Título: "Análisis de Identificadores para Abstraer conceptos del Dominio del Problema"

Autor: Javier Azcurra Marilungo.

Dirección:

- **Director:** Dr. Mario Marcelo Berón.
- **Co-Director:** Dr. Germán Antonio Montejano.

Objetivos: Los objetivos principales de este Trabajo Final de Licenciatura son:

- Extraer identificadores, comentarios y literales de programas escritos en JAVA.
- Analizar los identificadores capturados con ayuda de los comentarios y literales extraídos.
- Construir en JAVA una herramienta que implemente los ítems anteriores.
- Evidenciar una aproximación que relacione la salida de un programa con los elementos involucrados que producen dicha salida.

Antecedentes

La Ingeniería del Software (IS) es una disciplina encargada de desarrollar sistemas de software de alta calidad. Para conseguir esta meta, la IS abarca, entre otras, tres temáticas importantes: el mantenimiento de software, la evolución de software y la migración de software [21].

El mantenimiento de software [23] es la modificación del sistema después de la entrega al usuario. Esta modificación generalmente se hace para arreglar fallas, mejorar el rendimiento o adaptar el sistema al ambiente cambiante [18]. Es por esto, que la comprensión del sistema es crucial para poder mantenerlo correctamente.

La evolución de software se encarga de ampliar los sistemas a través del tiempo. Esta ampliación consiste en tomar una versión operativa del software y generar una nueva versión mejorada en su eficiencia [4]. La comprensión del sistema es necesaria y fundamental antes de comenzar hacia una nueva ampliación.

La migración de software es la conversión de un sistema antiguo a un nuevo contexto tecnológico sin alterar la funcionalidad original del mismo. Esta conversión normalmente es costosa y compleja [24]. Para reducir estos costos, se aconseja comprender correctamente el viejo sistema.

Como se describió en párrafos anteriores, la comprensión de sistemas es fundamental para realizar tareas de mantenimiento, de evolución y de migración de software. Por esta razón, dentro de la IS se encuentra un área que puede ser utilizada para minimizar los costos de las tareas de Mantenimiento, Evolución y Migración de Software, esta área se

la conoce como *Comprensión de Programas* (CP). La CP se define como: *una disciplina de la IS encargada de ayudar al desarrollador a lograr un entendimiento acabado del software, de forma que se pueda analizar disminuyendo el tiempo y los costos.*

El objetivo principal de la CP [7, 6, 5, 22] es desarrollar métodos, técnicas y herramientas que faciliten al programador el entendimiento de las funcionalidades de los sistemas de software. Una forma de alcanzar este objetivo consiste en relacionar el *Domino del Problema*, es decir la salida del sistema, con el *Dominio del Programa*, o sea con las partes del programa utilizadas para generar dicha salida [25, 20, 9, 2]. La construcción de esta relación representa el principal desafío en el contexto de la CP.

Una aproximación a la solución del desafío previamente descripto implica: I) Elaborar una representación del *Dominio del Problema*. II) Construir una representación para el *Dominio del Programa*. III) Unir las representaciones de los pasos I y II con una *Estrategia de Vinculación*.

Para lograr los 3 pasos mencionados en el párrafo precedente, se estudian temáticas importantes asociadas a la CP, algunas de ellas son: los *modelos cognitivos*, la *visualización de software*, la *interconexión de dominios*, la *extracción de la información*, la *administración de la información*. Los *modelos cognitivos* indican como el programador utiliza procesos mentales para comprender el software. La *visualización de software* genera representaciones visuales de los sistemas que ayudan a comprenderlos. La *interconexión de dominios* posee teorías útiles que facilitan la reconstrucción de la relación entre el Dominio del Programa y el Dominio del Problema. La *extracción de la información* captura información relevante que es valiosa para la CP. La *administración de la información* brinda técnicas de almacenamiento y acceso eficiente de la información extraída, cuando el volumen de la misma es grande.

Este trabajo final está orientado a las técnicas de *extracción de información*, por ende, se hará más hincapié en ese tema. Sin embargo, el resto de las temáticas no dejan de ser importantes y constituyen amplias iniciativas de investigación en el área de la CP.

La *extracción de la información consiste en el uso/desarrollo de técnicas que permiten extraer información desde el sistema de estudio.* Dependiendo de las necesidades del Ingeniero del Software, la información extraída puede ser estática o dinámica.

La información dinámica se extrae del sistema en tiempo de ejecución [3, 13]. Un ejemplo conocido de una técnica que extrae información dinámica es la instrumentación de código. Esta técnica consiste en insertar sentencias nuevas en el código fuente del programa, la finalidad de las nuevas sentencias es registrar las actividades realizadas durante la ejecución del programa. Esta inserción de sentencias nuevas no deben alterar la funcionalidad original del programa.

Por otro lado, la extracción de información estática a diferencia de la dinámica, no requiere ejecutar el sistema. Para extraer la información estática se utilizan técnicas de compilación tradicionales. Estas técnicas se encargan de capturar distintos elementos

estáticos que están presentes en el código [13, 1].

El correspondiente trabajo final hará más énfasis en la extracción de información estática. Cabe destacar que la información dinámica también es importante, sin embargo su análisis requiere de otro tipo de aproximaciones que escapan a los objetivos de este trabajo.

Un elemento que forma parte de la información estática y abunda en los códigos de los sistemas son los identificadores. Estudios realizados sobre 2.7 millones de líneas de código escritas en JAVA, indican que más del 70 % de los caracteres del código fuente forman parte de un identificador [12, 8]. Un identificador se define como *una secuencia de letras, dígitos o caracteres especiales de cualquier longitud que sirve para identificar las entidades del programa.* Los identificadores de un programa normalmente están compuestos por más de una palabra en forma de abreviatura, por ejemplo: inp_fl_sys.

Algunos autores [10, 19, 17, 14] señalan que los identificadores ocultan información importante del *Dominio del Problema*. Una forma de analizar identificadores y revelar la información oculta que estos contienen, implica: I) Extraer los identificadores del código de estudio. II) Aplicar una técnica de división, en donde se descompone al identificador en las distintas abreviaturas que lo componen. Por ejemplo: inp_fl_sys → inp fl sys. III) Por último, emplear una técnica de expansión de abreviaturas que las expande a palabras completas. Por ejemplo: inp fl sys → input file system.

Para realizar los pasos de división y expansión se recurren a fuentes de palabras/frases dentro del código, tales como: comentarios, literales strings, documentación. Sin duda, estas son fuentes naturales que ayudan a comprender el código [16]. En caso de que estas fuentes sean escasas, una alternativa viable es consultar fuentes de palabras externas al código como es el caso de los diccionarios con palabras en lenguaje natural, entre otros.

En la actualidad existen técnicas que extraen, dividen y expanden identificadores [17, 14, 11, 15]. Desafortunadamente, no todas estas estrategias de análisis de identificadores se han programado en herramientas automáticas. Por lo tanto, construir herramientas que extraigan y analicen identificadores presentes en programas es un aporte al área de CP.

Tomando como iniciativa el problema sobre la falta de herramientas automáticas en el marco de análisis de identificadores, el presente trabajo final de licenciatura pretende aproximar algunas soluciones. En primer lugar, se construirá una herramienta que extrae identificadores, comentarios y literales de programas escritos con lenguaje JAVA, cabe destacar que los comentarios y literales serán de utilidad a la hora de analizar los identificadores. Luego en esta herramienta se implementarán algunas estrategias de división y expansión de identificadores que fueron descriptas anteriormente. De esta manera, se obtendrá una herramienta que extrae y analiza información estática contenida detrás de los identificadores. Para concluir, elaborar una herramienta de estas características es un aporte directo al área de la CP, ya que permite encarar a uno de sus principales desafíos: vincular el *Dominio del Problema* con el *Dominio del Programa.*

Metodología

La metodología a utilizar en este trabajo final, esta enmarcada por 3 etapas principales:

- **Primera Etapa: Investigación de conceptos orientados a la Comprensión de Programas y a la extracción de información estática.**

  En esta etapa se incorporan los conocimientos necesarios para llevar adelante este trabajo final. Se investigará sobre comprensión de programas y las áreas que contiene. De todas estas áreas, se hará más hincapié en la extracción de información estática en base al análisis de identificadores presentes en los códigos. Este análisis, servirá para la implementación de técnicas de comprensión de sistemas.

- **Segunda Etapa: Elaboración de una estrategia para extraer identificadores.**

  En esta fase se realiza un estudio de técnicas/herramientas encargadas de la extracción de identificadores, comentarios y literales presentes en códigos de programas escritos en JAVA. Tanto los comentarios como los literales se necesitan, por que ayudarán a analizar los identificadores. De todas estas técnicas de extracción, se elegirá la más adecuada para ser implementada. Por último, se llevarán a cabo las pruebas pertinentes.

- **Tercera Etapa: Construcción de la herramienta que analiza identificadores.**

  Finalmente, en la última etapa se construye una herramienta que contendrá la estrategia de extracción elaborada en la etapa anterior y también implementa algunas de las técnicas de análisis de identificadores estudiadas en la primer etapa. La finalidad de esta herramienta es extraer y analizar información presente en identificadores ubicados en códigos JAVA. Esta información se considera propia del *Dominio del Problema*. Por ende, construir una herramienta de esta naturaleza es un aporte a la reconstrucción de la relación entre: el *Dominio del Problema* y el *Dominio del Programa*.

Plan y cronograma de tareas

1. Estudiar conceptos, técnicas y herramientas sobre la comprensión de programas: con esto se lográ conocer las características relacionadas al ámbito de la comprensión de sistemas.

2. Investigar conceptos, técnicas y herramientas relacionados al análisis de identificadores: aquí se adquieren los conceptos necesarios para extraer información del *Dominio del Problema*, en base a los identificadores.

3. Investigar técnicas/herramientas que extraigan identificadores, comentarios y literales ubicados en códigos escritos en lenguaje JAVA.

4. Elaborar un técnica de extracción de identificadores, comentarios y literales elegida en base a la investigación del ítem anterior.

5. Implementar una herramienta en JAVA que contenga alguna de las técnicas de análisis de identificadores investigadas en el ítem 2. Esta herramienta también contendrá la técnica de extracción elaborada en la etapa anterior.

6. Escribir el informe del trabajo final. Mientras se ejecutan los ítems precedentes, se irá redactando el informe correspondiente al trabajo final.

Cronograma

	Actividad	Mes											
		1	2	3	4	5	6	7	8	9	10	11	12
1	Estudiar conceptos, técnicas y herramientas sobre la comprensión de programas.	X	X										
2	Investigar conceptos, técnicas y herramientas relacionados al análisis de identificadores.		X	X									
3	Investigar técnicas/herramientas que extraigan identificadores, comentarios y literales en JAVA.				X								
4	Elaborar un técnica de extracción de identificadores, comentarios y literales en base al ítem anterior.				X	X	X						
5	Construir una herramienta en JAVA que posea la técnica elaborada en el ítem anterior e implemente alguna de las técnicas investigadas en el ítem 2.						X	X	X	X			
6	Escribir el informe del trabajo final.		X	X	X	X	X	X	X	X	X	X	X

Recursos

El correspondiente trabajo final se llevará a cabo en la Universidad Nacional de San Luis, en el Área Programación y Metodologías de Desarrollo de Software enmarcado en el siguiente proyecto:

- ♢ *"Ingeniería de Software: Aspectos de alta sensibilidad en el ejercicio de la profesión de Ingeniero de Software"*. Código: 22/F222. Director: D. Riesco. Co-director: R. Uzal. Universidad Nacional de San Luis. Dicho proyecto, es reconocido por el programa de incentivos y es la continuación de diferentes proyectos de investigación de gran éxito a nivel nacional e internacional.

Por otro lado, el equipamiento necesario es una PC con el sistema Linux, impresora y papel para realizar las impresiones del informe. También se hará uso de internet, de material bibliográfico provisto por la biblioteca "Antonio Esteban Agüero"; y de material provisto por las librerías digitales a las que se tiene acceso desde la Universidad Nacional de San Luis.

Bibliografía

[1] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison Wesley, August 2006.

[2] José Luís Albanes, Mario Berón, Pedro Henriques, and Maria João Pereira. Estrategias para relacionar el dominio del problema con el dominio del programa para la comprensión de programas. *XIII Workshop de Investigadores en Ciencias de la Computación*, pages 449–453, 2011.

[3] Thomas Ball. The concept of dynamic analysis. In Oscar Nierstrasz and Michel Lemoine, editors, *ESEC / SIGSOFT FSE*, volume 1687 of *Lecture Notes in Computer Science*, page 216–234. Springer, 1999.

[4] K. Bennett and V. Rajlich. Software maintenance and evolution: a roadmap. In *ICSE - Future of SE Track*, page 73–87, 2000.

[5] M. Beron, P. Henriques, and R. Uzal. *Inspección de Programas para Interconectar las Vistas Comportamentaly Operacional para la Comprensión de Programas.* PhD thesis, Universidade do Minho, Braga. Portugal, 2010.

[6] Mario Berón, Pedro Henriques, Maria João Pereira, and Roberto Uzal. Program inspection to interconnect behavioral and operational view for program comprehension. University of York, 2007.

[7] Mario Berón, Daniel Eduardo Riesco, Germán Antonio Montejano, Pedro Rangel Henriques, and Maria J Pereira. Estrategias para facilitar la comprensión de programas. In *XII Workshop de Investigadores en Ciencias de la Computación*, 2010.

[8] Dave Binkley, Marcia Davis, Dawn Lawrie, Jonathan Maletic, Christopher Morrell, and Bonita Sharif. The impact of identifier style on effort and comprehension. *Empirical Software Engineering*, 18(2):219–276, 2013. (early access).

[9] R. Brook. A theoretical analysis of the role of documentation in the comprehensionof computer programs. *Proceedings of the 1982 conference on Human factors in computingsystems.*, pages 125–129, 1982.

[10] Bruno Caprile and Paolo Tonella. Nomen est omen: analyzing the language of function identifiers. In *Proc. Sixth Working Conf. on Reverse Engineering*, page 112–122. IEEE, October 1999.

[11] Bruno Caprile and Paolo Tonella. Restructuring program identifier names. In *Proc. Int'l Conf. on Software Maintenance*, page 97–107. IEEE, 2000.

[12] Florian DeiBenbock and Markus Pizka. Concise and consistent naming. In *IWPC*, page 97–106. IEEE Computer Society, 2005.

[13] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Aiding program comprehension by static and dynamic feature analysis. In *ICSM*, pages 602–611, 2001.

[14] E. Enslen, E. Hill, L. Pollock, and K. Vijay-Shanker. Mining source code to automatically split identifiers for software analysis. In *6th IEEE International Working Conference on Mining Software Repositories.*, page 71–80. IEEE, may. 2009.

[15] Henry Feild, David Binkley, and Dawn Lawrie. Identifier splitting: A study of two techniques. In *Proceedings of the Mid-Atlantic Student Workshop on Programming Languages and Systems.*, pages 154–160, 2006.

[16] José Luís Freitas, Daniela da Cruz, and Pedro Rangel Henriques. The role of comments on program comprehension. In *INForum*, 2008.

[17] Emily Hill, Zachary P. Fry, Haley Boyd, Giriprasad Sridhara, Yana Novikova, Lori Pollock, and K. Vijay-Shanker. Amap: Automatically mining abbreviation expansions in programs to enhance software maintenance tools. In *Proceedings of the 5th Int'l Working Conf. on Mining Software Repositories.*, page 79–88. ACM, 2008.

[18] IEEE. *Standard Glossary of Software Engineering Terminology 610.12-1990.*, volume 1. IEEE Press, 1999.

[19] D. Lawrie, H. Feild, and D. Binkley. Extracting meaning from abbreviated identifiers. In *Source Code Analysis and Manipulation, 2007. SCAM 2007. Seventh IEEE International Working Conference on*, page 213–222, Sept 2007.

[20] Michael P O'brien. Software comprehension–a review & research direction. *Department of Computer Science & Information Systems University of Limerick, Ireland, Technical Report.*, 2003.

[21] Roger S. Pressman. *Software Engineering - A Practitioner's Approach.* McGraw-Hill, 5 edition, 2001.

[22] M. Storey. Theories, methods and tools in program comprehension: past, present

and future. In *Program Comprehension, 2005. IWPC 2005. Proceedings. 13th International Workshop on.*, page 181–191, May 2005.

[23] P.F. Tiako. Maintenance in joint software development. In *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International.*, page 1077–1080, 2002.

[24] Marco Torchiano, Massimiliano Di Penta, Filippo Ricca, Andrea De Lucia, and Filippo Lanubile. Software migration projects in italian industry: Preliminary results from a state of the practice survey. In *ASE Workshops.*, page 35–42. IEEE, 2008.

[25] A von Mayrhauser and AM. Vans. Program comprehension during software maintenance and evolution. *Computer*, 28(8):44–55, Aug 1995.

Firma de alumno y asesores

. . . . . .
Mario Berón

. . . . . .
Germán Montejano

. . . . . .
Javier Azcurra