usuarios, en contraposición del Software Libre de Richard Stallman quien desde la Free Software Foundation promueve la libertad en el uso para beneficio de los usuarios*”.

2.1.3 Categoría F.L.O.S.S

Años después de iniciado el debate entre la FSF y la *Open Source Initiative* se planteó la creación de una expresión “neutral” para referirse a los proyectos de software que en general permiten el acceso al código fuente a y a realizar modificaciones y distribución libre del software. Se acuñó el acrónimo FLOSS que resume la expresión anglosajona *Free/Libre and Open Source Software*, la cual sirve para categorizar pero que en palabras de Richard Stallman no contribuye a resolver el debate pues “...el código abierto es una metodología de programación, el Software Libre es un movimiento social”[4].

2.1.4 Comunidades del Software Libre y Código Abierto

Desde los inicios de múltiples proyectos de software dentro de la categoría FLOSS y con el advenimiento de la Internet como medio para compartir el código y a su vez publicar modificaciones se configuraron las primeras comunidades virtuales.

Las comunidades del software pueden ser entendidas no solo como espacios de encuentro en la Internet (foros, listas de correos, redes sociales, etc) sino también como “*agrupaciones de individualidades, organizaciones y empresas que contribuyen en el mejoramiento de un bien publico (en este caso un producto de software) a través de compartir libremente sus innovaciones (también dudas o criticas) con otros miembros de la comunidad o fuera de ella*”. [5]

Cabe resaltar que aunque en un principio las comunidades de proyectos FLOSS estaban conformadas en su mayoría por personas provenientes de entornos académicos o industriales de las ciencias de la computación y el desarrollo de software, su evolución ha permitido que hoy en día la participación en las mismas esté abierta a cualquiera que desde sus capacidades y conocimientos pueda aportar, pues adicionalmente a las tareas del desarrollo, son necesarios esfuerzos de traducción, edición de manuales, diseño gráfico y soporte.

Gracias a investigaciones realizadas como la de Baytiyeh y Pfaffman[6] se pueden caracterizar las comunidades FLOSS como espacios donde la participación se da de forma libre, voluntaria y con una perspectiva global.

Se podría concluir que en su mayoría quienes participan de estas comunidades lo hacen por motivaciones mas de tipo altruistas orientadas al bien común y que comparten entre si la visión de una sociedad utópica donde el conocimiento se comparta libremente, sin embargo también hay una variedad de otras motivaciones menos altruistas mas ligadas a los sectores industriales o académicos. Se podría jerarquizar entonces las motivaciones de quienes participan de estas comunidades como:

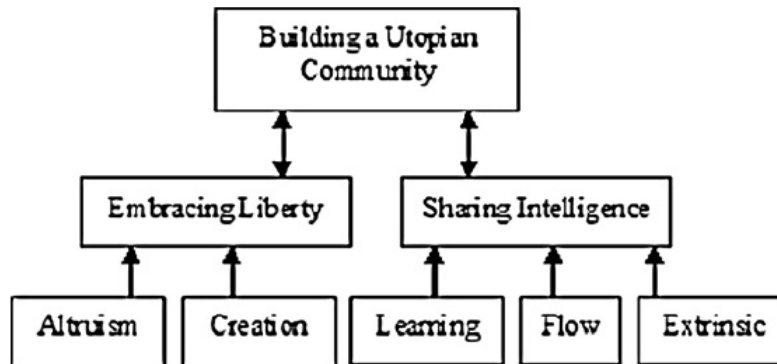


Fig 1. Motivaciones para participar en proyectos FLOSS – BAYTIYEH, et al.

Tamaño de la comunidad y calidad del software

Dado que la naturaleza propia de los proyectos de código abierto promueve el compartir conocimiento podría asumirse que mientras mayor sea el tamaño de la comunidad, mayor sera la calidad del producto final y menor serán sus tiempos de desarrollo. Sin embargo propuestas como el SQO-OSS² definen modelos donde la calidad del producto de software y la comunidad que lo soporta se evalúan de forma separada, extrayendo métricas para cada uno de estos componentes para finalmente emitir juicios de valor (excelente, bueno, pasable, pobre) sobre cada componente a evaluar.[7]

Sitios web como *OpenHub* permiten llevar una trazabilidad de las comunidades FLOSS mas activas, por ejemplo *Mozilla Firefox*, *Apache HTTP Server*, *.NET*, *MySQL*, entre otras.[8]

2.1.5 Naturaleza evolutiva del Software

Desde los inicios de la computación moderna y la producción del software los procesos mediante los cuales se creaban programas informáticos fueron importados desde áreas del conocimiento como la Ingeniería Civil. Iniciando con modelo de desarrollo en cascada propuesto en 1970 y mas adelante pasando por los modelos cíclicos iterativos se llegó a acuñar el termino *Software Evolution* en los años noventa del siglo XX. Finalmente, en el año 2001 fue gracias a la llegada de las *metodologías ágiles de desarrollo*[9] que se analizó con rigurosidad académica el fenómeno de la naturaleza evolutiva del software, diferenciando esta característica de las recurrentes tareas de mantenibilidad al final del proceso de desarrollo.

2 . *Software Quality Observatory for Open Source Software* (SQO-OSS por sus siglas en inglés)

En términos prácticos la expresión *Software Evolution* sirve para referirse al conjunto de tareas más comunes en la producción de software relacionadas con el mantenimiento en todas las etapas de la producción, esto incluye la corrección de errores, inclusión de mejoras y nuevas funcionalidades, optimización y adaptabilidad a nuevas tecnologías.

Dado que estas tareas siempre tenían que ser consideradas en cualquier proyecto se llegó a la conclusión de que el software en esencia es de naturaleza evolutiva y no estática como se creía en los inicios de la Ingeniería de Software. Las implicaciones de esta naturaleza evolutiva justificarían la necesidad de proveer visualizaciones de código que abarquen mas de un instante de tiempo pues los cambios que tuviesen los proyectos también deberían poder ser visualizados en una escala temporal[10]. En la actualidad la evolución de un proyecto de software, en especial de código abierto puede ser analizada gracias a sistemas de versionamiento como *Subversion*, *Git*, *Mercurial*, *CVS*[12], y plataformas colaborativas tales como *GitHub*, *SourceForge*, *RedMine*, entre otras[12].

2.1.6 Minería de Repositorios de Software

Los repositorios de software (públicos) son grandes colecciones de proyectos de software de código abierto los cuales proveen de un hospedaje en la web para todos los artefactos que puede comprender una pieza de software (código fuente, recursos, manuales, traducciones, etc) así como también provee de un sistema de control de versiones que permite llevar un seguimiento a la evolución histórica del proyecto así como para también realizar *forks* o duplicados del mismo (cuando el proyecto lo permite). La minería de repositorios de software se refiere a las técnicas usualmente aplicadas a repositorios públicos de software tales como *SourceForge*³ o *GitHub*⁴, que permiten realizar extraer información valiosa de grandes conjuntos de datos como lo pueden ser: el código fuente, la concurrencia de usuarios activos en el proyecto, sistemas de gestión de *bugs*⁵ de programa, listas de correo, entre otras. En conclusión la minería de repositorios de software se encarga de realizar tareas de ingeniería a grandes cantidades de datos relacionadas con proyectos de software para extraer información de valor a quienes la consulten.[13]

2.1.7 Visualización de Software

La visualización de software comprende el área de la ingeniería de software que se encarga de desarrollar herramientas para entender mejor el diseño y funcionalidad de un programa usando como base el análisis a profundidad de artefactos como el código fuente (principalmente), la meta-data extraída de repositorios de software, entre otros.

Por lo general las herramientas desarrolladas para apoyar el entendimiento del código fuente mediante la visualización de software se sustentan en la aplicación de metáforas visuales que permitan a quienes desarrollan los productos extraer información útil para la corrección de errores en el diseño o la prevención temprana de futuras fallas. Dichas metáforas pueden ser muy amplias y orientadas a metas diferentes del análisis. Por ejemplo una de las metáforas mas conocidas ha sido la de visualización de código fuente como ciudades propuesta por KESKIN en 1997[14] y que ha sido objeto de un amplio estudio en años recientes[15]

3 . <https://sourceforge.net>

4 . <https://github.com/>

5 . Errores de programa – *bugs*. La expresión anglosajona se acuño en el año 1967 al encontrarse presencia de insectos en la computadora Mark II.

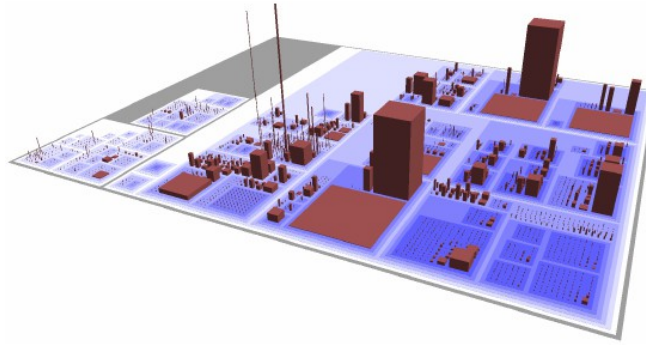


Fig 2. Ciudad de ArgoUML – [WETTEL, et al -2007]

Revisiones sistemáticas de la literatura respecto al uso y variedad de estas metáforas como la desarrollada por XU(et al)[16] permiten ver que aunque se agrupan en algunas categorías el mayor porcentaje de estas metáforas (28%) suele ser significativamente diferente a todas las demás.

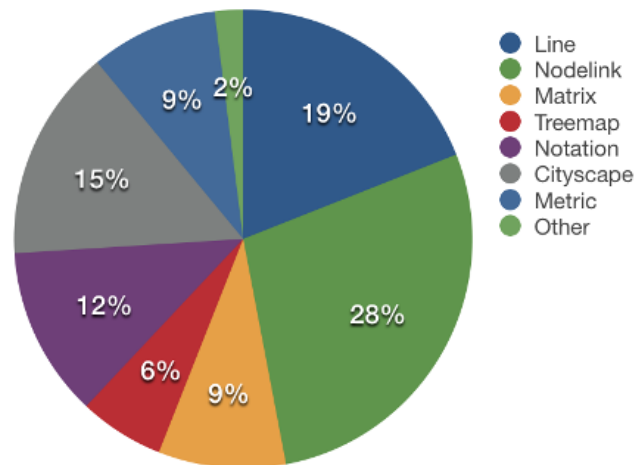


Fig 3. Número de estudios por metáfora [XU, et al – 2015]

2.1.8 Data Art

Data Art se refiere a una rama del denominado *Software Art* el cual se caracteriza porque las obras son desarrolladas usando lenguajes de programación e implican una relación directa de quien produce el arte y el uso de la tecnología en su nivel más puro (código fuente). De hecho el proceso mediante el cual se crea el software o el código fuente en si mismo puede considerarse una obra de arte[17]. El *Data Art* o *Data Driven Art* agrupa los trabajos artísticos donde se usan aproximaciones creativas para visualizar grandes cantidades de información que gracias a la actual abundancia de sistemas de información pueden ser consultadas desde cualquier parte del mundo[18]. El potencial del *Data Art* reside en la amplia gama de posibilidades que ofrece la re-interpretación de información que por si misma, sin ser clasificada, estructurada o visualizada no presentaría mayor interés para una audiencia cada vez más ávida de experiencias visuales de rápida asimilación. Respecto a lo anterior dice McGarrigle[19]: “... El potencial del *Data Art* reside en su habilidad para representar información de distintas maneras donde las conexiones se hacen evidentes, presentando la información como narrativa y revelando las estructuras y patrones subyacentes”.

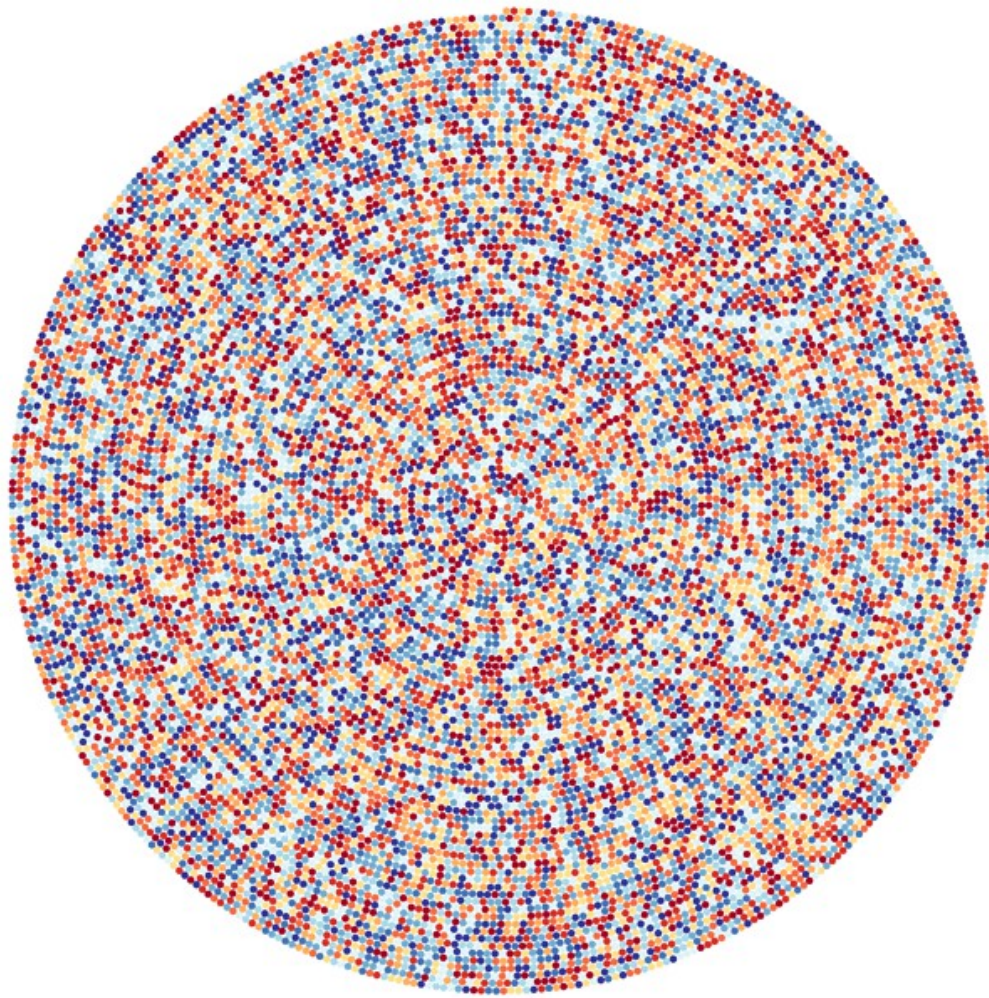


Fig 4. Distribución de los primeros 13,689 dígitos de π por M.Krzywinski[20]

2.1.9 Métricas para proyectos FLOSS

Métricas en las comunidades web

Un análisis de métricas para comunidades de proyectos FLOSS propuesto por el profesor Español Jesus M. González en 2015 [21] abarca las siguientes cinco métricas que permitirían establecer un diagnóstico para proyectos de código abierto:

- **Actividad**
Se refiere a la medición de la participación activa de la comunidad en el proyecto, no solo representada en aportes (modificaciones) al mismo sino en el reporte de errores o proposición de mejoras.
- **Tamaño**
El valor a lo largo del tiempo de la evolución respecto a la cantidad de contribuyentes (activos) en la comunidad
- **Desempeño**
La tasa de resolución o finalización de tareas pendientes, corrección de errores, tiempos de respuesta, etc.
- **Demografía**
Se refiere a la tasa de personas que generación por generación del proyecto se mantienen vigentes junto a la tasa nuevas personas atraídas a participar en el mismo.

- **Diversidad**

Participación en un proyecto diferenciada entre personas naturales y organizaciones u compañías interesadas en el mismo, así como su tasa de compromiso determinada por el “*Factor pony*”, el cual indica el Número mínimo de desarrolladores realizando 50% de las mejoras.

Métricas del código fuente

Gracias al aporte de la tesis doctoral de Paulo R. Miranda [22] se podrían generalizar la siguientes métricas del código fuente aplicadas a proyectos FLOSS:

Métricas de tamaño		
Abreviatura	Descripción (en Inglés)	Descripción (Castellano)
LOC	<i>Lines Of Code</i>	Líneas de código
AMLOC	<i>Average Method Lines Of Code</i>	Promedio de líneas de código por método
-	<i>Total Number of Modules or Classes</i>	Cantidad total de módulos o clases
Métricas de mantenibilidad, flexibilidad y entendimiento de código		
Abreviatura	Descripción (en Inglés)	Descripción (Castellano)
NOA	<i>Number of Attributes</i>	Número de atributos por clase
NOM	<i>Number of Methods</i>	Número de métodos por clase
NPA	<i>Number of Public Attributes</i>	Número de atributos públicos
NPM	<i>Number of Public Methods</i>	Número de métodos públicos
ANPM	<i>Average Number of Parameters per Method</i>	Número promedio de parámetros por método
DIT	<i>Depth of Inheritance Tree</i>	Profundidad del árbol de herencia entre clases
NOC	<i>Number of Children</i>	Número de hijos por clase
RFC	<i>Response for a Class</i>	Respuestas para una clase
ACCM	<i>Average Cyclomatic Complexity per Method</i>	Promedio de complejidad ciclomática por método

Métricas de acoplamiento entre clases		
Abreviatura	Descripción (en Inglés)	Descripción (Castellano)
ACC	<i>Afferent Connections per Class</i>	Conexiones aferentes de una clase
CBO	<i>Coupling Between Objects</i>	Acoplamiento entre objetos
COF	<i>Coupling Factor</i>	Factor de acoplamiento
LCOM	<i>Lack of Cohesion in Methods</i>	Ausencia de conexión entre métodos
SC	<i>Structural Complexity</i>	Complejidad Estructural

Tabla 1. Métricas aplicadas al código fuente

2.2 Estado del Arte

2.2.3 Proyectos más cercanos a la propuesta de *Shoggoth Viz*

Al establecer una clasificación acotada a proyectos de corte artístico o de las ramas de la ingeniería de software con aproximaciones artísticas al análisis de código fuente de proyectos o piezas de software se destacan un conjunto de proyectos que se describirán a continuación.

2.2.3.1 Malwarez

Es un proyecto del artista Alex Dragulescu[23], que provee de una serie de visualizaciones a programas tales como gusanos, virus, troyanos y software espía entre otros. El artista implementó distintas métricas dentro de las cuales destacan :llamadas de API's, direcciones en memoria y subrutinas. Finalmente empleaba la información extraída para generar una visualización 3D que transformaba estas líneas de código en “organismos artificiales”.

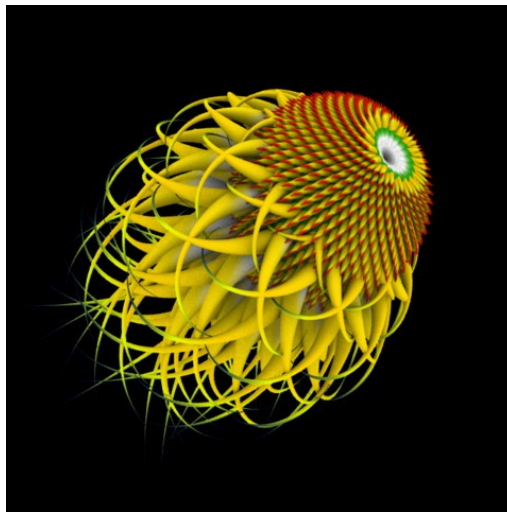


Fig 8. Troyano MSNThreat – Código Visual Basic -2009

2.2.3.2 Code_Swarm

Nacido en el año 2009 *code_swarm* [24] se convirtió en un referente para los proyectos de visualización en la evolución del historial de modificaciones hechas a un proyecto de software. Dicha visualización presenta la metáfora de un universo en expansión y de estrellas que representan los momentos donde alguien colaboró.[25]

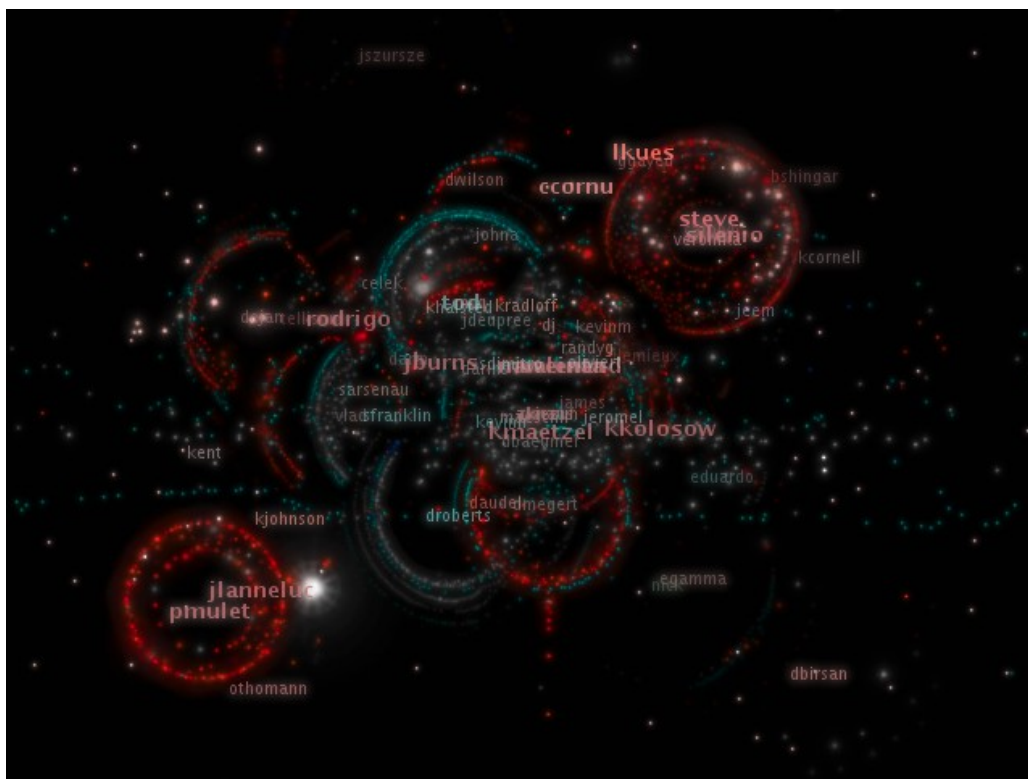


Fig 9. Visualización de *code swarm* del proyecto Eclipse

2.2.3.3 Gource

Nacido en el año 2014 el proyecto Gource[26] ha tomado una gran popularidad gracias a que no solo ha sido implementado para el análisis de proyectos FLOSS sino también para proyectos de código privativo. La propuesta de Gource se basa en la construcción de un video elaborado a partir de una visualización de la historia de un repositorio de código usando nodos y arboles.

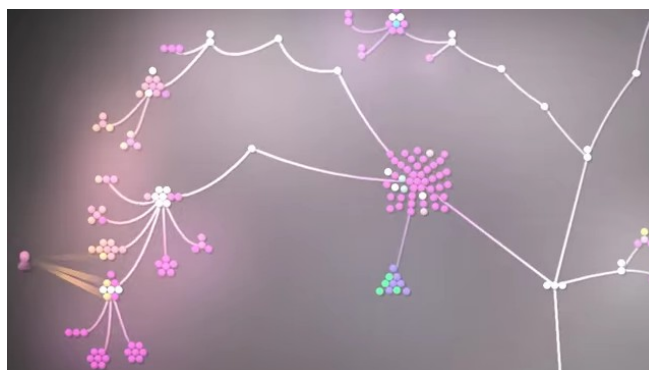


Fig 10. Visualización de *Gource*

2.2.3.4 Software Feathers

Desarrollado por Fabian Beck[27] en 2014 se constituye como una aproximación al mapeo de características de artefactos de código fuente relacionándolas con características de plumas orgánicas, pero va mas allá pues este tipo de metáfora visual aporta soluciones para la detección de posibles errores en el código y para el estudio de la evolución del mismo.

Table 3: Evolution of code entities.




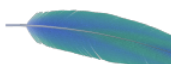


JHotDraw 7.0.6	JHotDraw 7.6
ClearRecentFilesAction	ClearRecentFilesMenuAction
	
ColorIcon	
	
AbstractApplicationAction	
	

Fig 11. Visualización del proyecto *JhotDraw* usando Software Feathers

2.2.3.5 Codeology

Publicado en 2015 [28] consiste en analizar proyectos FLOSS hospedados en la plataforma GitHub y con dicha información crea formas orgánicas únicas por cada proyecto (en su totalidad, no por artefactos).

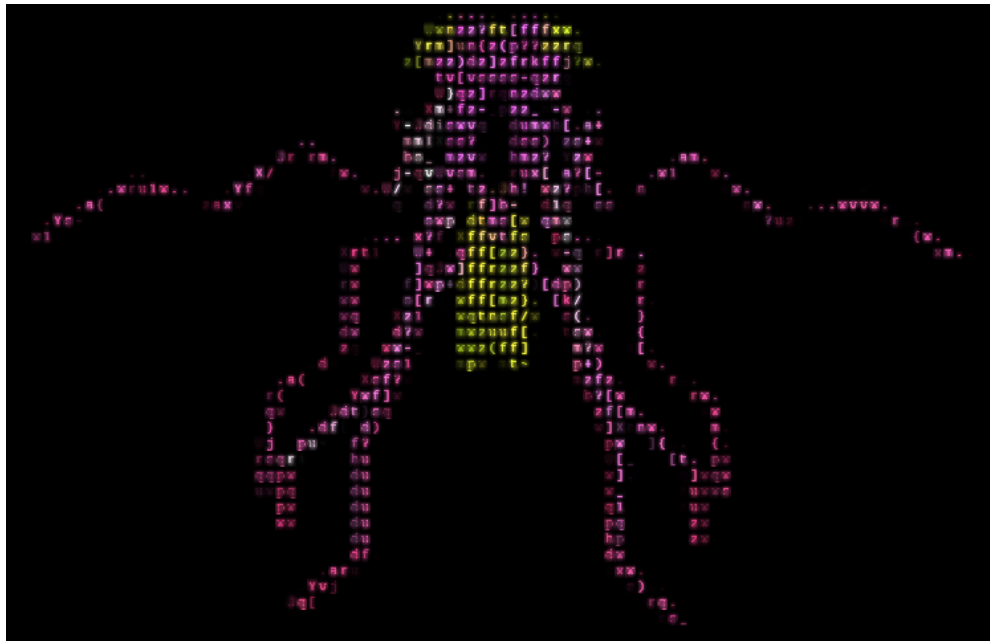


Fig 11. Visualización del proyecto *nighmare* con Codeology

Las métricas que implementa *Codeology* están basadas en la longitud de los artefactos de código y en el lenguaje de los mismos. Tiene como ventajas que es un proyecto para la web, lo cual le permite llegar a una gran cantidad de personas pues no requiere instalaciones específicas, y solo se necesita de un explorador moderno. La principal característica de este proyecto es su conexión con la plataforma colaborativa *GitHub*, ya que permite buscar por usuarios o proyectos y la carga de las visualizaciones se realiza de forma rápida y bien presentada.

Dentro de las tecnologías usadas están *WebGL* (para la implementación en exploradores) y *Three.js* (para el procesamiento de la información extraída de *GitHub*). De igual manera el proyecto es de código abierto y esta hospedado en *GitHub*, disponible para que cualquiera pueda descargarlo o modificarlo.

2.3 Conclusiones del estado del arte

1. Existe una brecha de funcionalidades, entre las propuestas de *Software Feathers*, *Gource* y *Codeology*, que justificaría la realización de *ShoggothViz* al proveer de una metáfora distinta para las visualizaciones, apoyada en las métricas definidas por *Software Feathers*, pero que también suministre una vista para hacerse a una idea de la evolución del proyecto en el tiempo, así como en *Gource*.
2. No existen muchas propuestas que permitan a través de una instalación artística acercar a una audiencia a las problemáticas de las comunidades de proyectos FLOSS y sensibilizar acerca de la importancia en su apropiamiento y participación.
3. Sería interesante realizar ejercicios de evaluación respecto a los efectos que tienen sobre las personas las metáforas de visualización de código fuente para evaluar si realmente dichas visualizaciones tienen un impacto en la forma como se entiende el proceso de desarrollo de una pieza de software y las contribuciones de una comunidad a lo largo del tiempo.

3. DEFINICIÓN Y JUSTIFICACIÓN DEL PROBLEMA

Dentro de los proyectos que en la actualidad abordan metáforas alternativas a las tradicionales en el campo de la visualización de código fuente (aplicado en proyectos de Software Libre y Open Source) existen aun brechas creativas donde nuevas herramientas como la que se quiere desarrollar tienen cabida.

Se puede entonces identificar un área de exploración donde el proyecto propuesto puede aportar. Esta área se ubica precisamente entre la relevancia de la metáfora visual del código fuente representado como un organismo en 3D, combinada con la metáfora de la evolución cronológica de dicho organismo (con origen de datos en repositorios de software).

Se justifican entonces propuestas visuales que vayan mas allá de una “fotografía” en un instante del tiempo, expandidas con una funcionalidad que permita ver la evolución que han tenido los artefactos de código y la participación de las comunidades para que de esta manera se pueda entender a la pieza de software como un proceso evolutivo, que al igual que un ser vivo que nace, crece se desarrolla y muere.

La aplicación propuesta buscar relacionar distintas áreas del análisis de programas, como lo son la visualización de métricas de código fuente y la minería de repositorios abiertos con el propósito de permitir al usuario obtener información en tiempo real de los siguientes tópicos (obtenidos a partir de las visualizaciones):

- Detección de problemas en la calidad del código (*bad smells*).
- Visualización del crecimiento y evolución de los distintos artefactos que componen los proyectos.
- Identificación de la participación y del impacto de las personas que han hecho contribuciones a las aplicaciones analizadas.

4. OBJETIVOS

4.1 Objetivo general

Desarrollar una metodología que permita visualizar y abstraer diferentes métricas del código fuente de un grupo delimitado de proyectos de software pertenecientes a la categoría FLOSS⁶, implementando una metáfora para la representación de las características y la evolución en el tiempo de cada artefacto analizado.

4.2 Objetivos específicos

1. Investigar el estado del arte de trabajos relacionados con las metáforas figurativas de código fuente y sus relaciones con los avances en el mejoramiento de la calidad del software.
2. Definir la metáfora a implementar y el conjunto de métricas a extraer de los artefactos de código fuente hospedados en repositorios abiertos o plataformas colaborativas.
3. Desarrollar una aplicación de software que dados un conjunto de artefactos de código fuente (clases, *scripts*, etc) genere una visualización que represente las distintas métricas a analizar de dichos elementos.
4. Validar la aplicación desarrollada mediante el análisis de algunos proyectos tipo FLOSS, que servirán de casos de estudio, comparando la metodología con otras similares encontradas en el estado del arte.

6. “Free/Libre and Open Source” (F.L.O.S.S, por sus siglas en inglés)

5. METODOLOGÍA

1. Estrategia General

Para la efectiva implementación del proyecto se optará por una metodología del tipo experimental, es decir, que se validara la propuesta mediante pruebas de control (experimentos) y comparaciones respecto a los antecedentes encontrados en el estado del arte. Para llevar a cabo lo definido en el objetivo principal (desarrollar e implementar una metodología para la visualización de métricas aplicadas al software) se definirán un conjunto de por lo menos tres propuestas de metáforas figurativas las cuales serán validadas mediante la construcción de una aplicación de software por cada una. Finalmente se espera que con los productos de cada propuesta (visualizaciones) poder realizar un estudio comparativo respecto a otras metáforas de visualización encontradas en la literatura, así como también se contempla la realización de un experimento controlado con la metáfora definitiva.

Los cuatro objetivos específicos se repartirán en tres fases de evolución del proyecto, las cuales se pueden resumir como : Definición de la metáfora figurativa de visualización, implementación de la metodología y finalmente validación de la misma.

2. Fases del proyecto

Fase I: Definición de la metáfora y las métricas de la visualización

Esta fase inicial abarca los objetivos específicos No.1 y No.2. Representa el esfuerzo inicial por comprender el estado del arte en el tema de visualizaciones figurativas de software, el estudio de la evolución del software y las distintas métricas a analizar, clasificándolas según su relevancia y/o utilidad en las etapas del desarrollo de un proyecto de código abierto. Después de llegar a unas conclusiones respecto al estado actual del campo de estudio se planea explorar por lo menos tres propuestas para la metáfora final a seleccionar, escogiendo aquella que cuente con las bases mas solidas que justifiquen su usabilidad y pertinencia. De igual manera se definirá el conjunto de métricas a extraer tanto de los artefactos de código fuente como de los repositorios abiertos de un conjunto delimitado de proyectos *open source*.

Esta fase será la mas importante pues con la construcción de una muy buena justificación para la metáfora definitiva se podrán obtener mejores resultados de los posteriores experimentos controlados.

Fase II: Implementación de la metodología en proyectos de muestra

Después definir la metáfora y las métricas que comprenderán la propuesta, se procederá a la implementación de dicha metodología mediante la construcción de una aplicación que tenga como objetivo generar las visualizaciones sobre los artefactos de código de un proyecto *open source* y su respectivo repositorio o plataforma colaborativa.

Para la realización del prototipo se utilizara la metodología de desarrollo SCRUM simplificada, la cual, aplicada a un equipo unipersonal con *sprints* o iteraciones semanales permitirá agilizar la generación cada semana de nuevas visualizaciones que se acerquen mas al producto deseado.

Fase III: Validación de la metodología

Para esta fase final se planea realizar un análisis experimental que tenga como objetivo sustentar la justificación de la metáfora en comparación con otras propuestas encontradas en el estado del arte, explorando sus posibles aplicaciones y el aspecto de las distintas etapas de la construcción de un proyecto de software para las cuales para la cual pueda ser mas relevante.

Se contempla la realización de un experimento controlado para retro-alimentar la propuesta y que permita validar las principales características de la visualización así como sus potenciales debilidades.

Fase IV (Final):

Esta fase se considera la etapa de divulgación pues contempla la publicación de un artículo científico que resuma el proceso de la desarrollo de la propuesta , así como los resultados obtenidos del experimento controlado y que exponga algunos ejemplos de la visualización aplicada.

3. Actividades por fase

A continuación se presentan las actividades y productos por cada objetivo específico:

Fase I: Definición de la metáfora y las métricas de la visualización	
OBJETIVO	<i>Investigar el estado del arte de trabajos relacionados con las metáforas figurativas de código fuente y sus relaciones con los avances en el mejoramiento de la calidad del software.</i>
ACTIVIDADES	<ol style="list-style-type: none"> 1. Definir un marco teórico mínimo para las temáticas de visualización de software. 2. Realizar un filtrado inicial de referencias recientes en el campo de la visualización de software 3. Realizar un segundo filtrado de referencias para seleccionar los proyectos relacionados con metáforas del código fuente 4. Elaborar las conclusiones del estado del arte
PRODUCTOS	<ol style="list-style-type: none"> 1. Documento del marco teórico y estado del arte del proyecto
OBJETIVO	<i>Definir la metáfora a implementar y el conjunto de métricas a extraer de los artefactos de código fuente hospedados en repositorios abiertos o plataformas colaborativas.</i>
ACTIVIDADES	<ol style="list-style-type: none"> 1. Escoger y clasificar un conjunto inicial de métricas de software de acuerdo a las conclusiones del estado del arte. 2. Escoger y clasificar un conjunto inicial de métricas de los repositorios de código abierto de acuerdo a las conclusiones del estado del arte. 3. Proponer y desarrollar por lo menos tres ideas para la metáfora de representación de las métricas anteriormente seleccionadas. 4. Diseñar ilustraciones de ejemplo sobre las visualizaciones propuestas aplicadas a métricas extraídas de algunos artefactos de código fuente.
PRODUCTOS	<ol style="list-style-type: none"> 1. Documento de la descripción de la metáfora a implementar. 2. Documento de la descripción de las otras posibles metáforas a implementar. 3. Presentación de la propuesta de la metáfora principal, utilidad y aplicaciones.

Fase II: Implementación de la metodología en proyectos de muestra

OBJETIVO	<i>Desarrollar una aplicación de software que dados un conjunto de artefactos de código fuente (clases, scripts, etc) genere una visualización que represente las distintas métricas a analizar de dichos elementos.</i>
ACTIVIDADES	<ol style="list-style-type: none"> 1. Documentar los casos de uso y diagramas de flujo de la aplicación a desarrollar. 2. Escoger la tecnología a implementar en la propuesta de visualización y conexiones con los repositorios de código fuente. 3. Diseñar la arquitectura y el modelado de información para la generación de las visualizaciones. 4. Implementar el diseño (desarrollo de la aplicación).
PRODUCTOS	<ol style="list-style-type: none"> 1. Prototipo inicial de la aplicación que implemente la metodología propuesta.

Fase III: Validación de la metodología	
OBJETIVO	<i>Validar la aplicación desarrollada mediante el análisis de algunos proyectos tipo FLOSS, que servirán de casos de estudio, comparando la metodología con otras similares encontradas en el estado del arte y la realización de un experimento controlado</i>
ACTIVIDADES	<ol style="list-style-type: none"> 1. Seleccionar un grupo de tres proyectos como casos de estudio 2. Generar por cada proyecto una visualización (de ser posible) usando alguna de las metodologías cercanas (identificadas en el estado del arte). 3. Generar por cada proyecto una visualización usando la metodología propuesta. 4. Comparar los resultados obtenidos por las metodologías de referencia y la metodología propuesta. 5. Generar un listado de conclusiones respecto a la comparación del punto anterior.
PRODUCTOS	<ol style="list-style-type: none"> 1. Documento con la descripción del proceso de validación y de conclusiones respecto a posibles usos y aplicaciones de la metodología propuesta. 2. Documento que resuma las hipótesis y resultados del experimento controlado.

Fase IV: Divulgación	
ACTIVIDADES	<ol style="list-style-type: none"> 1. Redacción de un artículo científico para la difusión de la propuesta.
PRODUCTOS	<ol style="list-style-type: none"> 1. Artículo científico para enviar a varias conferencias del tema en <i>Software Visualization</i> o <i>Software Evolution</i>

3. Cronograma

Se distribuirá la realización del proyecto a lo largo de un (1) año de trabajo completo distribuido de la siguiente manera:

Fase I

Duración: Dos (2) meses

Fecha de inicio: Enero 2018

Fecha de finalización: Marzo 2018

Fase II

Duración: Tres (3) meses

Fecha de inicio: Marzo 2018

Fecha de finalización: Junio 2018

Fase III

Duración: Tres (3) meses

Fecha de inicio: Junio 2018

Fecha de finalización: Septiembre 2018

Fase IV

Duración: Un (1) mes

Fecha de inicio: Noviembre 2018

Fecha de finalización: Diciembre 2018

Fig.1 : Cronograma de trabajo

	2018	ENE	FEB	MAR	ABR	MAY	JUN	JUL	AGO	SEP	OCT	NOV	DIC
FASE I	Definir un marco teórico mínimo para las temáticas de visualización de software.												
	Realizar un filtrado inicial de referencias recientes en el campo de la visualización de software												
	Realizar un segundo filtrado de referencias para seleccionar los proyectos relacionados con metáforas del código fuente												
	Elaborar las conclusiones del estado del arte												
	Escoger y clasificar un conjunto inicial de métricas de software de acuerdo a las conclusiones del estado del arte.												
	Escoger y clasificar un conjunto inicial de métricas de los repositorios de código abierto de acuerdo a las conclusiones del estado del arte.												
	Proponer y desarrollar por lo menos tres ideas para la metáfora de representación de las métricas anteriormente seleccionadas.												
	Diseñar ilustraciones de ejemplo sobre las visualizaciones propuestas aplicadas a métricas extraídas de algunos artefactos de código fuente.												
FASE II	Documentar los casos de uso y diagramas de flujo de la aplicación a desarrollar.												
	Escoger la tecnología a implementar en la propuesta de visualización y conexiones con los repositorios de código fuente.												
	Diseñar la arquitectura y el modelado de información para la generación de las visualizaciones.												
	Implementar el diseño (desarrollo de la aplicación).												
FASE III	Seleccionar un grupo de tres proyectos como casos de estudio												
	Generar por cada proyecto una visualización (de ser posible) usando alguna de las metodologías cercanas (identificadas en el estado del arte).												
	Generar por cada proyecto una visualización usando la metodología propuesta.												
	Comparar los resultados obtenidos por las metodologías de referencia y la metodología propuesta.												
	Generar un listado de conclusiones respecto a la comparación del punto anterior.												
FASE IV	Redacción de un artículo científico para la difusión de la propuesta.												

6. BIBLIOGRAFÍA

- [1] Free Software Foundation, «¿Qué es el software libre?» [En línea]. Disponible en: <https://www.gnu.org/philosophy/free-sw.es.html>. [Accedido: 21-sep-2016].
- [2] Open Source Initiative, «About the Open Source Initiative». [En línea]. Disponible en: <https://opensource.org/about>. [Accedido: 21-sep-2016].
- [3] M. Dávila Sguerra, *GNU/linux y el software libre: y sus múltiples aplicaciones*. Bogotá Alfaomega 2009, 2009.
- [4] Richard Stallman, «Por qué el “código abierto” pierde de vista lo esencial del software libre». [En línea]. Disponible en: <https://www.gnu.org/philosophy/open-source-misses-the-point.html>. [Accedido: 21-sep-2016].
- [5] W. Stam, «When does community participation enhance the performance of open source software companies?», *Res. Policy*, vol. 38, n.º 8, pp. 1288-1299, oct. 2009.
- [6] H. Baytiyeh y J. Pfaffman, «Open source software: A community of altruists», *Online Interactivity Role Technol. Behav. Change*, vol. 26, n.º 6, pp. 1345-1354, nov. 2010.
- [7] I. Samoladas, G. Gousios, D. Spinellis, y I. Stamelos, «The SQO-OSS Quality Model: Measurement Based Open Source Software Evaluation», en *Open Source Development, Communities and Quality: IFIP 20th World Computer Congress, Working Group 2.3 on Open Source Software, September 7-10, 2008, Milano, Italy*, B. Russo, E. Damiani, S. Hissam, B. Lundell, y G. Succi, Eds. Boston, MA: Springer US, 2008, pp. 237-248.
- [8] «Open Hub, the open source network». [En línea]. Disponible en: <https://www.openhub.net/>. [Accedido: 22-sep-2016].
- [9] «Manifiesto por el Desarrollo Ágil de Software». [En línea]. Disponible en: <http://agilemanifesto.org/iso/es/manifesto.html>. [Accedido: 13-nov-2016].
- [10] T. Mens y S. Demeyer, *Software Evolution*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.
- [11] «Subversion Alternatives and Similar Software - AlternativeTo.net», *AlternativeTo*. [En línea]. Disponible en: <http://alternativeto.net/software/subversion/>. [Accedido: 22-sep-2016].
- [12] «GitHub Alternatives and Similar Software - AlternativeTo.net», *AlternativeTo*. [En línea]. Disponible en: <http://alternativeto.net/software/github/>. [Accedido: 22-sep-2016].
- [13] S. W. Thomas, A. E. Hassan, y D. Blostein, «Mining Unstructured Software Repositories.», *Evol. Softw. Syst.*, p. 139, ene. 2014.
- [14] C. Keskin y V. Vogelmann, «Effective Visualization of Hierarchical Graphs with the Cityscape Metaphor», en *Proceedings of the 1997 Workshop on New Paradigms in Information Visualization and Manipulation*, New York, NY, USA, 1997, pp. 52-57.
- [15] R. Wettel y M. Lanza, «Visualizing Software Systems as Cities», en *2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, 2007, pp. 92-99.
- [16] Y. Xu, Y. Liu, y J. Zheng, «To Enlighten Hidden Facts in The Code: A Review of Software Visualization Metaphors.», en *The 27th International Conference on Software Engineering and Knowledge Engineering, SEKE 2015, Wyndham Pittsburgh University Center, Pittsburgh, PA, USA, July 6-8, 2015*, 2015, pp. 294-297.
- [17] W. Lieser, T. Baumgärte, y K. Rosés Becker, *Arte digital: nuevos caminos en el arte*. Potsdam: Ullmann, 2010.
- [18] M. Michails, «Mining data, making art», *Digit. Creat.*, vol. 26, n.º 3-4, pp. 279-286, oct. 2015.
- [19] C. McGarrigle, «Augmented Resistance: The Possibilities for AR and Data Driven Art + Interview, Statement, Artwork», *Leonardo Electron. Alm.*, vol. 19, n.º 1, ene. 2013.
- [20] «The Art of Pi - A Colorful Data Visualization», *Visual News*, 09-jul-2013. .

- [21] «Top 5 open source community metrics to track», *Opensource.com*. [En línea]. Disponible en: <https://opensource.com/business/15/12/top-5-open-source-community-metrics-track>. [Accedido: 25-sep-2016].
- [22] Paulo Roberto Miranda Meirelles, «Monitoramento de métricas de código-fonte em projetos de software livre», 2013.
- [23] «Malwarez». [En línea]. Disponible en: <http://sq.ro/malwarez.htm#>. [Accedido: 19-sep-2016].
- [24] M. Ogawa y K. L. Ma, «code_swarm: A Design Study in Organic Software Visualization», *IEEE Trans. Vis. Comput. Graph.*, vol. 15, n.º 6, pp. 1097-1104, nov. 2009.
- [25] «code_swarm». [En línea]. Disponible en: http://www.michaelogawa.com/code_swarm/. [Accedido: 18-sep-2016].
- [26] «Gource - a software version control visualization tool». [En línea]. Disponible en: <http://gource.io/>. [Accedido: 26-ago-2016].
- [27] F. Beck, «Software Feathers figurative visualization of software metrics», *2015 Int. Conf. Soft-Comput. Netw. Secur. ICSNS*, p. 5, ene. 2015.
- [28] «Codeology». [En línea]. Disponible en: <http://codeology.braintreepayments.com/>. [Accedido: 19-sep-2016].