Inspección de Código para relacionar los Dominios del Problema y Programa para la Comprensión de Programas

Mario M. Berón Roberto Uzal
Universidad Nacional de San Luis - Departamento de Infomática
San Luis - Argentina
mberon@unsl.edu.ar, ruzal@uolsinectis.com.ar

Pedro R. Henriques
Universidade do Minho - Departamento de Informática
Braga - Portugal
prh@di.uminho.pt

Maria J. Varanda Pereira
Instituto Politécnico de Bragança
Bragança - Portugal
mjoao@ipb.pt

Resumen

La Comprensión de Programas es una disciplina de la Ingeniería de Software cuyo objetivo es proveer métodos, técnicas y herramientas para facilitar el estudio y entendimiento de programas.
La construcción de estos productos de comprensón implica el estudio de disciplinas tales como *Ciencias Cognitivas*, *Visualización de Software* y *Métodos de Extracción de la Información*.
En este artículo se presenta una línea de investigación cuyo objetivo es analizar productos de comprensión existentes y construir otros nuevos basados en los conceptos comunes a las tres grandes áreas mencionadas en el párrafo anterior.
Palabras Claves: Comprensión de Programas, Métodos, Técnicas, Herramientas.

1. Introducción

La Comprensión de Programas es un área de la Ingeniería del Software destinada a elaborar 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 Comprensión de Programas*.

El proceso de ingeniería incluye investigaciones sobre *Visualización de Programas* 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 [BHU07].

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 visualización y extracción de la información son plasmadas en esas herramientas. Además, esas aplicaciones están centradas en analizar y presentar el código fuente del programa limitando a la Comprensión de Programas a la inspección de código. Como se verá en el desarrollo de este artículo, la Comprensión de Programas implica, además de plasmar claramente los conceptos comunes de sus principales áreas constituyentes, 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 [LF94].

Este artículo está organizado de la siguiente manera. La sección 2 presenta los estudios realizados en el contexto de los Modelos Cognitivos. La sección 3 conceptualiza visualización de programas y describe las características que un sistema de visualización debe poseer. La sección 4 explica algunas técnicas de extracción de la información útiles para implementar estrategias de comprensión. La sección 5 expone algunas ideas para la elaboración de estrategias de interconexión de dominios. Finalmente, la sección 6 exhibe las conclusiones de este trabajo.

2. Modelos Cognitivos de Comprensión de Programas

Los término Modelo Cognitivo [Sto98] hace referencia a las estructuras de la información y estrategias de estudio usadas por los programadores para entender programas [Wal02]. 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 él puede encontrar las componentes de software usadas para producir la salida del sistema* [O'B03] [Tie89]. En otras palabras cuando es posible relacionar los dominios del problema y programa. El camino adecuado para alcanzar este objetivo consiste en:

1. proveer representaciones para los dominios del problema y programa
2. definir un proceso de que permita unir ambas representaciones.

Los pasos mencionados previamente conforman la base para construir verdaderas aplicaciones de comprensión de programas. En consecuencia, ellos deben ser tenidos presente para el diseño de cada una de las partes constituyentes, es decir, para visualizaciones de programas y métodos de extracción de la información.

3. Visualización de Software

La Visualización de Software es una disciplina de la Ingeniería del Software cuyo objetivo es mapear ciertos aspectos de software en una o mas representaciones multimediales [SDBP98] [Che06] [PdQ06]. 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. 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. 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.

Este problema se debe a la ausencia de una clara concepción de Comprensión de Programas y de un modelo de comprensión. Afortunadamente, nuestra investigación en el contexto de los Modelos Cognitivos y del estudio del estado del arte de herramientas de comprensión, nos permitió resolver este inconveniente en un estado inicial de la investigación (ver sección 2).

Esta debilidad en las teorías de visualización de programas posibilitó encontrar una nueva clase de sistemas de visualziación denominada: *Sistemas de Visualización de Software Orientados a la Comprensión de Programas*. Esta clase de sistema posee: i) Representaciones del dominio del problema, ii) Representaciones del dominio del programa y iii) Estrategias para visualizar la relación entre ambos dominios. Además de estas características, este tipo de aplicación tiene operaciones para manipular cada una de esas representaciones.

Otro factor derivado de la carencia de una definición y modelo de comprensión fue la imposibilidad de las taxonomías existentes para describir adecuadamente los sistemas de visualización orientados a la comprensión. Para detectar este problema fue necesario analizar la gran mayoría las taxonomías actuales y ver de que forma ellas caracterizaban el dominio del problema y la relación de este con el dominio del programa. Esta tarea reveló que las taxonomías están fuertemente orientadas a caracterizar solamente el dominio del programa. Teniendo en cuenta esta observación, se seleccionó la taxonomía más reconocida y se procedió a extenderla para que considere las componentes restantes de nuestra concepción de Comprensión de Programas. El lector interesado en estas investigaciones puede ver descripciones más detalladas en [BHU07].

4. Métodos de Extracción de la Información

Para implementar la técnicas basadas en Modelos Cognitivos y Visualización de Software fue necesario extraer información estática y dinámica del sistema de estudio [EKS] [RD94].

Para la recuperación de la información estática se utilizaron técnicas de compilación tradicionales para extraer información de cada componente del programa. Esta tarea fue llevada a cabo con el objetivo de implementar procedimientos de interrelación de dominios, por esta razón se evitó el uso de análisis muy complejos, como lo es el seguimiento de punteros, que si bien son muy interesantes e importantes dejan de lado del principal objetivo que consiste en relacionar los dominios del problema y programa.

Para la extracción de la información dinámica se definió un esquema de Instrumentación de Código [BHVU06a]. 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 a intrumentar? y ii) Que 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 al loop 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.

5. Estrategias de Interconexión de Dominios

Dos técnicas para la interconexión de dominios, que utilizan los conceptos extraídos de las investigaciones presentadas en las secciones previas, están siendo desarrolladas. Una de ellas denominada SVS (Simultaneous Visualization Strategy) [BHU07] 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 (Behavioral-Operational Relation Strategy) [BHVU06b], 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 la información es procesada y algunas estructuras de datos útiles para construir explicaciones, como por ejemplo el árbol de ejecución de funciones, deben ser construidas. Luego se realizan algunas consultas sobre dichas estructuras para recuperar alguna información relacionada con los objetos del dominio del problema.

6. Conclusión

La construcción de aplicaciones de Comprensión de Programas implica encontar e implementar los conceptos comunes a disciplinas tales como *Modelos Cognitivos*, *Visualización de Software* y *Métodos de Extracción de la Información*. Para alcanzar este objetivo se requiere realizar investigaciones profundas en cada una de esas disciplinas.

En este artículo se presentó el estado actual de esas investigaciones y se describieron, en términos generales, algunos resultados parciales obtenidos tales como: una conceptualización de Comprensión de Programas, un Modelo de Comprensón, la dectección de una nueva clase de sistema de Visualización de Software, la creación de un esquema de instrumentación de código y la propuesta de dos estrategias de inteconexión de dominios.

La dirección furtura de esta línea de investigación consiste en elaborar representaciones más robustas para los dominios del problema y programa de forma tal de poder obtener una interconexión de dominios más precisa. Además es necesario investigar y proponer otras estrategias de interconexión que utilicen abordajes diferentes a SVS y BORS, con el objetivo de realizar evaluaciones de desempeño. Finalmente, otras aristas de este proyecto están centradas en investigar la sistematización de los conceptos de Modelos Cognitivos para facilizar su implementación, elaboración innovadoras vistas de software y análisis de métodos de extracción de la información más sofisticados.

Referencias

[BHU07] M. Berón, P. Henriques, and R. Uzal. Program Inspection to interconnect Behavioral and Operational Views for Program Comprehension. *Technical Report*, 2007.

[BHVU06a] M. Beron, P. Henriques, M. Varanda, and R. Uzal. A Language Processing Tool for Program Comprehension. *Congreso Argentino de Ciencias de la Computacion (CACIC06)*, 2006.

[BHVU06b] M. Beron, P. Henriques, M. Varanda, and R. Uzal. Static and Dynamic Strategies to Understand C Programs by Code Annotation. *European Joint Conferences on Theory and Practice of Software (ETAPS07)*, 2006.

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

[EKS] T. Eisenbarth, R. Koschke, and D. Simon. Aiding program comprehension by static and dynamic feature analysis.

[LF94] H. Lieberman and C. Fry. Bridging the gulf between code and behavior in programming. In *ACM Conference on Computers and Human Interface*, Denver, Colorado, April 1994.

[O'B03] Micheal P. O'Brien. Sofware Comprehension - A Review and Research Direction. *Technical Report*, 2003.

[PdQ06] Marian Petre and Ed de Quincey. A Gentle Overview of Software Visualization. *PPIG: Psychology of Programing Interest Group*, pages 1–10, 2006.

[RD94] S. Rifkin and L. Deimel. Applying Program Comprehension Techniques to Improve Software Inspections. *Proceedings of the 19th Annual NASA Software Engineering Laboratory Workshop, Greenbelt, MD, Nov*, 1994.

[SDBP98] J. Stasko, J. Domingue, M. Brown, and B. Price. *Software Visualization: Programming as a Multimedia Experience*. The MIT Press, 1998.

[Sto98] Margaret A. Storey. *A Cognitive Framework for Describinh and Evaluating Software Exploration Tools*. PhD thesis, Simon Fraser University, 1998.

[Tie89] Tim Tiemens. Cognitive Model of Program Comprehension. *Technical Report*, 1989.

[Wal02] Andrew Walestein. *Cognitive Support in Software Engineering Tools: A Distributed Cognitive Framework*. PhD thesis, Simon Fraser University, 2002.