Comprensión de Programas

Mario M. Berón, Pedro Rangel Henriques, Roberto Uzal y Maria João Varanda Pereira
Ingeniería del Software: Conceptos Métodos Herramientas en un Contexto de Ingeniería de Software en Evolución
Universidad Nacional de San Luis - Argentina
Program Comprehension by Visual Inspection and Animation
Universidade do Minho – Braga – Portugal
Departamento de Informática
Universidad Nacional de San Luis

CONTEXTO

Las líneas de investigación descriptas en este artículo se encuentran enmarcadas en el contexto del proyecto: *Ingeniería del Software: Conceptos Métodos Herramientas en un Contexto de Ingeniería de Software en Evolución* de la Universidad Nacional de San Luis y del Proyecto *Program Comprehension by Visual Inspection and Animation* desarrollado en la Universidade do Minho, Portugal. Las líneas aquí presentadas surgen a partir de la realización de una tesis doctoral auspiciada por ambos proyectos y de los vínculos de investigación establecidos entre la Universidad Nacional de San Luis y la Universidade do Minho, Portugal.

RESUMEN

La *Comprensión de Programas* (CP) es una disciplina de la Ingeniería de Software cuyo objetivo es proveer *Modelos, Métodos, Técnicas* y *Herramientas* para facilitar el estudio y entendimiento de programas.

La construcción de estos productos de comprensión implica el estudio de disciplinas tales como *Modelos Cognitivos*, *Visualización de Software, Estrategias de Interrelación de Dominios* y *Métodos de Extracción de la Información*.

En este artículo se presentan líneas de investigación cuyos objetivos consisten en el análisis los productos de comprensión existentes y en la construcción de otros nuevos basados en los conceptos comunes a las grandes áreas mencionadas en el párrafo anterior.

Palabras Claves: Comprensión de Programas, Modelos, Métodos, Técnicas, Herramientas.

1. INTRODUCCION

La Comprensión de Programas es un área de la Ingeniería del Software destinada a elaborar *Modelos, Métodos, Técnicas* y *Herramientas,* basados en un proceso cognitivo y de ingeniería con el objetivo de facilitar el entendimiento de software.

El proceso cognitivo implica el estudio y análisis de las fases y pasos seguidos por los programadores para entender programas. Este tema es abordado a través de la investigación de los *Modelos Cognitivos* de CP.

El proceso de ingeniería incluye investigaciones sobre *Visualización de Programas, Estrategias de Interrelación de Dominios* y *Métodos de Extracción de la Información*. En este contexto interdisciplinario, el desarrollo de productos de compresión se basa en encontrar el común denominador a esas tres grandes disciplinas .

Actualmente existen muchos sistemas destinados a facilitar el entendimiento de software. Sin embargo, en muchas situaciones, no es claro como las teorías cognitivas, estrategias de viualización y extracción de la información se instancian en esas herramientas. Además, esas aplicaciones están centradas en analizar y presentar el código fuente del programa limitando CP a la Inspección de código. Como se verá en el desarrollo de este artículo, la Comprensión de Programas implica: i) plasmar claramente los conceptos de sus principales áreas en los productos de comprensión y ii) encontrar relaciones entre el *Dominio del Problema* y el *Dominio del Programa*, es decir, detectar las componentes de software utilizadas por el sistema para producir su salida [Bro78].

Este artículo está organizado de la siguiente manera. La sección 2 describe sintéticamente las líneas de investigación desarrolladas en nuestro proyecto. La sección 3 expone brevemente los resultados obtenidos en cada una de las temáticas después de un año de trabajo. Finalmente, la sección 4 menciona las estrategias planificadas para la formación de recursos humanos.

2. LINEAS DE INVESTIGACION

A continuación se exponen las líneas de investigación desarrolladas en nuestro proyecto.

2.1 Modelos Cognitivos

El término Modelo Cognitivo hace referencia a las estructuras de la información y estrategias de estudio usadas por los programadores para entender programas [BVDB93]. Los modelos cognitivos constan de diferentes componentes. Ellas son: el *Conocimiento,* un *Modelo Mental* y un *Proceso de Asimilación*.

Existen dos tipos de Conocimiento, el *Interno,* compuesto por el conjunto de conceptos y relaciones que conforman la estructura de conocimiento del programador; y el *externo* cuyos componentes son los nuevos conceptos proporcionados por el sistema de estudio.

El *Modelo Mental* se define como la representación mental que tiene el programador del sistema. El *Grafo de Funciones, Comunicaciones de Módulos,* etc. son posibles modelos mentales. Finalmente, el *Proceso de Asimilación* describe la estrategia utilizada por el programador para entender programas. Esta puede ser *top-down*, *bottom-up* o *híbrida.* Teniendo en cuenta esos elementos y sus relaciones muchos autores sostienen que: *un programador entiende un programa cuando puede encontrar las componentes de software usadas para producir la salida del sistema* [Bro82]. El camino adecuado para alcanzar este objetivo consiste en: i) Proveer representaciones para los dominios del problema y programa; ii) Definir un proceso que permita unir ambas representaciones. Los pasos mencionados previamente conforman la base para construir verdaderas aplicaciones de comprensión de programas.

Esta línea de investigación estudia el interjuego entre las diferentes componentes de los modelos cognitivos con el objetivo de elaborar estrategias que faciliten el entendimiento del software.

2.2 Visualización de Software

La Visualización de Software [BkK01a,Che06, Hen01] es una disciplina de la Ingeniería del Software cuyo objetivo es mapear ciertos aspectos de software en una o mas representaciones multimediales. Para alcanzar este objetivo es necesario la interacción con otras áreas del conocimiento tales como: *Diseño Gráfico*, *Psicología Cognitiva* y otras disciplinas directamente relacionadas con la elaboración de efectos multimediales [Ext02]. Si la visualización esta orientada a la comprensión de programas, el principal desafío consiste en construir vistas que permitan relacionar el Dominio del Problema con el Dominio del Programa.

Existen innumerables herramientas de visualización de programas que, según sus autores tienen como finalidad facilitar la comprensión de programas [BkK01b]. Sin embargo, la gran mayoría propone visualizaciones concernientes con el Dominio del Programa (por ejemplo *Funciones, Módulos, Variables,* etc.) dejándo de lado dos importantes componentes como lo son el Dominio del Problema y su relación con el Dominio del Programa.

Esta línea de investigación estudia las visualizaciones que relacionan el Dominio del Problema con el Dominio del Programa y la integración de este tipo de vista con las visualizaciones tradicionales [LELP06].

2.3 Métodos de Extracción de la Información

El desarrollo de técnicas de extracción de la información estática es importante porque permite recuperar todos los atributos de los objetos definidos en el programa.

El desarrollo de técnicas de extracción de la información dinámica es relevante porque posibilita conocer cuales son las componentes del programa utilizadas para una ejecución específica del sistema.

Ambos tipos de información son necesarias para la elaboración de técnicas de comprensión que faciliten el estudio de sistemas grandes. Esta tarea es uno de los principales objetivos de nuestro grupo de investigación.

Esta línea de investigación tiene por objetivo el estudio y elaboración de técnicas de extracción de información estática y dinámica desde los sistemas de software [CCMT93,CHZ+07,KGG05].

2.4 Estrategias de Interrelación de Dominios

Una de las formas de facilitar la comprensión de grandes sistemas consiste en relacionar el Dominio del Problema (es decir el comportamiento del programa) con el Dominio del Programa (o sea la operación del programa). Esto se debe a que dicha relación permite que el programador pueda localizar rápidamente los objetos del programa que se

utilizaron para una funcionalidad específica del sistema de estudio.

Si el sistema que se está analizando es pequeño (5 Kloc o menos) entonces las bondades de este tipo de relación no se pueden apreciar claramente. Pero si el sistema es de gran envergadura la relación antes mencionada disminuye el esfuerzo del programador porque él puede identificar rápidamente los objetos sobre los cuales debe concentrar su estudio. Por otra parte, los costos implicados en la modificación, mantenimiento o evolución del sistema se disminuyen debido a que se decrementa el tiempo requerido por el programador para la realización de la tarea. Es importante mencionar que son casos excepcionales las herramientas de comprensión profesionales que incluyen este tipo de relación. Esta característica hace que esta temática sea de suma importancia en el contexto de CP debido a la posibilidad que se presenta para realizar contribuciones. Además del desarrollo de estrategias para interconectar los Dominios del Problema y Programa se estudian formas de integrar esta importante relación con vistas tradicionales como por ejemplo: *Grafo de Funciones*, *Grafo de Módulos*, *Grafo de Tipos*, etc.

2.5 Evaluación de Herramientas de Comprensión

En nuestro grupo de investigación se ha realizado un estudio integral de las herramientas de comprensión abarcando todas las temáticas involucradas en su creación, las cuales han sido sintéticamente descriptas en las secciones previas. Cada temática introduce una serie de criterios que permiten evaluar la calidad y pertinencia de cada uno de los sub-sistemas que componen a una herramienta de comprensión [GC99]. En esta línea de investigación se estudia la forma más adecuada de integrar todos esos criterios en un marco de evaluación que posibilite un análisis acabado de las herramientas de comprensión.

3. RESULTADOS OBTENIDOS/ESPERADOS

En las siguientes sub-secciones se describen los resultados obtenidos en cada una de las líneas de investigación descritas en la sección 2.

3.1 Modelos Cognitivos

Los resultados obtenidos en este contexto están relacionados con la formalización de los conceptos utilizados para describir las distintas formas que el programador utiliza para comprender programas. Estos conceptos son ambiguos y difíciles de entender por parte de la comunidad de Ciencias de la Computación porque los mismos son descritos por profesionales de otras áreas, como por ejemplo la Psicología Cognitiva. Por esta razón se realizó un esfuerzo en tratar de formalizar los conceptos dándoles un significado más preciso. Estas formalizaciones tienen como objetivo permitir el desarrollo de técnicas que posibiliten traducir el código fuente del sistema en una especificación basada en modelos con alto poder descriptivo. Está última tarea (el desarrollo de técnicas) es un resultado esperado para los futuros pasos de la investigación en esta área. El lector interesado en ver las formalizaciones de los conceptos puede leer [BHVU08].

3.2 Visualización de Software

Los *Sistemas de Visualización de Software Orientados a la Comprensión de Programas* son una nueva clase de sistemas que surge a partir de la investigación de los *Sistemas de Visualización* en general. La principal diferencia entre este nuevo tipo de sistemas y los tradicionales radica en que los Sistemas de Visualización Orientados a la Comprensión de Programas contemplan el Dominio del Problema y su relación con el Dominio del Programa. Esta peculiaridad permitió detectar la ausencia, en las clasificaciones actuales, de criterios para caracterizar el Dominio del Problema y la relación existente con el Dominio del Programa [Mye90,PSB92]. Esta debilidad permitió desarrollar una nueva clasificación de los Sistemas de Visualización Orientados a la Comprensión que define nuevos criterios que solucionan el inconveniente presentado previamente. Es importante notar que este resultado es relevante porque es original y por consiguiente es una contribución en el contexto de CP. El lector interesado en conocer esta nueva clasificación puede leer [BHVU08].

Los trabajos futuros en este ámbito están relacionados con la creación de estrategias de visualización que cumplan con la mayoría de los criterios declarados en la nueva taxonomía elaborada por nuestro grupo de investigación. Además se pretende que dichos esquemas de visualización sean incorporados en las herramientas de comprensión de nuestra propiedad con el objetivo de realizar estudios experimentales que permitan

analizar el grado de mejora alcanzada por los programadores en el proceso de comprensión.

3.3 Extracción de la Información

Para la extracción de la información dinámica se definió un esquema de *Instrumentación de Código*. Esta técnica consiste en insertar sentencias dentro del código fuente del sistema de estudio con la finalidad de recuperar las componentes del programa que se utilizaron para producir la salida. Para implementar una estrategia de estas características es necesario responder a los siguientes interrogantes: i) ¿Cuáles son los puntos del programa candidatos intrumentar? y ii) ¿Qué información debe ser recuperada? Teniendo en mente esas preguntas, se seleccionaron como puntos de inspección el inicio y fin de cada función del sistema. La razón de esta decisión se basa en que en esos lugares del programa se puede obtener información resumida acerca de las componentes del programa. Por ejemplo, se pueden conocer las funciones utilizadas, sus parámetros y si se desea ser más preciso los datos (valores de las variables globales y de los parámetros) que son utilizados por la función. Esta aproximación, aún recuperando parte de las operaciones y datos utilizados por el programa, tiene el inconveniente de extraer una enorme cantidad de información. Por este motivo, es necesario el empleo de técnicas de control de las iteraciones. Una de las estrategias elaboradas por nuestro grupo de investigación consistió en insertar sentencias antes, dentro y después de las iteraciones. Las sentencias previas a la iteración colocan en una pila de control el número de veces que las funciones invocadas dentro de la iteración pueden ser recuperadas. Las sentencias dentro del loop decrementan ese valor en uno. Cuando el valor del tope de la pila es cero las sentencias insertadas no recuperan mas información. Finalmente, las instrucciones insertadas después de la iteración suprimen el valor del tope de la pila. El lector interesado en conocer todos los pormenores de la estrateiga de instrumentación puede leer [BHVU08]. Los trabajos futuros en esta temática se relacionan con el desarrollo de nuevos esquemas de instrumentación de código y técnicas de análisis de la información dinámica.

3.4 Estrategias de Interrelación de Dominios

Se desarrollaron dos técnicas para la interrelación de dominios, que utilizan los conceptos extraídos de las investigaciones presentadas en las secciones previas. Una de ellas denominada SVS (**S**imultaneous **V**isualization **S**trategy) se basa en la ejecución paralela del sistema instrumentado y del administrador de funciones de inspección (un programa que implementa las acciones de las sentencias incorporadas en el código fuente del sistema). Esta característica permite que las componentes de software usadas sean mostradas cuando el sistema está en ejecución.

La otra estrategia es BORS (**B**ehavioral-**O**perational **R**elation **S**trategy), este procedimiento, al igual que SVS, utiliza la información reportada por el esquema de instrumentación pero de una manera diferente. BORS requiere que el sistema sea ejecutado, después de eso se procesa la información y se construyen algunas estructuras de datos útiles para construir explicaciones, como por ejemplo el árbol de ejecución de funciones. Luego se realizan algunas consultas sobre dichas estructuras para recuperar alguna información relacionada con **l**os objetos del dominio del problema. Como trabajo futuro se pretende el desarrollo de otras estrategias de interrelación de dominios que permitan extraer más información de tiempo de ejecución.

3.5 Evaluación de Herramientas de Comprensión

Los resultados obtenidos en esta línea de investigación no son tan avanzados como los obtenidos en las otras líneas descriptas en las sub-seciones precedentes. No obstante se puede mencionar que se optó por adaptar una metodología de evaluación utilizada para sitios web que usa una serie de criterios definidos para este tipo de software y define un proceso de ranking que posibilita establecer que sitio web es el mejor [OGLR99,OR02]. La elección de este camino se debió a que el proceso de ranking de esta metodología hace un fuerte uso de los atributos del objeto de estudio los cuales son dificiles de definir. Como la definición de los atributos de las herramientas de comprensión ha sido realizada para cada uno de sus componentes en las otras líneas de investigación que conforman nuestro proyecto, entonces los mismos sirven como entrada al proceso de ranking definido por la metodología de evaluación seleccionada. Se espera poder aplicar este proceso de evaluación a varias herramientas de comprensión para establecer un ranking y analizar

los resultados obtenidos con los nuevos criterios definidos por nuestro grupo invetigación.

4. FORMACIÓN DE RECURSOS HUMANOS

Los resultados alcanzados en este momento están relacionados con la realización de una tesis doctoral denominada: "Inspección de Programas para Interconectar las Vistas Operacional y Comportamental para la Comprensión de Programas", desarrollada con entre la Universidad Nacional de San Luis y la Universidad de Minho, Portugal. Además de la realización de la tesis doctoral se proyectan la elaboración de diferentes publicaciones nacionales e internacionales como así también el desarrollo de tesis de licenciatura, maestría y doctorado. Todas las tareas mencionadas previamente serán realizadas con el auspicio de la UNSL y la Universidade do Minho.

5. BIBLIOGRAFIA

[BkK01a] Sarita Bassil and Rudolf k. Keller. A Qualitative and Quantitative Evaluation of Software Visualization Tools. Proc. of the IEEE Symposium on Information Visualization, pages 69–75, 2001.

[BkK01b] Sarita Bassil and Rudolf k. Keller. Software Visualization Tools: Survey and Analysis. Proc. of Program Comprehension, 2001. IWPC 2001, pages 7–17, 2001.

[Bro78] R. Brook. Using a behavioral theory of program comprehension in software engineering. Proceedings of the 3rd international conference on Software engineering, pages 196–201, 1978.

[Bro82] R. Brook. A theoretical analysis of the role of documentation in the comprehension of computer programs. Proceedings of the 1982 conference on Human factors in computing systems, pages 125–129, 1982.

[BVDB93] K. Bertels, P. Vanneste, and C. De Backer. A cognitive approach to program understanding. Reverse Engineering, 1993., Proceedings of Working Conference on, pages 1–7, 1993.

[CCMT93] G. Canfora, A. Cimible, M. Munro, and C. Taylor. Extracting abstract data types from C programs: A case study. Proceedings, 12th IEEE International Conference on Software Maintenance, 27:100–209, 1993.

[Che06] Chaomei Chen. Information Visualization. Springer Verlag, 2006.

[CHZ+07] Bas Cornelisse, Danny Holten, Andy Zaidman, Leon Moonen, Jarke Wijk, and Arie Deursen. Understanding execution traces using massive sequence and circular bundle views. In Delft University of Technology Software Engineering Research Group Technical Report Series, pages 2–9. Delft University, 2007.

[Ext02] C. Exton. Constructivism and program comprehension strategies. Program Comprehension, 2002. Proceedings. 10th International Workshop on, pages 281–284, 2002.

[GC99] Gerald C. Gannod and Betty H. C. Cheng. A framework for classifying and comparing software reverse engineering and design recovery techniques. In WCRE '99: Proceedings of the Sixth Working Conference on Reverse Engineering, page 77, Washington, DC, USA, 1999. IEEE Computer Society.

[BHVU08] Mario Berón, Henriques Pedro, Maria João Varanda, Roberto Uzal. Inspección de Programas para Interconectar las Vistas Comportamental y Operacional para la Comprensión de Programas. Reporte de Tesis Doctoral. UNSL.

[Hen01] Gómez Henriques. Software Visualization: an Overview. Informatik, 2:4–7, 2001.

[KGG05] A. Kuhn, O. Greevy, and T. Girba. Applying Semantic Analysis to Feature Execution Traces. Program Comprehension through Dynamic Analysis, 1:48–53, 2005.

[LELP06] W. Lowe, M. Ericsson, J. Lundber, and T. Panas. Software Comprehension-Integrating Program Analysis and Software Visualization. Technical Report, 2006.

[Mye90] B. Myers. Taxonomies of Visual Programming and Program Visualization. Journal of Visual Languages and Computing, 1(1):97–123, 1990.

[OGLR99] Luis Olsina, Daniela Godoy, Guillermo Lafuente, and Gustavo Rossi. Assessing the Quality of Academic Web Sites: a Case Study. New Review Hypermedia Journal, 05:81– 103, 1999.

[OR02] Luis Olsina and Gustavo Rossi. Measuring Web Application Quality with WebQEM. IEEE MultiMedia, 2002, 09(4):20–29, 2002.

[PSB92] BA Price, IS Small, and RM Baecker. A taxonomy of software visualization. System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference on, 2, 1992.