UNIVERSIDAD NACIONAL DE COLOMBIA

# Proyecto ShoggothViz[1]

*Metodología para la visualización de la evolución de código fuente*

**Desarrollado por:** Joan Sebastian Lopez Riaño
**Institución:** Universidad Nacional de Colombia – Sede Bogotá
**Programa:** Maestría en Ingeniería de Sistemas y Computación – Perfil de profundización
**Fecha:** Abril de 2017

# MARCO TEÓRICO Y ESTADO DEL ARTE

## MARCO TEÓRICO

### 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 especifico. 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.

### 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

1. Es un nombre de proyecto provisional. Al definirse la metáfora visual a implementar se definirá un nuevo nombre para el proyecto.

garantiza las cuatro libertades básicas enunciadas por Richard Stallman (Free Software Foundation, n.d.):

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 Iniciative* (Open Source Initiative, n.d.) 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 Iniciative* 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 Sguerra, 2009): "*La Open Source Iniciative 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*".

**La categoría *F.L.O.S.S***

Años después de iniciado el debate entre la *FSF* y la *Open Source Iniciative* 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"*(Richard Stallman, n.d.).

***Software Evolution* o la 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* ("Manifiesto por el Desarrollo Ágil de Software," n.d.) 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.

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 las labores de mantenimiento 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 (Mens and Demeyer, 2008). 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* ("Subversion Alternatives and Similar Software – AlternativeTo.net," n.d.) y plataformas colaborativas tales como *GitHub*, *SourceForge*, *RedMine*, entre otras ("GitHub Alternatives and Similar Software – AlternativeTo.net," n.d.).

**Minería de repositorios de *Software***
La minería de repositorios de software se refiere a las técnicas usualmente aplicadas a repositorios públicos de software tales como *SourceForge*[2] o *GitHub*[3], 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*[4] 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 (Thomas et al., 2014).

Los repositorios de software pueden ser privados o públicos, siendo estos últimos 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 duplicados del mismo (también conocidos como "*forks"*).

**Métricas del código fuente**
Gracias al aporte de (Paulo Roberto Miranda Meirelles, 2013) se podrían generalizar la siguientes métricas del código fuente aplicadas a proyectos de la categoría *FLOSS*:

2. Sitio web : *sourceforge.net*
3. Sitio web : *github.com*
4.Errores de programa , también conocidos por la expresión *bugs*. El termino se acuño en el año 1967 al encontrarse presencia de insectos en la computadora Mark II.d

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

***Tabla 1. Métricas de tamaño***

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

***Tabla 2. Métricas de acoplamiento entre clases***

## Visualización de Software

Comprende el área de la ingeniería de software que se encarga de desarrollar metodologías y herramientas visuales para entender mejor el diseño y funcionalidad de una aplicación usando como base el análisis a profundidad de artefactos como el código fuente (principalmente), la información extraída de repositorios de software, entre otros.

Sin embargo una apropiada definición se extendería al relacionar otros aspectos que vayan más allá de la visualización de artefactos de software y el proceso de desarrollo del mismo. En el área de investigación de la visualización de software se encaminan esfuerzos para el desarrollo de propuestas que permitan la visualización de *estructura*, *comportamiento* y la *evolución* de todas las piezas que comprenden un programa (Diehl, 2007).
El proceso según el cual se construyen las visualizaciones podría resumirse como : Adquisición de información, luego análisis de la misma y finalmente el mapeado de lo analizado en una representación

visual, la cual puede estar compuesta por una o mas imágenes, incluso hasta animaciones (Diehl, 2007).

**Aplicaciones de la *visualización de software***

Según investigaciones y encuestas enfocadas en el estudio de los principales beneficios de la implementación de metodologías de visualización en proyectos de software, se encuentran (Stefan-Lucian Voinea, 2007) :

- Reducción en los costos de producción del software
- Mejor comprensión del sistema
- Incremento en la productividad
- Gestión de la complejidad
- Asistencia en la búsqueda de errores
- Mejoramiento de la calidad

**Grafos y metáforas de visualización**

Las visualizaciones podrían clasificarse en dos grandes grupos: Representaciones de grafos y metáforas visuales. Las representaciones de grafos por lo general se componen de representaciones abstractas compuestas de conjuntos de nodos y conexiones entre los mismos, así como también diagramas tipo árbol o las ya clásicas representaciones por medio de figuras geométricas como cilindros, rectángulos, cubos, etc.

Por otra parte las metáforas visuales son todas aquellas representaciones donde se usan conceptos de la realidad tales como edificios, arboles, planetas, plumas, entre otras, con el objetivo de *"transferir propiedades de el dominio especifico de la representación gráfica al dominio de la entidad o concepto abstracto que se quiere representar"* (Diehl, 2007).

**Enfoques de las metodologías de visualización**

Dependiendo del enfoque del objeto o concepto a visualizar se pueden también clasificar las metodologías de como estáticas, dinámicas y de evolución. Las visualizaciones estáticas son aquellas que se enfocan en el análisis y comprensión de aspectos relacionados con la estructura del sistema o pieza de software, lo cual incluye no solo el código fuente sino también toda aquella información relacionada con el proceso de desarrollo o versionamiento. Por otra parte las visualizaciones dinámicas son aquellas que se componen por todas aquellas metodologías que estudian el comportamiento de los flujos de información cuando el programa esta ejecutándose y que busca abstraer conceptos relacionados con algoritmia, optimización y localización de fallos en entornos de pruebas.

**Visualización de la evolución del software**

En su disertación doctoral (Stefan-Lucian Voinea, 2007) plantea que la principal aplicación de la visualización de software cuando se enfoca en el área de evolución es la de la facilitar las tareas de la fase de mantenimiento de sistemas de software complejos. En general el autor plantea que la principal cuestión de la naciente área de la visualización de la evolución del software es: "*¿Como habilitar a los usuarios para obtener información sobre la evolución de un sistema de software?"*. Gracias al planteamiento de esta cuestión se pueden esbozar los principales problemas o retos que donde la aplicación de metodologías de visualización podría contribuir:

- *Escalábilidad*
- *Intuición*
- *Usabilidad*
- *Integración*

Por otra parte *(Diehl, 2007) plantea cinco campos de aplicación de metodologías de visualización enfocadas en el análisis de la evolución de sistemas de software:*

- *Visualización de cambios en métricas relacionadas al software*
- *Visualización de archivos o artefactos*
- *Visualización del cambio estructural*
- *Visualización del acoplamiento evolutivo*
- *Minería de datos visuales*

## ESTADO DEL ARTE

### Objeto de estudio

El presente estudio tiene como objetivo indagar sobre las principales tendencias de la actualidad respecto a la producción académica en el área de visualización de software, pero mas en especifico en el campo de *metodologías (con sus correspondientes implementaciones) para la generación de visualizaciones por medio de metáforas enfocadas en el análisis de la evolución del software y que a su vez tengan un componente de interacción con el usuario final*.

Para hacerse a una idea del comportamiento de la comunidad académica y de las investigaciones que permitan acotar el campo de estudio es necesario primero establecer las principales características de las propuestas en el campo de la visualización de software en su conjunto. La mejor forma de acercarse al entendimiento de este fenómeno es a través del estudio de revisiones sistemáticas de la literatura desde el año 2015 hasta el año 2017. Finalmente después de acotar los trabajos de investigación mas relacionados con el área de estudio se expondrán algunos de los mas relevantes con el objetivo de llegar a un conjunto de conclusiones que permitan delimitar la definición del problema y la justificación de la propuesta.

**Producción y diversidad de representaciones visuales**

En su investigación (Xu et al., 2015) buscó encontrar relaciones cuantificando la producción académica del momento partiendo primero de la idea de clasificar las propuestas encontradas (un total de 81)según la tarea que abordaban así como el tipo de representación visual implementada. Respecto a las tareas o tipo de análisis que soportaban las investigaciones estudiadas los autores implementaron las tres categorías definidas por (Diehl, 2007):

- Análisis estático: 54 estudios
- Análisis dinámico: 15 estudios
- Evolución: 12 estudios

De igual forma el mencionado estudio busco cuantificar y clasificar los distintos tipos de representaciones visuales implementadas. Dicho análisis se puede resumir en el siguiente gráfico:

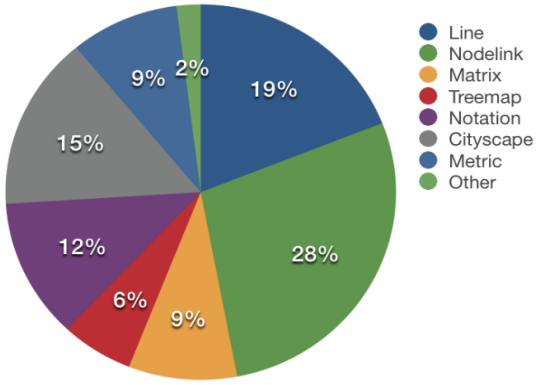


*Fig 1. Cantidad de estudios por tipo de representación visual (metáforas)*

Dentro de las principales conclusiones de la investigación se encuentra:

La mayoría de las metáforas implementadas son de tipo abstracto (diagramas, grafos) por sobre las metáforas análogas a conceptos de la realidad (ciudades, arboles, etc).

La mayoría de estudios se enfocan en la generación de nuevas propuestas de representación visual por sobre la validación de las ya existentes.

La mayoría de estudios enfocados en la evolución se concentran en el análisis de los cambios de distintas métricas del código fuente.

**Técnicas de visualización según el enfoque**

En (Salameh et al., 2016) se evidencia un trabajo por llevar a cabo una revisión sistemática de la literatura que permitiera dar cuenta de una clasificación actual para los trabajos de investigación en herramientas aplicadas a la visualización de la evolución del software. La propuesta establece la siguiente clasificación de los tipos de técnicas de visualización implementadas:

- Basada en grafos
- Basada en notaciones
- Basada en matrices
- Basada en metáforas
- Otras

Para una clasificación que dividiera los trabajos encontradas respecto a su objetivo o concepto a representar los autores proponen las siguientes categorías:

- *Visualización enfocada en los cambios sobre artefactos*
- *Visualización enfocada en los cambios sobre métricas*
- *Visualización enfocada en los cambios sobre características*
- *Visualización enfocada en los cambios sobre la arquitectura*

Los resultados de cuantificar dichas categorías se pueden evidenciar en los siguientes gráficos:

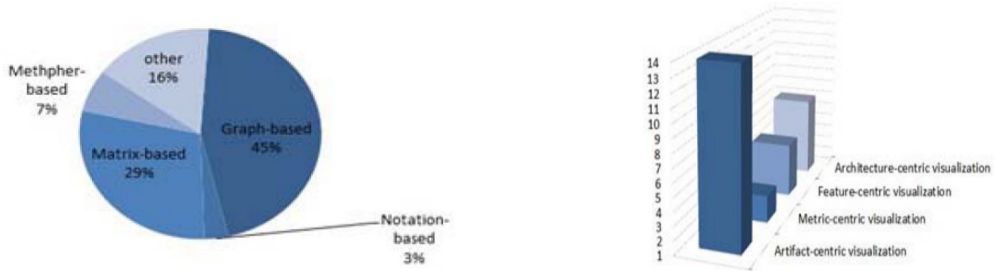


*Fig.2 Cantidad de trabajos encontrados según el tipo de técnica de visualización y según el enfoque*

**Herramientas que soportan la visualización de software**

En (Cruz et al., 2016) se llevó a cabo una revisión sistemática de la literatura donde a partir del análisis de 52 investigaciones se pudieron identificar 28 técnicas de visualización así como 33 herramientas o implementaciones de dichas técnicas. Los resultados del estudio, fueron clasificados según la técnica, el enfoque (estructura, jerarquía, artefactos, dependencia, evolución y métricas) , el tipo de análisis (estático o dinámico) , el formato (grafos abstractos o metáforas) entre otras y si existía o no una herramienta de software concreta que implementara dicha técnica o metodología de visualización. El siguiente cuadro resume la clasificación descrita:

| | Técnica | Dimensión | Análisis | Formato | Herramienta |
|---|---|---|---|---|---|
| Estructura | 3D-HEB | 3D | Estática | Grafos/Metáfora de Ciudad | - |
| | BoxTree | 3D | Estática | Cajas / Árboles | Bloom |
| | City | 3D | Estática | Metáfora de Ciudad | CodeCity<br>UnifiedCity<br>VizzAspect |
| | Nested View | 2D | Estática | Rectángulos anidados | Creole |
| | Solar System | 3D | Estática | Metáfora de Sistema Solar | - |
| | Spring | 3D | Estática | Grafos | Creole |
| | Sunburst | 3D | Estática | Grafos Discos anidados | - |
| | Tree | 2D | Estática | Bosques / Árboles | Creole<br>CodeForest<br>CodeTrees |
| | TreeMap | 2D | Estática | Rectángulos anidados | Shrimp Concept<br>Creole<br>SourceMiner<br>CRISTA<br>X-Ray |
| | Verso | 3D | Estática | Cajas / Cilindros | - |

*Fig.3 Clasificación de técnicas y herramientas para la visualización de software*

| | Técnica | Dimensión | Análisis | Formato | Herramienta |
|---|---|---|---|---|---|
| Jerarquía | Hiperbolic Tree | 2D | Estática | Árbol radial | Concept |
| | Nested View | 2D | Estática | Rectángulos anidados | Creole |
| | Polymetric Views | 2D | Estática | Rectángulos en formato de árbol | CodeCrawler SourceMiner X-Ray VizzAspect CodeTrees JCHVT |
| | Radial Tree | 2D | Estática | Árbol radial | SA4J |
| Código fuente | Animation System | 3D | Dinámica | Diversas Formas | GASP ReqViz3D |
| | File Maps | 2D | Dinámica | Columnas paralelas | Bloom |
| | Layouts | 3D/2D | Dinámica | Grafos | Bloom |
| | Point Maps | 3D | Dinámica | Puntos de dispersión | Bloom |
| | Polycilinders | 3D | Estática | Cilindros | SeeIT |
| | Spirais | 3D | Dinámica | Espiral | Bloom |
| Dependencias | Package Blueprint | 2D | Estática | Matriz | - |
| | Hypergrafos | 2D | Estática | Grafos | SEXTANT TraceCrawler Rigi |
| | Pyramid | 2D | Estática | Tablas dispuestas como piramide | SA4J |
| | Semantic Dependency Matrix | 2D | Estática | Matriz | Softwarenaut |
| Evolución | Edge Evolution Film Strip | 2D | Estática | Grafos | Softwarenaut |
| | RelVis | 2D | Estática | Grafos/Diagramas Kiviat | - |
| | Timeline | 3D | Estática | Metáfora de Cidade | - |
| Métricas | Graphs | 2D | Dinámica | Gráficos | Evolve |
| Detalles | Exploded View | 3D | Estática | Diversas Formas | - |

*Fig.3 Clasificación de técnicas y herramientas para la visualización de software (Continuación)*

De los resultados encontrados en el estudio de (Cruz et al., 2016) se puede identificar que las representaciones abstractas son mas implementadas que las metáforas y de como en el enfoque evolutivo no se encuentran muchas herramientas concretas que signifiquen una puesta en practica de las técnicas mencionadas.

**Implementación de metáforas en la visualización**

Gracias a los aportes de las revisiones sistemáticas de la literatura mencionados anteriormente se puede evidenciar el uso y evolución en el tiempo de las distintas metáforas visuales propuestas. Se pueden resumir las propuestas de metáforas implementadas en la siguiente tabla:

| Metáfora | Trabajos relacionados |
|---|---|
| Ciudades | (Balogh et al., 2016, 2015; Caserta et al., 2011; Fittkau et al., 2015a; Steinbrückner, 2010; Wettel and Lanza, 2008, 2007) |
| Casas | (Boccuzzo and Gall, 2007) |
| Bosques | (Erra et al., 2012) |
| Plumas | (Beck, 2014) |
| Sistemas solares | (Graham et al., 2004) |
| Mapas cartográficos | (Kuhn et al., 2010), (Hawes et al., 2015), (Steinberg and Ziv, 1992) |
| Dispersión de ondas | (Magnavita, 2016) |
| Mapas de calor | (Benomar et al., 2013) |
| Lineas del metro | (Martínez et al., 2008a) |
| Paisajes | (Balzer et al., 2004; Martínez et al., 2008b; Zirkelbach, 2016) |
| Estructuras moleculares | (Malloy and Power, 2005 |

Si bien el campo de las metáforas puede ser tan amplio como los esfuerzos de la comunidad académica se lo permitan, existen metáforas muy versátiles que pueden ajustarse a distintos enfoques utilizando la misma analogía. Tal es el caso de la metáfora de las Ciudades. En estos casos se encuentran implementaciones del modelo de ciudades aplicado a tareas de comprensión de software (Zirkelbach, 2016), análisis de métricas de código fuente (Balogh et al., 2016), evolución (Lanza et al., 2009), arquitectura *(Kobayashi et al., 2013)***,** acoplamiento(Caserta et al., 2011), revisiones criticas de la metáfora en si misma (Steinbrückner, 2010) así como distintas aproximaciones en términos de interactividad como el uso de la realidad virtual para navegar por la visualización (Fittkau et al., 2015b). También podría considerarse algunas metáforas como re-interpretaciones de otras analogías, tal es el caso de la implementación de bosques para la representación de características del software reemplazando la metáfora del edificio en una ciudad por la de un árbol en un bosque. Por otra parte existen proyectos artísticos, por fuera del dominio de la ingeniería de software, que buscan "dar un rostro" a los artefactos de código fuente, siendo los mas relevantes *Malwarez (Dragulescu, 2014)*, que implementa una metáfora de organismos unicelulares y *Codeology* (Leyden, 2016) que usa la metáfora de insectos sobre proyectos hospedados en *Github*.

**Visualizaciones aplicadas a la evolución del Software**

Dentro de los trabajos recientes mas relevantes relacionados con el tema de evolución del software, se encuentra *Evowave* (Magnavita, 2016), . Dicho proyecto resalta por la versatilidad para representar distintos conjuntos de información relacionados con el proceso de desarrollo y evolución de un proyecto de software, involucrando distintas métricas que van desde el análisis de cambios de artefactos hasta los autores que se han involucrado en el desarrollo. La metáfora de las ondas es análoga al fenómeno de la dispersión de las mismas a lo largo del tiempo de forma circular. En la siguiente figura se puede ver la participación de varios autores siento el anillo mas interior el tiempo inicial y el anillo exterior el tiempo final del análisis:

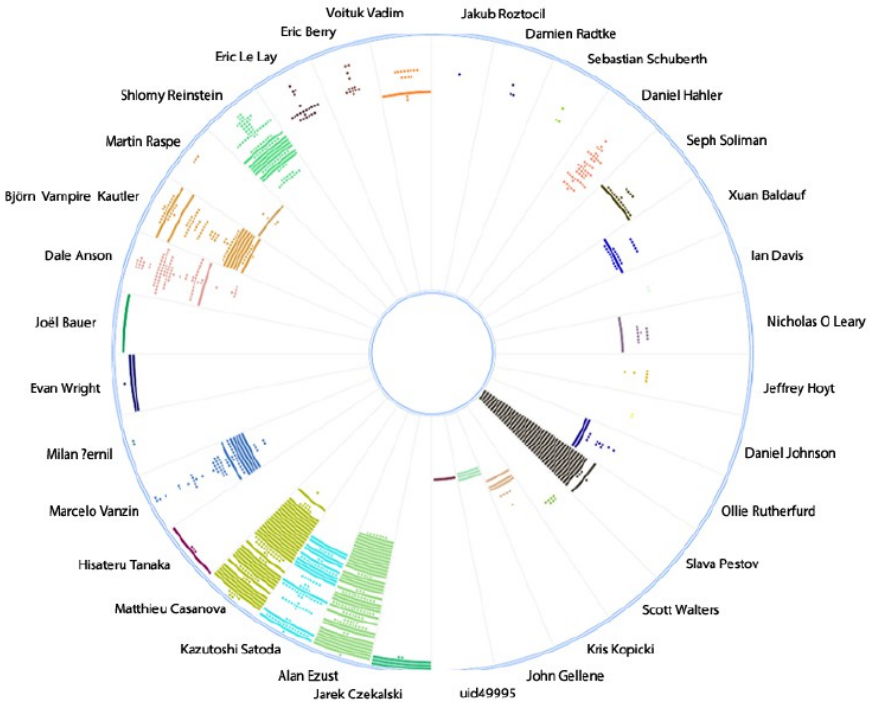


*Fig 4.Visualización generada por Evowave para representar la cantidad de trabajo de cada autor en un proyecto de software a lo largo del tiempo.*

Por otra parte el proyecto *Evospaces* (Alam and Dugerdil, 2007), (Lanza et al., 2009)es una herramienta desarrollada por el mismo equipo de investigadores que implementaron las metodologías de *CodeCity* (Wettel and Lanza, 2008) y *CocoViz* (Boccuzzo and Gall, 2007), unificando ambos conceptos en una herramienta que sintetiza las características de sus predecesores pero que a su vez implementa categorías nuevas como lo es el análisis dinámico de las trazas de ejecución.

Respecto al análisis de la evolución en el tiempo y de la participación de autores en los procesos de desarrollos los mas fuertes referentes en el campo son el proyecto de animación basada en los sistemas de versionamiento *Gource*("Gource - a software version control visualization tool," n.d.) y en el estudio de las interacciones y la historia de los *commits* de un proyecto se encuentra *Code_swarm (Ogawa and Ma, 2009).*

**La interactividad en las técnicas de visualizaciones implementadas**

Dentro de el conjunto de herramientas que implementan metodologías de visualización, solo algunas de ellas tienen un componente interactivo, es decir, que el usuario final puede manipular la visualización resultante, bien sea modificando su conjunto de origen de datos, los enfoques de proximidad o simplemente navegando por la visualización. Dentro de los proyectos que mas resaltan por su interactividad encontramos:

- *(Shah and Guyer, 2016)* –Una herramienta para la visualización del grafo de llamadas entre métodos. Desarrollado en la plataforma *Processing* la cual permite implementar una interfaz de usuario interactiva.
- *(Schneider et al., 2016) – CuboidMatrix,* implementa una visualización dentro de un cubo, donde uno de sus ejes representa el tiempo, permitiendo de esta manera navegar por un conjunto de datos desde la interfaz de usuario.
- *(Merino et al., 2016) – Metavis* , implementa la metáfora de nubes, o *tags* para navegar por conceptos relevantes marcados en un conjunto de datos
- *(Hawes et al., 2015) – CodeSurveyor,* su interactividad esta dada por la navegación que se puede tener a lo largo de la visualización, la cual implementa la metáfora de un mapa cartográfico permitiendo realizar acercamientos a distintos niveles de detalle.
- *(Benomar et al., 2013)* – A través de una visualización que mezcla la metáfora de la ciudad con la de mapas de calor, este trabajo implementa elementos de navegación a lo largo de un entorno 3D.
- *(Erra et al., 2012)* –Al igual que el trabajo con mapas de calor, esta propuesta implementa la metáfora del bosque y árboles donde el usuario puede navegar por el escenario 3D generado por el proceso de visualización.

**Conclusiones del estado del arte**

1. Es notable que la mayoría de propuestas se decantan por el uso de representaciones gráficas abstractas (nodos, grafos, diagramas) por sobre la implementación de metáforas.
2. Son escasas las propuestas que internacionalizan visualizaciones enfocadas al cuestiones de la evolución del software con componentes interactivos enfocados en el usuario final.
3. No son muchas las propuestas dentro de la categoría de estudio de la evolución del software que implementen metáforas figurativas.

4. Si bien existen muchas metodologías y trabajos que se enmarcan dentro del area de visualización de software, no todas las propuestas implementan herramientas concretas que permitan poner en practica las mismas.

**BIBLIOGRAFÍA**


Alam, S., Dugerdil, P., 2007. EvoSpaces Visualization Tool: Exploring Software Architecture in 3D, in: 14th Working Conference on Reverse Engineering (WCRE 2007). Presented at the 14th Working Conference on Reverse Engineering (WCRE 2007), pp. 269–270. doi:10.1109/WCRE.2007.26

Balogh, G., Gergely, T., Beszédes, Á., Gyimóthy, T., 2016. Using the City Metaphor for Visualizing Test-Related Metrics, in: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). Presented at the 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pp. 17–20. doi:10.1109/SANER.2016.48

Balogh, G., Szabolics, A., Beszédes, Á., 2015. CodeMetropolis: Eclipse over the city of source code, in: 2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM). Presented at the 2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM), pp. 271–276. doi:10.1109/SCAM.2015.7335425

Balzer, M., Noack, A., Deussen, O., Lewerentz, C., 2004. Software Landscapes : Visualizing the Structure of Large Software Systems. Presented at the IEEE TCVG. doi:10.2312/VisSym/VisSym04/261-266

Beck, F., 2014. Software Feathers figurative visualization of software metrics, in: 2014 International Conference on Information Visualization Theory and Applications (IVAPP). Presented at the 2014 International Conference on Information Visualization Theory and Applications (IVAPP), pp. 5–16.

Benomar, O., Sahraoui, H., Poulin, P., 2013. Visualizing software dynamicities with heat maps, in: 2013 First IEEE Working Conference on Software Visualization (VISSOFT). Presented at the 2013 First IEEE Working Conference on Software Visualization (VISSOFT), pp. 1–10. doi:10.1109/VISSOFT.2013.6650524

Boccuzzo, S., Gall, H., 2007. CocoViz: Towards Cognitive Software Visualizations, in: 2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis. Presented at the 2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, pp. 72–79. doi:10.1109/VISSOF.2007.4290703

Caserta, P., Zendra, O., Bodénès, D., 2011. 3D Hierarchical Edge bundles to visualize relations in a software city metaphor, in: 2011 6th International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT). Presented at the 2011 6th International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT), pp. 1–8. doi:10.1109/VISSOF.2011.6069451

Cruz, A., Bastos, C., Afonso, P., Costa, H., 2016. Software visualization tools and techniques: A systematic review of the literature, in: 2016 35th International Conference of the Chilean Computer Science Society (SCCC). Presented at the 2016 35th International Conference of the Chilean Computer Science Society (SCCC), pp. 1–12. doi:10.1109/SCCC.2016.7836048

Dávila Sguerra, M., 2009. GNU/linux y el software libre : y sus múltiples aplicaciones. Bogotá Alfaomega 2009.

Diehl, S., 2007. Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software. Springer Science & Business Media.

Dragulescu, A., 2014. From the Malwarez and Ekisto series. diacritics 42, 110–117. doi:10.1353/dia.2014.0005

Erra, U., Scanniello, G., Capece, N., 2012. Visualizing the Evolution of Software Systems Using the Forest Metaphor, in: 2012 16th International Conference on Information Visualisation. Presented at the 2012 16th International Conference on Information Visualisation, pp. 87–92. doi:10.1109/IV.2012.25

Fittkau, F., Koppenhagen, E., Hasselbring, W., 2015a. Research perspective on supporting software engineering via physical 3D models, in: 2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT). Presented at the 2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT), pp. 125–129. doi:10.1109/VISSOFT.2015.7332422

Fittkau, F., Krause, A., Hasselbring, W., 2015b. Exploring software cities in virtual reality, in: 2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT). Presented at the 2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT), pp. 130–134. doi:10.1109/VISSOFT.2015.7332423

Free Software Foundation, n.d. ¿Qué es el software libre? [WWW Document]. URL https://www.gnu.org/philosophy/free-sw.es.html (accessed 9.21.16).

GitHub Alternatives and Similar Software - AlternativeTo.net [WWW Document], n.d. . AlternativeTo. URL http://alternativeto.net/software/github/ (accessed 9.22.16).

Gource - a software version control visualization tool [WWW Document], n.d. URL http://gource.io/ (accessed 8.26.16).

Graham, H., Yang, H.Y., Berrigan, R., 2004. A Solar System Metaphor for 3D Visualisation of Object Oriented Software Metrics, in: Proceedings of the 2004 Australasian Symposium on Information Visualisation - Volume 35, APVis '04. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, pp. 53–59.

Hawes, N., Marshall, S., Anslow, C., 2015. CodeSurveyor: Mapping large-scale software to aid in code comprehension, in: 2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT). Presented at the 2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT), pp. 96–105. doi:10.1109/VISSOFT.2015.7332419

Kobayashi, K., Kamimura, M., Yano, K., Kato, K., Matsuo, A., 2013. SArF map: Visualizing software architecture from feature and layer viewpoints, in: 2013 21st International Conference on Program Comprehension (ICPC). Presented at the 2013 21st International Conference on Program Comprehension (ICPC), pp. 43–52. doi:10.1109/ICPC.2013.6613832

Kuhn, A., Erni, D., Loretan, P., Nierstrasz, O., 2010. Software Cartography: thematic software visualization with consistent layout. J. Softw. Maint. Evol. Res. Pract. 22, 191–210. doi:10.1002/smr.414

Lanza, M., Gall, H., Dugerdil, P., 2009. EvoSpaces: Multi-dimensional Navigation Spaces for Software Evolution, in: 2009 13th European Conference on Software Maintenance and Reengineering. Presented at the 2009 13th European Conference on Software Maintenance and Reengineering, pp. 293–296. doi:10.1109/CSMR.2009.14

Leyden, K., 2016. Turning Code into Gold with Codeology [WWW Document]. Strongly Typed. URL http://www.braintreepayments.com/blog/turning-code-into-gold-with-codeology/ (accessed 9.19.16).

Magnavita, R.C., 2016. Evowave: A Multiple Domain Metaphor for Software Evolution Visualization.

Malloy, B.A., Power, J.F., 2005. Using a molecular metaphor to facilitate comprehension of 3D object diagrams, in: 2005 IEEE Symposium on Visual Languages and Human-Centric Computing

(VL/HCC'05). Presented at the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05), pp. 233–240. doi:10.1109/VLHCC.2005.66

Manifiesto por el Desarrollo Ágil de Software [WWW Document], n.d. URL http://agilemanifesto.org/iso/es/manifesto.html (accessed 11.13.16).

Martínez, A.A., Cosín, J.J.D., García, C.P., 2008a. A Metro Map Metaphor for Visualization of Software Projects, in: Proceedings of the 4th ACM Symposium on Software Visualization, SoftVis '08. ACM, New York, NY, USA, pp. 199–200. doi:10.1145/1409720.1409754

Martínez, A.A., Cosín, J.J.D., García, C.P., 2008b. A Landscape Metaphor for Visualization of Software Projects, in: Proceedings of the 4th ACM Symposium on Software Visualization, SoftVis '08. ACM, New York, NY, USA, pp. 197–198. doi:10.1145/1409720.1409753

Mens, T., Demeyer, S., 2008. Software Evolution. Springer Berlin Heidelberg, Berlin, Heidelberg.

Merino, L., Ghafari, M., Nierstrasz, O., Bergel, A., Kubelka, J., 2016. MetaVis: Exploring Actionable Visualization, in: 2016 IEEE Working Conference on Software Visualization (VISSOFT). Presented at the 2016 IEEE Working Conference on Software Visualization (VISSOFT), pp. 151–155. doi:10.1109/VISSOFT.2016.19

Ogawa, M., Ma, K.L., 2009. code_swarm: A Design Study in Organic Software Visualization. IEEE Trans. Vis. Comput. Graph. 15, 1097–1104. doi:10.1109/TVCG.2009.123

Open Source Initiative, n.d. About the Open Source Initiative [WWW Document]. URL https://opensource.org/about (accessed 9.21.16).

Paulo Roberto Miranda Meirelles, 2013. Monitoramento de métricas de código-fonte em projetos de software livre.

Richard Stallman, n.d. Por qué el «código abierto» pierde de vista lo esencial del software libre [WWW Document]. URL https://www.gnu.org/philosophy/open-source-misses-the-point.html (accessed 9.21.16).

Salameh, H.B., Ahmad, A., Aljammal, A., 2016. Software evolution visualization techniques and methods - a systematic review, in: 2016 7th International Conference on Computer Science and Information Technology (CSIT). Presented at the 2016 7th International Conference on Computer Science and Information Technology (CSIT), pp. 1–6. doi:10.1109/CSIT.2016.7549475

Schneider, T., Tymchuk, Y., Salgado, R., Bergel, A., 2016. CuboidMatrix: Exploring Dynamic Structural Connections in Software Components Using Space-Time Cube, in: 2016 IEEE Working Conference on Software Visualization (VISSOFT). Presented at the 2016 IEEE Working Conference on Software Visualization (VISSOFT), pp. 116–125. doi:10.1109/VISSOFT.2016.17

Shah, M.D., Guyer, S.Z., 2016. An Interactive Microarray Call-Graph Visualization, in: 2016 IEEE Working Conference on Software Visualization (VISSOFT). Presented at the 2016 IEEE Working Conference on Software Visualization (VISSOFT), pp. 86–90. doi:10.1109/VISSOFT.2016.14

Stefan-Lucian Voinea, 2007. Software evolution visualization. Technische Universiteit Eindhoven, Eindhoven.

Steinberg, D., Ziv, H., 1992. Software visualization and Yosemite National Park, in: Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences. Presented at the Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences, pp. 607–618 vol.2. doi:10.1109/HICSS.1992.183312

Steinbrückner, F., 2010. Coherent Software Cities, in: 2010 IEEE International Conference on Software Maintenance. Presented at the 2010 IEEE International Conference on Software Maintenance, pp. 1–2. doi:10.1109/ICSM.2010.5610421

Subversion Alternatives and Similar Software - AlternativeTo.net [WWW Document], n.d. . AlternativeTo. URL http://alternativeto.net/software/subversion/ (accessed 9.22.16).

Thomas, S.W., Hassan, A.E., Blostein, D., 2014. Mining Unstructured Software Repositories. Evol. Softw. Syst. 139.

Wettel, R., Lanza, M., 2008. CodeCity: 3D Visualization of Large-scale Software, in: Companion of the 30th International Conference on Software Engineering, ICSE Companion '08. ACM, New York, NY, USA, pp. 921–922. doi:10.1145/1370175.1370188

Wettel, R., Lanza, M., 2007. Visualizing Software Systems as Cities, in: 2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis. Presented at the 2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, pp. 92–99. doi:10.1109/VISSOF.2007.4290706

Xu, Y., Liu, Y., Zheng, J., 2015. To Enlighten Hidden Facts in The Code: A Review of Software Visualization Metaphors., in: The 27th International Conference on Software Engineering and Knowledge Engineering, SEKE 2015, Wyndham Pittsburgh University Center, Pittsburgh, PA, USA, July 6-8, 2015. pp. 294–297. doi:10.18293/SEKE2015-203

Zirkelbach, C., 2016. ExplorViz - Live Trace Visualization for System and Program Comprehension in Large Software Landscapes, in: [Invited Talk] In: ZBW Research Colloquium, 27 Jul 2016, Kiel, Germany . Presented at the ZBW Research Colloquium, Kiel, Germany.