HERRAMIENTAS PARA LA COMPRENSIÓN DE PROGRAMAS

Mario M. Berón
Departamento de Informática – Universidad Nacional de San Luis
San Luis – Capital – Argentina
Correo Electrónico: marioberon@alfa.di.uminho.pt

Pedro R. Henriques
Departamento de Informática – Universidade do Minho
Braga – Minho – Portugal
Correo Electrónico: prh@di.uminho.pt

Maria J. Varanda Pereira
Departamento de Informática – Universidade do Minho
Braga – Minho – Portugal
Correo Electrónico: mjoao@ipb.pt

Roberto Uzal
Departamento de Informática – Universidad Nacional de San Luis
San Luis – Capital – República Argentina
Correo Electrónico: ruzal@sinectis.com

RESUMEN

La comprensión de programas es un área de la Ingeniería del Software que se encarga del estudio y construcción de modelos y herramientas con el objetivo de facilitar el mantenimiento, la modificación y el estudio de aplicaciones de informática.

En este artículo presentamos una línea de investigación que aborda esta temática siguiendo tres pasos. El primero analiza los modelos cognitivos. El segundo estudia las herramientas actuales. El tercero examina la posibilidad de aplicación de las técnicas de comprensión de programas a los algoritmos de ruteo.

El análisis de los modelos cognitivos permitirá la elaboración de parámetros para analizar las herramientas de comprensión. El estudio de las aplicaciones existentes posibilitará conocer los modelos y estrategias utilizados por esta clase de aplicaciones. Finalmente, pensamos que la aplicación de las técnicas de comprensión de programas a los algoritmos de ruteo será un aporte para la generación de visiones innovadoras de los sistemas informáticos.

Palabras Claves: Comprensión de Programas, Estrategias de Comprensión, Herramientas.

1. INTRODUCCIÓN

La comprensión de programas consiste en la habilidad de entender varias unidades o módulos de código escritos en un lenguaje de programación de alto nivel, que integran una aplicación informática. La comprensión de programas está profundamente ligada a la Ingeniería del Software y se necesita para la reutilización, inspección, manutención, reingeniería, migración y extensión de los sistemas de software existentes [5] [9].

La comprensión de un programa puede tener diferentes significados o perspectivas. Podemos estar interesados en la forma en que la computadora ejecuta las instrucciones con el objetivo de entender el flujo de control; o podemos enfocar nuestro interés en los efectos que producirá la ejecución del programa en el dispositivo (físico o virtual) que es manipulado por el mismo. En cualquiera de los dos casos pensamos que una herramienta de inspección visual es fundamental para efectuar este tipo de tareas.

En este contexto el proyecto PCVIA (Program Comprensión by Visual Inspection and Animation) tiene como propósito: i) estudiar los modelos cognitivos; ii) analizar continuamente el estado del arte de sistemas de comprensión, iii) desarrollar herramientas de comprensión de programas basadas en vistas complementarias.

En este artículo describiremos esta línea de investigación y relataremos las actividades que se están llevando a cabo en el proyecto relacionadas con comprensión de algoritmos de ruteo [4] [7].

Este artículo está organizado como sigue. En la sección 2 describimos concisamente los modelos comprensión de programas más utilizados. La sección 3 describe brevemente las herramientas de comprensión de programas actuales. En la sección 4 presentamos los trabajos iniciales orientados a la creación de una herramienta de comprensión de algoritmos de ruteo. Finalmente, en la sección 5 se presentan las conclusiones y trabajos futuros.

2. MODELOS COGNITIVOS

Los modelos cognitivos [12] son importantes porque son la base para la construcción de herramientas de comprensión de programas[16]. En este ámbito, diversos autores han presentado diferentes modelos cognitivos para describir la forma que tienen de entender los sistemas los programadores. Los artículos [12][14][17] nos permiten distinguir las siguientes formas principales de comprensión: desde conceptos específicos hacia conceptos generales (Bottom Up), desde conceptos generales hacia conceptos específicos (Top Down), Comprensión Basada en una Base de Conocimiento (Knowledge based Understanding Model), Comprensión Oportunista y Sistemática (Sistematic and As needed Program Understanding), Modelos Integrados (Strategies and Integrated Techniques). La comprensión *Bottom Up* implica la lectura del programa, luego la construcción de abstracciones y la incorporación de semántica a las mismas. Este proceso se repite hasta obtener un nivel de abstracción que permita entender el programa. La comprensión *Top Down* usa un proceso inverso a la comprensión bottom up. Esta técnica supone que el programador conoce algo acerca del dominio de la aplicación o la funcionalidad del programa. El programador formula hipótesis y luego lee el código para verificar esas hipótesis; en otras palabras comienza con un nivel de abstracción alto y luego construye niveles más bajos que le permiten verificar las hipótesis. El *Knowledge Based Understanding Model* incorpora una base de conocimiento y un modelo mental. La base de conocimiento consiste de

la comprensión del programador y el modelo mental codifica el entendimiento que tiene el mismo del programa. Este modelo puede usar aproximaciones bottom up o top down para el proceso de asimilación. El *Systematic and As-needed Program Understanding* está basado en la utilización de micro-estrategias y macro-estrategias. Las primeras permiten estudiar parte del código del programa mientras que las segundas hacen posible un análisis más global. Finalmente, las *Técnicas Integradas* son una combinación de las estrategias bottom up, top down y knowledge based.

Los modelos cognitivos pueden ser usados para la elaboración de criterios de evaluación. En [17] el lector puede ver diferentes criterios de evaluación que fueron obtenidos teniendo presente los modelos cognitivos mencionados. Esos criterios son clasificados con el fin de mejorar la comprensión de programas y reducir las demoras en el proceso cognitivo del programador. Usaremos esos criterios para evaluar diferentes herramientas de comprensión de programas. Además, definiremos nuevos criterios orientados a la evaluación de una clase más específica de algoritmos: los algoritmos de ruteo [7].

3. HERRAMIENTAS DE COMPRENSIÓN DE PROGRAMAS

En términos generales podemos decir que el software utilizado en el contexto de la comprensión de programas puede ser clasificado en dos grandes categorías: desarrollo en general y de propósito específico. La primera clase hace referencia a aquellas herramientas que son de desarrollo de sistemas en general, pero que poseen un conjunto de funcionalidades que facilitan el desarrollo de herramientas de comprensión como por ejemplo: Eclipse SDK, Visual Estudio .Net. La segunda hace referencia a herramientas destinadas específicamente al proceso de comprensión de programas como por ejemplo: SHriMP, Creole, Codesurfer, Imagix 4D y Jeliot 3. Este artículo está orientado al estudio de la segunda clase de software porque consideramos que, la exploración de sus características, nos permitirá desarrollar herramientas de comprensión más avanzadas y completas.

Herramientas Estudiadas

Según nuestro criterio, pensamos que las herramientas abajo descriptas son muy significativas para el estudio de esta temática debido a que poseen un conjunto bastante diverso de técnicas de comprensión [8]. En las siguientes subsecciones damos una breve descripción de cada una de ellas.

SHriMP

SHriMP es una aplicación diseñada para visualizar y explorar software desarrollado en el lenguaje java. SHriMP provee diferentes visiones del software por ejemplo: vistas anidadas de módulos, jerarquía de clases e interfaces, grafos de llamadas, dependencias de paquetes vía llamadas de métodos y accesos a campos, visión del sistema usando treemaps (una representación de árboles compacta y útil para mostrar sistemas), entre otras tantas características. Todas esas técnicas de visualización ayudan al programador a comprender un sistema. Además, SHriMP tiene facilidades para la navegación de software y diferentes formas de personalizar el ambiente. Todas esas características hacen de SHriMP una buena herramienta de comprensión de programas. Una particularidad de SHriMP es la posibilidad de obtener vistas. Esta peculiaridad es importante porque brinda más alternativas de comprensión.

Creole

Creole es el término usado para describir un plug in para Eclipse SDK [11], un ambiente de desarrollo de aplicaciones cuyo lenguaje de programación es java, que integra SHriMP con Eclipse SDK. Con Creole, podemos analizar códigos java visualmente viendo su estructura y enlaces (llamadas, accesos, entre otras) entre las diferentes piezas. En otras palabras esta aplicación posee las mismas funcionalidades que SHriMP y además se le incorporan las del ambiente de programación Eclipse.

Codesurfer

Codesurfer es una aplicación diseñada para reducir defectos y verificar si el software cumple con todos los requerimientos. Esta aplicación permite analizar sistemas escritos en C y C++ y algunas de las facilidades que presenta son: construye un grafo de llamadas de funciones, permite visualizar las llamadas a funciones realizadas a través de punteros, posibilita ver las variables y sus usos. Además se puede navegar fácil y eficientemente por el código fuente del sistema de estudio, posee herramientas de búsquedas eficaces y puede ser usada por distintos lenguajes de programación. Otra de las características importantes de esta aplicación es que puede ser invocada con las opciones específicas del compilador que se está usando. La precisión de las operaciones que realiza hace de ésta una herramienta destacable y de alta calidad. Por otra parte, puede ser ejecutada en distintas plataformas (Unix, Linux, Windows, entre otras).

Imagix 4D

Imagix 4D es una herramienta para la Ingeniería Inversa estática. Esta herramienta funciona en distintas plataformas (Unix, Linux, Windows, entre otras tantas) y posibilita el análisis de sistemas escritos en C y C++. En Imagix 4D existen ventanas que muestran lo siguiente: i) relaciones entre archivos, clases, funciones, variables, macros y tipos; ii) lugares donde los símbolos se encuentran ubicados dentro de la estructura y también posibilita ver el contenido de los archivos; iii) formas de navegar entre los distintos archivos del sistema de estudio; iv) resultados de búsquedas; v) navegación entre las distintas clases del sistema; vi) diagramas de flujos.

Las funciones provistas con Imagix 4D ayudan a diferentes tareas relacionadas con la comprensión de programas, por ejemplo: i) permite una rápida navegación a través del flujo de control; herencias y relaciones entre las clases; ii) facilita el estudio del programa por medio de componentes reusables y el proceso de construcción; iii) analiza el software de las siguientes maneras: focalizando en actividades de verificación; ubicando cuellos de botellas; iv) permite generar documentación actualizada y de código indocumentado.

Jeliot 3

Jeliot 3 [13] es una aplicación de visualización de la dinámica [15] de un programa escrito en lenguaje java. Esta herramienta fue pensada para los principiantes. La característica más interesante, desde nuestro punto de vista, es la máquina de animación. Con este mecanismo, los alumnos pueden ver los objetos del programa, los mensajes llamados, la evaluación de expresiones, entre otras aplicaciones, todas esas actividades son muy útiles para la comprensión de programas. Las animaciones muestran el comportamiento interno del sistema. Jeliot 3 es una herramienta simple si la comparamos con otras pero el atributo descrito previamente (las animaciones)

son difíciles de desarrollar y generalmente, no se incluye en aplicaciones más complejas porque requiere de aspectos más específicos del dominio del problema de estudio y de la implementación.

Evaluación de Herramientas

La evaluación de las herramientas se está llevando a cabo usando los criterios definidos por Storey, M; Fracchia, F; Müller, A; en [17]. Esos autores declaran que para evaluar herramientas de comprensión de programas el programador debe observar si las aplicaciones pueden: i) mejorar la comprensión de programas, ii) reducir el tiempo de cognición requerido por el programador. La primera característica intenta clasificar las herramientas de acuerdo al modelo cognitivo usado. La segunda considera las facilidades y complejidades de las herramientas de comprensión.

En lo siguiente, presentamos los criterios de evaluación que utilizaremos para el estudio de estas herramientas de comprensión.

Mejora de la Comprensión del Programa

- Aumento la comprensión bottom up
  - Indica las relaciones sintácticas y semánticas entre los distintos objetos de software (C1);
  - Reduce los efectos de planes deslocalizados (C2);
  - Provee mecanismos de abstracción (C3).
- Aumento de la Comprensión top down
  - Soporta comprensión dirigida por hipótesis orientada a objetivos (C4);
  - Provee visiones de la arquitectura del programa en distintos niveles de abstracción (C5).
- Integra las aproximaciones top down y bottom up
  - Soporta la construcción de múltiples modelos mentales (dominio, situación, programa) (C6);
  - Referencias cruzadas de modelos mentales (C7).

Reduce el tiempo de cognición del programador

- Facilita la navegación:
  - Provee navegación direccional (C8);
  - Soporta navegación arbitraria (C9);
  - Provee navegación entre diferentes modelos mentales (C10).
- Provee señales de orientación:
  - Le indica al programador la sección de trabajo corriente (C11);
  - Muestra el paso que conduce a la sección de trabajo corriente (C12);
  - Indica las opciones que conducen a los nodos corrientes (C13).
- Reduce la desorientación:
  - Reduce el esfuerzo para ajustarse a la interfaz de usuario (C14);
  - Provee estilos efectivos (C15).

Experiencias

Las experiencias que deseamos realizar con los algoritmos presentados consisten en evaluarlos utilizando los criterios descriptos en la sección anterior utilizando sistemas de pruebas. La finalidad de esta tarea es establecer un ranking de las distintas herramientas de forma tal que sea un aporte para el investigador que desea analizar, construir o mejorar las herramientas de comprensión de programas.

Básicamente el trabajo de evaluación [15] consistirá en una exploración sistemática de las funcionalidades y la medición de los criterios establecidos. Los resultados obtenidos serán anotados en una tabla que luego se analizará computando la cantidad de criterios que en promedio soporta la herramienta. Estos resultados nos permitirán establecer un ranking de las herramientas de comprensión estudiadas.

4. COMPRENSIÓN DE ALGORITMOS DE RUTEO

En esta sección discutiremos la necesidad de aplicar técnicas de comprensión a los algoritmos de ruteo. Luego presentaremos el modelo cognitivo que pensamos es el adecuado para la utilización de las estrategias de comprensión. Finalmente, definimos criterios de evaluación que nos permitirán construir o determinar la calidad de las herramientas de comprensión de algoritmos de ruteo.

Necesidad de Comprender los Algoritmos de Ruteo

En esta sección mostramos la necesidad de explicar operaciones internas cuando se intenta estudiar los algoritmos de ruteo [1][2][6]. Con este objetivo, tomamos como ejemplo al Ruteo por Brújula (RpB), una estrategia de ruteo reciente, y veremos que su especificación en términos matemáticos es simple, sin embargo su implementación requiere de distintas subrutinas que dificultan la comprensión del programa que lo implementa.

El ruteo por brújula [3][10] funciona como sigue: Suponga que $G=(N,R)$ es un grafo geométrico plano, es decir los nodos del conjunto N están conectados por segmentos de línea recta y no existen cruces entre los segmentos de G. Además, admita que el nodo u desea enviar un mensaje al nodo v. El nodo u sólo conoce las coordenadas de sus vecinos y del nodo destino. Si u usa el ruteo por brújula entonces envía el mensaje al vecino de él que minimiza el ángulo con respecto a la recta determinada por u y v. Este proceso se repite en cada nodo visitado hasta que se alcance el destino o bien se regrese a uno tratado previamente.

La función de selección del próximo nodo en el camino se puede definir matemáticamente como sigue:

$$RpB(u,V(u),v) = w_1 \in V(u) : \angle w_1uv < \angle w_2uv, \forall\, w_2 \in V(u) \quad (1)$$

Donde $V(u)$ es la operación que retorna como resultado los vecinos del nodo u.

Podemos observar que la definición (1) es simple y elegante. Sin embargo, su implementación necesita de otros mecanismos ocultos en la definición (1) como se puede observar en [10]. Esto nos conduce a pensar lo siguiente:

> *Una definición matemática elegante y sintética oculta estrategias que:*
> - *Imposibilitan tener una visión acabada del algoritmo, y*
> - *Requieren de muchas rutinas de código para su implementación.*

Estas características dificultan la comprensión de programas de este tipo a partir de su implementación.

Modelo posible para la Comprensión de Algoritmos de Ruteo

Básicamente un sistema de algoritmos de ruteo puede ser analizado usando las mismas estrategias utilizadas para otros sistemas. Sin embargo, creemos que esta clase de algoritmos necesita desarrollos visuales y de animación más avanzados. Por esta razón, el modelo cognitivo utilizado para entender este tipo de sistema debe incluir diferentes visiones. Estamos principalmente interesados en la visión operacional y comportamental [8]. Nos hemos percatado, a través del estudio del estado del arte, que existen diversos trabajos que abordan la visión operacional pero no sucede lo mismo con la visión comportamental.

Pensamos que el modelo cognitivo más apropiado para la comprensión de programas debe soportar ambas formas de aprendizajes: top down y bottom up, porque estimamos que el uso de una estrategia u otra depende de la estructura de pensamiento del programador y de las características de la aplicación que estudia. Por lo tanto, los modelos cognitivos posibles de usar son: Knowledge-based Understanding Model, Systematic and As-Needed Program Understanding Strategies y el Integrated Metamodel of Program Comprehension. Creemos que la última opción es la más atractiva porque hace posible entender programas en forma top down y bottom up. Además, este modelo tiene otros aspectos interesantes, por ejemplo la base de conocimiento que representa todo el trasfondo conceptual del programador antes de comenzar la tarea de mantenimiento, evolución o comprensión.

Criterios para Herramientas de Comprensión Algoritmos de Ruteo

En esta sección presentamos algunos criterios para evaluar o construir herramientas de comprensión de programas destinadas a estudiar los Algoritmos de Ruteo.

Nuestro estudio estará enfocado en el aspecto comportamental de los algoritmos de ruteo. Esto se debe a que la vista operacional puede ser obtenida usando las herramientas de comprensión de programas clásicas.

En este contexto podemos decir que la representación visual de las operaciones y las animaciones es muy importante. Consecuentemente, la herramienta de comprensión de programas debería tener:

1. Un sistema de visualización interactivo/automático.
2. Un modo de ejecución paso a paso.

El primer ítem puede ser usado para generar casos de pruebas aleatorios y específicos porque normalmente se desea estudiar el comportamiento, en términos generales, de esos algoritmos. Por otra parte, esas clases de algoritmos tienen actividades particulares que algunas veces determinan la diferencia entre una estrategia y otra. Esta es la razón de la necesidad de un modo interactivo de ejecución. La disponibilidad de un modo paso a paso nos permitirá estudiar los algoritmos en profundidad.

Las herramientas para estudiar algoritmos de ruteo tendrían que permitir visualizar las operaciones internas realizadas por las acciones de alto nivel. Este parámetro hace posible analizar la implementación de esas funciones.

3. La herramienta permite visualizar las operaciones de bajo nivel implicadas en las acciones de alto nivel mostradas por los sistemas de visualización de algoritmos de ruteo.

Esta característica es importante porque posibilita incorporar otras componentes al modelo cognitivo usado. Construir una herramienta con todas las operaciones es imposible. Sin embargo, podemos concebir que el modelo cognitivo tiene una base de conocimiento asociada a la aplicación (BCAA). En otras palabras, el sistema de comprensión de programas tendrá un conjunto de operaciones posibles de visualizar pero, al igual que los humanos, tendría asociados mecanismos de aprendizaje que posibilitarían incrementar la capacidad de BCAA. Esta característica nos permitiría construir un sistema cada vez más experto en la comprensión de algoritmos de ruteo. Consecuentemente otro criterio de evaluación es:

4. El sistema tiene la capacidad de incorporar nuevas primitivas para dejarlas disponibles para usos futuros.

Usualmente los humanos cuando resuelven un problema hacen tareas elementales y anotan los resultados parciales para potenciales usos futuros. Estos resultados pueden o no ser pertinentes por lo tanto no se necesita almacenarlos en una memoria permanente. Llamaremos a este almacenamiento Memoria de Trabajo. Por consiguiente, otro criterio de evaluación es:

5. El sistema tiene elementos visuales para almacenar resultados parciales.

Y otro derivado de éste:

6. El sistema tiene mecanismos para mostrarle al usuario los resultados parciales pertinentes al contexto del programa.

En ciertas ocasiones es necesario repetir alguna secuencia de animación para analizar situaciones claves. Por esta razón es deseable que el software de comprensión tenga esta alternativa.

7. El sistema tiene estrategias para repetir eventos.

Pensamos que para considerar óptima una herramienta de comprensión de programas de estas características es necesario que posea los requerimientos relacionados con la dinámica del programa. Además debe reunir los criterios para estudiar la visión estática del mismo.

5. CONCLUSIONES Y TRABAJOS FUTUROS

La comprensión de programas es un área de la Ingeniería del Software que presenta muchos desafíos de investigación. El desarrollo de sistemas en esta área implica un duro trabajo intelectual y de desarrollo. Podemos hacer esta afirmación porque observamos los requisitos que deben reunir las herramientas de comprensión de programas. También creemos que no existe una herramienta de comprensión de programas que satisfaga todos los requerimientos.

Por otra parte seleccionamos un conjunto de herramientas que consideramos significativas para analizar su calidad y establecer un ranking de las mismas.

Finalmente, hemos discutido en este artículo la posibilidad de aplicar técnicas de comprensión de programas a los algoritmos de ruteo. Centramos nuestros estudios en la visión comportamental de sistemas debido a que hemos detectado la carencia de trabajos en este ámbito.

Pensamos que los trabajos futuros deben estar orientados a perfeccionar el modelo cognitivo incorporando nuevas componentes que posibiliten la construcción de herramientas más sofisticadas que ayuden al programador a hacer inferencias. Por otra parte, los criterios de evaluación deben ser perfeccionados haciendo actividades experimentales que permitan tener conocimiento del impacto de las visiones comportamentales en la comprensión de este tipo de sistemas. Finalmente nuestros

esfuerzos están centrados en la construcción de aplicaciones de comprensión cada vez más completas y poderosas.

6. REFERENCIAS

[1]. Berón, M; Gagliardi, O; Hernández Peñalver, G. Evaluación de Métricas en Redes de Computadoras. *Congreso de Informática y Ciencias de la Computación CACIC 2003*. La Plata.

[2]. Berón, M; Gagliardi, O; Flores, S. Ruteo en Redes Inalámbricas. *Congreso Argentino de Ciencias de la Computación CACIC 2002*. Buenos Aires.

[3]. Berón, M; Gagliardi, O; Hernández Peñalver, G. Evaluación de Algoritmos de Ruteo de Paquetes en Redes de Computadoras. *Workshop de Investigadores en Ciencias de la Computación WICC 2004*. Rio Negro.

[4]. Berón, M; Gagliardi, O; Hernández Peñalver, G. Evaluación de Algoritmos de Ruteo en Redes de Computadoras. *V Workshop de Investigadores en Ciencias de la Computación WICC 2003*. Tandil.

[5]. Berón, M; Grosso, A; Gonzales, A; Printista, M; Maldocena, P; Ordoñes, G; Molina, S; Apolloni, R. Una Aproximación hacia el estudio de los Sistemas de Computación. *Workshop de Investigadores en Ciencias de la Computación WICC 2003*. Tandil.

[6]. Berón, M; Hernández Peñalver, G; Gagliardi, O. Factibilidad de Uso del Ruteo Voraz en los Grafos de Gabriel, Vecindad Relativa y Triangulaciones. *Congreso Argentino de Ciencias de la Computación CACIC 2004*. Buenos Aires.

[7]. Berón, M; Hernádez Peñalver, G; Gagliardi, O. Un Evaluador de Algoritmos de Ruteo. *Tesis de Maestría en Ingeniería del Software*. *Universidad Nacional de San Luís*. San Luis 2005.

[8]. Brooks, R. Using Behavioral Theory of Program Comprehension in Software Engineering. *Proceedings of the 3rd Intl. Conf. On Software Engineering, pág: 196201*. IEEE CS Press, 1978.

[9]. Grosso, A; Riesco, D; Berón, M. Una Herramienta para la Ingeniería Inversa de Sistemas Escritos en Lenguaje C, Bajo Linux. *Comunicación. Congreso Argentino de Ciencias de la Computación CACIC 2002*. Buenos Aires.

[10]. Hernández Peñalver, G; Berón, M; Gagliardi, O. Ruteo Geométrico Aplicado a las Redes de Computadoras. *Workshop de Investigadores en Ciencias de la Computación WICC 2005*. Córdoba.

[11]. Linter, R; Michand, J; Storey, M; Wu, X. Pluggingin Visualization: Experiences Integrating a Visualization Tool with Eclipse. *In Proceeding of 2003 ACM Symposium on Software Visualizaton, pág: 4756*. 2003.

[12]. Mayrhauser, A; Vans, M. Program Comprehension During Software Maintenance and Evolution. *Computer, vol. 28, no. 8, pp. 4455*. 1995.

[13]. Moreno, A; Myller, N; Satinen, E; BenAri, M. Visualizing Programs with Jeliot 3. *In the Proceeding of the International Working Conference on Advanced Visual Interfaces AVI* . 2004.

[14]. Oliveira, E; Henriques, P; Varanda, M. Características de um Sistema de Visualização para Compreensão de Aplicações Web através de Inspecção de Software e baseadas em Modelos Cognitivos. *Tesis de Maestría en Ingeniería del Software*. 2006. Portugal. Braga.

[15]. Pacione, M; Roper, M; Wood, M. A Comparative Evaluation of Dynamic Visualization Tools. *In the Proceedings of the 10th Working Conference on Reverse Engineering (WCRE'03), pág: 8089*. IEEE. 2003.

[16]. Sajaniemi, J. Program Comprehension through Multiple Simultaneous Views: A Session with VinEd. *In the Proceeding of 8th International Workshop on Program Comprehension, pág: 99108*. IEEE. 2000.

[17]. Storey, D; Fracchia, F; Müller, H. Cognitive Design Elements to Support the Construction of a Mental Model during Software Visualization. *In International Workshop on Program Comprehension, pág: 1728*. IEEE. 1997.