Estrategias para Facilitar la Comprensión de Programas

Mario M. Berón Daniel Riesco Germán Montejano
Departamento de Informática-Facultad de Ciencias Físico Matemáticas y Naturales
Universidad Nacional de San Luis
{mberon,driesco,gmonte}@unsl.edu.ar

Pedro R. Henriques
Departamento de Informática-Universidade do Minho
pedrorangelhenriques@gmail.com

Maria J. Pereira
Departamento de Informática-Instituto Politécnico de Bragança
mjoao@ipb.pt

RESUMEN

La *Comprensión de Programas* es una disciplina de la Ingeniería de Software cuyo objetivo es proveer *Modelos, Métodos, Técnicas* y *Herramientas* para facilitar el estudio y entendimiento de los sistemas de software.

A través de un extenso estudio y experiencia en el desarrollo de productos de comprensión se pudo comprobar que el principal desafío en esta área consiste en: *Relacionar el Dominio del Problema con el Dominio del programa.* El primero hace referencia a la salida del sistema. El segundo a las componentes de software usadas para producir dicha salida. La construcción de este tipo de relación es muy compleja e 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 son:

- Analizar los productos de comprensión existentes.
- Construir productos innovadores basados en los conceptos comunes a las grandes áreas mencionadas en el párrafo anterior.

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

CONTEXTO

Las líneas de investigación descriptas en este artículo se encuentran enmarcadas en el contexto del proyecto: *Ingeniería del Software: Conceptos Métodos Técnicas y Herramientas en un Contexto de Ingeniería de Software en Evolución* de la Universidad Nacional de San Luis.

Las líneas aquí presentadas surgen a partir de: i) La realización de una tesis doctoral auspiciada por la Universidad Nacional de San Luis y la Universidade do Minho, Portugal y ii) El inicio de diferentes tesis de licenciatura que abordan la problemática de Comprensión de Programas.

1. INTRODUCCIÓN

La Comprensión de Programas (CP) 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 propósito 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 fue abordado en la investigación relacionada con los Modelos Cognitivos presentada por Henriques, et. al. en [BHUV08].

El proceso de ingeniería incluye investigaciones sobre *Visualización de Programas, Estrategias de Interrelación de*

Dominios y *Métodos de Extracción de Información*.

Actualmente existen muchos productos destinados a facilitar la comprensión de software. Sin embargo, en muchas situaciones, no es claro como las teorías cognitivas, estrategias de visualización y extracción de la información se instancian en dichos productos. Como se verá en el desarrollo de este artículo, la Comprensión de Programas implica:

- Plasmar claramente los conceptos de sus principales áreas en los productos de comprensión.
- 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 el proyecto. La sección 3 expone brevemente los resultados obtenidos/esperados en cada una de las temáticas abordadas por los investigadores. Finalmente, la sección 4 menciona los trabajos planificados para la formación de recursos humanos.

2. LÍNEAS DE INVESTIGACION

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

2.1 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 más 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 está 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. Sin embargo, la gran mayoría propone visualizaciones concernientes con el Dominio del Programa (por ejemplo *Funciones, Módulos, Variables,* etc.) dejando 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 visualizaciones tradicionales [LELP06].

2.2 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 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 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].

2.3 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 (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. El sustento de esta afirmación se basa en la posibilidad de 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 disminuyen debido a que se reduce 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.

Esta línea de investigación tiene los siguientes objetivos:

- Elaborar estrategias para interconectar los Dominios del Problema.
- Estudiar y crear de formas de integrar la relación mencionada en el ítem previo con otras vistas tradicionales como por ejemplo: *Grafo de Funciones*, *Grafo de Módulos*, *Grafo de Tipos*, etc.

3. RESULTADOS

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

3.1 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 de Software* 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. En este contexto, se pudo elaborar una serie de criterios que esta clase de sistemas deben reunir para presentar visualizaciones de software que ayuden al programador a entender programas [Mye90, PSB92, BHUV09].

A través de la experiencia en la construcción de visualizaciones, se pudo detectar la existencia de numerosas librerías gráficas que son útiles para la construcción de vistas de software que reúnen los requisitos establecidos por los criterios referenciados en el párrafo precedente. Entre las más relevantes se pueden mencionar: *Jung*, *Prefuse*, *Graphviz* y *Cairo*. Es importante notar que la selección de la librería más adecuada para visualizar software implica una reflexión profunda por parte del diseñador de la visualización. Con el propósito de facilitar esta tarea se ha propuesto, como trabajo de licenciatura, la construcción de una aplicación que permita establecer un ranking de este tipo de software. Este ranking se basa en los criterios definidos por los autores de este artículo en [BHVU08] y en otros que surgirán de la experiencia en la construcción de visualizaciones utilizando las librerías propuestas. Se piensa que los resultados de esta investigación serán obtenidos después de un año de trabajo.

3.2 Extracción de la Información

Es bien conocido que es posible extraer información Estática y Dinámica desde los sistemas de software. La primera hace referencia a: *tipos*, *variables*, *funciones*, etc. definidas en el código fuente del programa. La segunda se centra en la recuperación de las componentes de software usadas en tiempo de ejecución.

Para la extracción de información estática se utilizan técnicas de compilación tradicionales. Es decir se construyen un analizador sintáctico con las acciones semánticas apropiadas para extraer la información deseada.

Para la extracción de la información dinámica se definió un esquema de *Instrumentación de Código*. Esta técnica consiste en la inserción de 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. En este contexto, se definió un esquema de instrumentación para programas escritos en lenguaje C. Este esquema inserta funciones de inspección que muestran los nombres de las funciones ejecutadas por el sistema y otra información que el programador considere importante (el lector que desee profundizar este tema puede estudiar [BHVU08]). Se piensa que este esquema de instrumentación no sólo es aplicable al paradigma de programación imperativo. Teniendo presente esta observación, la investigación está centrada en extender el esquema de instrumentación para que el mismo pueda ser aplicado a lenguajes orientados a objetos. Para llevar a cabo esta tarea se ha tomado como caso de estudio al lenguaje de programación Java.

3.3 Estrategias de Interrelación de Dominios

Se desarrollaron dos estrategias para la interrelación de dominios. Cada una de ellas utiliza los conceptos extraídos de las investigaciones presentadas en las secciones previas (visualización e instrumentaciones de código).

Una estrategia es SVS (Simultaneous Visualization Strategy). Esta aproximación 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 en tiempo de ejecución sean visualizadas mientras el sistema se está ejecutando.

La otra estrategia es BORS (Behavioral-Operational Relation Strategy), 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 dinámica, recuperada por el esquema de instrumentación, 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 los objetos del dominio del problema.

Ambas estrategias presentan problemas que se deben abordar para simplificar el proceso de comprensión de programas.

En el caso de SVS, no se pueden obtener explicaciones debido a la forma en la que se representa la información dinámica (una lista).

En el caso de BORS, existe una fase de inspección manual del Dominio del Problema que introduce complejidades en el proceso de comprensión.

Con el propósito de solucionar los problemas mencionados en los párrafos precedentes se elaboró, a nivel conceptual, la estrategia SVSi (Simultaneous Visualization Strategy Improved). Dicha estrategia es una combinación de SVS y BORS que:

- Reduce el tiempo requerido por la inspección manual del Dominio del Problema a través de la utilización de información de tiempo de ejecución provista por SVS.
- Permite realizar explicaciones por medio de la utilización de las estructuras de datos provistas por BORS.

4. FORMACIÓN DE RECURSOS HUMANOS

Actualmente la línea de investigación se encuentra compuesta por cinco investigadores. Uno de ellos actúa como supervisor de los trabajos de los integrantes del equipo y los restantes están abocados al estudio de cada una de las áreas de trabajo descriptas en las secciones 2.1, 2.2 y 2.3.

En cuanto a formación de recursos humanos se puede decir que se encuentran en estado inicial los siguientes trabajos: i) Instrumentación de Sistemas Orientados a Objetos; ii) Comparación de Librerías de Visualización de Software y iii) Elaboración de Estrategias de Interconexión de Dominios basadas en Ontologías.

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.

[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.

[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.

[PSB92] BA Price, IS Small, and RM Baecker. A taxonomy of software visualization. System Sciences, 1992. Proceedings of the TwentyFifth HawaiiInternational Conference on, 2, 1992.

[BHUV09] Berón, M; Henriques, P; Uzal, R; Varanda, M. Comprensión de Programas. XI Workshop de Investigadores en Ciencias de la Computación. San Juan. 2009.