

# Análisis de Identificadores para Abstraer conceptos del Dominio del Problema

Javier Azcurra

Trabajo Final de Licenciatura en Cs. de la Computación

Facultad de Ciencias Físico Matemáticas y Naturales  
Universidad Nacional de San Luis

21 de noviembre de 2013

## Resumen

Las demandas actuales en el desarrollo del software implican una evolución y mantenimiento constante del mismo con el menor costo de tiempo y de recursos. Pensar en estrategias que faciliten las tediosas tareas que diariamente conllevan al crecimiento de los sistemas nos da incapie a iniciarnos en la investigación de herramientas automatizadas que posibiliten el reemplazo del esfuerzo manual que realizan los ingenieros de software a la hora de interpretar un programa.

Estudios indican que los identificadores abundan en los códigos de programas y poseen información relevante detrás de sus abreviaturas por lo tanto no debe ser pasados por alto su análisis a la hora de elaborar herramientas automatizadas de interpretación de códigos. El correspondiente trabajo habla de técnicas basadas en el análisis de identificadores en códigos escritos en lenguaje JAVA<sup>TM</sup>. Las técnicas de análisis constan de dos etapas la primera consiste en la división de las distintas abreviaturas que componen un identificador y la segunda etapa se encarga en la expansión “semántica” de cada abreviatura capturada en la etapa previa, es decir, expandir las abreviaciones en palabras completas. Para lograr esta expansión se utilizan distintas fuentes semánticas comentarios, literales y documentación o bien fuera del sistema basándose en diccionarios. Algunas estrategias de análisis de identificadores se implementaron en la herramienta *Identifier Analyzer* (IDA) donde se emplearon técnicas de compilación para extraer la información estática y analizar los identificadores con el objeto de poder comparar el desempeño de las estrategias y arribar a conclusiones.

# Capítulo 1

## Introducción

### 1.1. Problema

La ingeniería de software contiene tres ítems muy importantes en desarrollo de los sistemas: el *mantenimiento del software*, la *evolución del software* y la *migración del software*. Para que puedan llevar a cabo sus iniciativas los tres necesitan interpretar previamente el sistema que se está desarrollando.

La etapa de mantenimiento es importante en el desarrollo de un software porque los mismos están sujetos a cambios y a una permanente evolución[7]. Es común también que por la constante actualización de los sistemas operativos, los motores de base de datos y demás sistemas externos que interactúan con el software desarrollado entren en conflicto, por eso en la fase de mantenimiento también se debe ir actualizando los distintos componentes del producto para una mejor compatibilidad.[5]. Por lo antedicho entre otras diversas razones indican que el mantenimiento del software consume mucho esfuerzo y dinero. Es necesario pensar en estrategias de automatización que puedan ser aplicadas en fases del mantenimiento del software que ayuden a reducir estos costos, para lograrlo se requiere una comprensión del objeto que se va a modificar antes de realizar algún cambio que sea de utilidad.

Los sistemas complejos evolucionan con el tiempo, los nuevos usuarios y requisitos durante el desarrollo del mismo causan que el producto final posiblemente no sea el que se planteó en un comienzo. Generalmente los ingenieros del software utilizan modelos de procesos conocidos que se diseñan de antemano para adaptarse a un producto que evoluciona con el tiempo, estos modelos se conocen con el nombre de modelos evolutivos o modelos

iterativos, este ultimo hace referencia a que el ingeniero del software pueda obtener una versión entregable del producto cada vez más completa en cada iteración, esto se considera importante ya que las fechas ajustadas y el mercado competitivo obligan a entregar una versión funcional limitada lo más rápido posible y no esperar a tener una única versión completa al final del proceso de desarrollo[5]. La evolución del software básicamente se atribuye al crecimiento de los sistemas, es decir, tomar una versión operativa y generar una nueva versión ampliada, para lograrlo se necesita una previa interpretación del sistema.

La migración del software es una tarea fundamental y compleja dentro del mantenimiento del software. El objetivo es convertir un viejo sistema dentro de una nueva tecnología sin cambiar la funcionalidad del mismo, lograr esto es costoso [4]. Es fundamental lograr una buena comprensión del sistema antiguo, una forma de alcanzarlo es elevar el código antiguo a un nivel más alto de abstracción tomando las principales características [8].

Todos los problemas a los que se enfrentan los desarrolladores de software el primordial es el de mantener los sistemas en buen funcionamiento [9]. Esta tarea es imposible de llevar a cabo de forma manual debido a que consume muchos costos y esfuerzo humano. Por esta razón, existe una subárea de la Ingeniería de Software que se dedica al desarrollo de técnicas de inspección y comprensión de software. Esta área tiene como principal objetivo que el desarrollador logre un entendimiento acabado del software de estudio de forma tal de poder modificarlo disminuyendo en lo posible la gran mayoría de costos [1]. El área mencionada se conoce en la jerga de la Ingeniería de Software como: *Comprensión de Programas (CP)*.

La CP es un área de la Ingeniería de Software cuyo objetivo principal 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 la salida del sistema. **La construcción de esta relación representa el principal desafío en el contexto de la CP.**

## 1.2. Solución

Una solución posible al desafío previamente mencionado consiste en **construir una representación para el dominio del problema y otra para el dominio del programa luego unir ambas representaciones utilizando una estrategia de vinculación**. Los caminos que conducen a facilitar la vinculación de las dos representaciones consiste en el uso/creación de Herramientas de Comprensión. Una Herramienta de Comprensión presenta diferentes perspectivas del software que posibilitan que el ingeniero pueda percibir el funcionamiento del sistema. Para construir herramientas de comprensión, se deben tener en cuenta tres pilares importantes, ellos son: *Interconexión de Dominios*, *Visualización de Software* y *Extracción de la Información* [6, 2].

La *visualización del software* es una característica importante en la comprensión de programas, básicamente provee una o varias representaciones visuales de algún sistema particular [1]. Dichas vistas, cuando están bien elaboradas, permiten analizar y percibir la información extraída desde un programa con mayor facilidad.

Por *Extracción de la Información* se entiende el uso/desarrollo de técnicas que permitan extraer información desde el sistema de estudio. Esta información puede ser: Estática o Dinámica, dependiendo de las necesidades del ingeniero de software o del equipo de trabajo.

Para la extracción de la información estática se utilizan técnicas de compilación tradicionales, que se encargan de recuperar información de cada componente del sistema. Todas las actividades que forman parte de esta tarea se realizan desde el código fuente sin ejecutar el sistema. Generalmente, en este tipo de trabajos se construye un analizador sintáctico con las acciones semánticas necesarias para extraer la información requerida. Por otro lado la extracción dinámica de información del sistema se obtiene aplicando técnicas de instrumentación de código, estas técnicas consisten en insertar sentencias dentro del código fuente del sistema con el fin de recuperar las partes del programa que se utilizaron para producir la salida. La principal diferencia que radica entre ambas técnicas es que las dinámicas requieren que el sistema se ejecute, mientras que las estáticas esto no es necesario.

Los párrafos precedentes permiten percibir la importancia de las técnicas de **extracción de la información**. Sin ellas no sería posible la construcción de visualizaciones y técnicas de interconexión de dominios[1].

**Concluimos que tanto la representación del dominio del proble-**

ma como la representación del dominio del programa se construye en base a la información, estática y dinámica, que se extrae de los mismos. La estrategia de vinculación usa esa información para construir un mapeo entre los elementos de ambos dominios. Por esta razón, en este trabajo se describe una línea de investigación que tiene como principal foco de estudio el análisis, la creación y elaboración de técnicas de extracción de la información estática de los sistemas de software que permitan aproximar a la construcción de la relación entre el Dominio del Problema y el Dominio del Programa. Finalmente, es importante mencionar que la información dinámica es tan importante como la información estática, sin embargo su extracción requiere del estudio de otro tipo de aproximaciones que conforman en sí otra línea de investigación.

Actualmente, existen muchas herramientas de comprensión que basan sus análisis en componentes que están presentes en la información estática de los códigos de programas, alguno de ellos son nombres de variables, tipos de las variables, los métodos de un programa orientado a objeto, las variables locales a un método, etc. Estos son elementos que componen la información formal. Sin embargo, a través del estudio del estado del arte, se pudo detectar que son pocas las estrategias de análisis estático que analizan la información informal que se encuentra disponible en el código fuente, por información informal se entiende aquella contenida en los: *palabras contenidas en comentarios, comentarios de los módulos, comentarios de las funciones, literales strings, documentación* etc. Esto se debe a que dicha información se encuentra expresada en lenguaje natural y por lo tanto su interpretación escapa del análisis estático y requiere de la aplicación de *Técnicas de Procesamiento de Lenguaje Natural* [?, ?]. Este trabajo se centra en los *identificadores* ids como fuente principal de información. Detrás de los ids se encuentra oculta información que es relevante para el objetivo de estudio. Generalmente los ids están compuestos por más de una palabra en forma de abreviatura, una posibilidad para exponer la información oculta consiste en traducir estas abreviaturas en sus correspondientes palabras expresadas en lenguaje natural. La dificultad que presenta esta técnica es que normalmente los nombres de los ids se basan en función de la idiosincrasia del programador [?, ?] y esto representa un problema para el traductor. Otro problema a tener en cuenta es que las abreviaturas representan palabras en lenguaje natural, el cual es ambiguo y puede generar controversia en la conversión.

Por lo tanto es necesario la ayuda de las otras fuentes de información informal presentes en el código que ayuden a la interpretación de los ids como

ya mencionados previamente los comentarios y los literales string son de gran utilidad. Sin duda, los comentarios tienen como principal finalidad ayudar a entender un segmento de código [3]. Dicho de otra manera, son una fuente importante de información de los conceptos del Dominio del Problema. Por esta razón, se puede ver a los comentarios como una herramienta natural para entender el significado de los identificadores de un código, como así también el funcionamiento del sistema en sí.

Otra manera de entender la semántica de un identificador es analizando los literales o constantes strings. Estos representan un valor constante formado por secuencias de caracteres. Ellos son generalmente utilizados en la muestra de carteles por pantalla, y comúnmente se almacenan en variables de tipo *string* (como es el caso de los programas escritos en Java). Tanto los literales como los comentarios están escritos en lenguaje natural, por lo tanto es un desafío su correcta interpretación. Detrás de la información informal se oculta información relevante del Dominio del Problema. Esta información es muy importante porque facilita la reconstrucción de la relación del Dominio del Problema con el Dominio del Programa [?, ?].

- Lograr una mejor comprensión de programas escritos en java a través del análisis de los identificadores encontrados en el código fuente de programas escritos con java.
- Aplicar distintas técnicas de análisis de identificadores para facilitar la comprensión de programas.

### 1.3. Contribución

La línea de investigación descrita en este trabajo final se encuentra enmarcada 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. 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.

También se forma parte del proyecto bilateral entre la Universidade do Minho (Portugal) y la Universidad Nacional de San Luis (Argentina) denominado *Quixote: Desarrollo de Modelos del Dominio del Problema para Inter-relacionar las Vistas Comportamental y Operacional de Sistemas de Software*. Quixote<sup>1</sup> fue aprobado por el Ministerio de Ciencia, Tecnología e Innovación Productiva de la Nación (MINCyT) y la Fundação para a Ciência e Tecnologia (FCT) de Portugal. Ambos entes soportan económicamente la realización de diferentes misiones de investigación desde Argentina a Portugal y viceversa.

### 1.4. Organización de Trabajo Final

En este trabajo final se presenta una línea de investigación que se centra en el estudio, creación e implementación de técnicas de extracción de la información estática desde los sistemas de software. Esta información puede ser estrictamente relacionada con el código del programa, o bien con la información informal provista por los programadores a través de comentarios, literales y documentación. El trabajo esta organizado de la siguiente manera. El capítulo 2 define conceptos teóricos relacionados a la comprensión de programas, cuales son las necesidades que implica su estudio y que soluciones presenta en la vida del desarrollo del software. El capítulo 3 habla de técnicas de análisis de identificadores y que fuentes semánticas son utilizadas para su interpretación. El capítulo 4 trata sobre la herramienta *Identifier Analyzer* (IDA) que implementa algunas técnicas de análisis de identificadores sobre códigos escritos en JAVA<sup>TM</sup> además de algunos casos de estudio. El capítulo 5 menciona las conclusiones elaboradas y posibles trabajos futuros.

---

<sup>1</sup><http://www3.di.uminho.pt/gepl/>



# Bibliografía

- [1] Mario M. Berón, Daniel Riesco, and Germán Montejano. Estrategias para facilitar la comprensión de programas. San Luis, Argentina, April 2010.
- [2] 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.
- [3] José Luís Freitas, Daniela da Cruz, and Pedro Rangel Henriques. The role of comments on program comprehension. 2008.
- [4] Wilhelm Hasselbring, Andreas Fuhr, and Volker Riediger. First international workshop on model-driven software migration. Marzo 2011.
- [5] Roger S. Pressman. *Ingeniería del Software un enfoque práctico*. 2002.
- [6] M. A. Storey, F. D. Fracchia, and H. A. Müller. Cognitive design elements to support the construction of a mental model during software exploration. *The Journal of Systems & Software*, 44(3):171–185, 1999.
- [7] Pierre F. Tiako. Maintenance in joint software development. *Annual International Computer Software and Applications Conference*, 2002.
- [8] Marco Torchiano and Massimiliano Di Penta. Software migration projects in italian industry: Preliminary results from a state of the practice survey. 2007.
- [9] A. Von Mayrhauser and A. M. Vans. Program comprehension during software maintenance and evolution. *Computer*, 28(8):44–55, 1995.