Capítulo 1

Introducción: Problema y Solución

Cuando se desarrollan los sistemas de software se busca que el producto final satisfaga las necesidades de los usuarios, que el buen funcionamiento perdure en el mayor tiempo posible y en caso de hacer una modificación para mejorarlo no implique grandes costos. Estos puntos conllevan a un software de alta calidad y la disciplina encargada de conseguirlo es la *Ingeniería de Software* (IS). Para que la IS logre las metas antes mencionadas se utilizan, entre otras tantas, tres temáticas importantes que están orientadas al desarrollo de buenos sistemas: el *mantenimiento del software*, la *evolución del software* y la *migración del software*.

1.1. Mantenimiento del Software

La etapa de mantenimiento es importante en el desarrollo del software, dado que los sistemas están sujetos a cambios y a una permanente evolución [**?**].

El mantenimiento del software según la IEEE standard [**?**], *es la modificación del software que se realiza posterior a la entrega del producto al usuario con el fin de arreglar fallas, mejorar el rendimiento, o adaptar el sistema al ambiente que ha cambiado.*

De la definición anterior se desprenden 4 tipos de cambios que se pueden

realizar en la etapa de mantenimiento. El *mantenimiento correctivo* cambia el software para reparar las fallas detectadas por el usuario. El *mantenimiento adaptativo* modifica el sistema para adaptarlo a cambios externos, como por ejemplo actualización del sistema operativo o el motor de base de datos. El *mantenimiento perfectivo* se encarga de agregar nuevas funcionalidades al software que son descubiertas por el usuario. Por último, el *mantenimiento preventivo* realiza cambios al programa para facilitar futuras correcciones o adaptaciones que puedan surgir en el futuro [**?**].

Por lo antedicho, entre otras diversas razones [**?**] indican que el mantenimiento del software consume mucho esfuerzo y dinero. En algunos casos, los costos de mantenimiento pueden duplicar a los que se emplearon en el desarrollo del producto. Las causas que pueden aumentar estos costos son, el mal diseño de la arquitectura del programa, la mala codificación, la ausencia de documentación.

No es descabellado pensar estrategias de automatización que puedan ser aplicadas en fases del mantenimiento del software que ayuden a reducir estos costos, lo cual requiere una comprensión del objeto que se va a modificar antes de realizar algún cambio que sea de utilidad.

Algunos autores [**?**, **?**, **?**, **?**] concluyen que el mantenimiento del software y la comprensión de los programas son conceptos que están fuertemente relacionados. Una frase conocida en la jerga de la IS es: *"Mientras más fácil sea comprender un sistema, más fácil será de mantenerlo"*.

1.2. Evolución del Software

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.

La evolución del software básicamente se atribuye al crecimiento de los sistemas a través del tiempo, es decir, tomar una versión operativa y generar una nueva versión ampliada o mejorada en su eficiencia [**?**].

Generalmente, los ingenieros del software recurren a los modelos evolutivos. Estos modelos indican que cuando se desarrolla un producto de software

es conveniente que se divida en distintas iteraciones. A medida que avanza el desarrollo, cada iteración retorna una version entregable cada vez más compleja [**?**].

Estos modelos son muy recomendados sobre todo si se tienen fechas ajustadas donde se necesita una versión funcional lo más rápido posible. De esta manera, no se requiere esperar una versión completa al final del proceso de desarrollo.

Es inevitable y fundamental comprender correctamente la iteración actual del producto antes de comenzar con una nueva. Nuevos requerimientos del usuario pueden aparecer, como así también nuevas dificultades en el desarrollo del sistema. Es por esto que la comprensión del sistema durante su evolución es crucial.

1.3. Migración del Software

Las tareas de mantenimiento de software no solo se realizan para mejorar cuestiones internas del sistema, como es el caso de arreglos de errores de programación o requisitos nuevos del usuario. También se necesitan para adaptar el software al contexto cambiante. Es aquí donde la *migración del software* entra en escena.

De acuerdo con la IEEE standard [**?**], *la migración del software es convertir o adaptar un viejo sistema (sistema heredado) a un nuevo contexto tecnológico sin cambiar la funcionalidad del mismo.*

Las migraciones de sistemas más comunes que se llevan a cabo son causadas por cambios en el hardware obsoleto, nuevos sistemas operativos, cambios en la arquitectura y nuevas base de datos. Sin duda, la que más se ha acentuado en los últimos años son la aparición de nuevas tecnologías web y las mobile (basadas en dispositivos móviles) [**?**].

La migración de grandes sistemas es fundamental, sin embargo está demostrado que es costosa y compleja [**?**]. Para llevarla a cabo reduciendo estos costos se recomienda hacer previamente una buena comprensión del sistema antiguo.

Una técnica de comprensión de sistemas ideada recientemente para la

migración de software, consiste en un modelado de datos [**?**]. Este modelado tiene como finalidad representar el sistema en distintos niveles de abstracción. De esta forma, el grupo encargado de la migración solo debe preocuparse por convertir el antiguo código a la nueva tecnología.

Nuevamente la comprensión de programas se hace presente en temáticas relacionadas al desarrollo de proyectos de software.

1.4. Comprensión de Programas

Como se explicó en las secciones anteriores, es crucial lograr comprender el sistema para llevar adelante las tareas de mantenimiento, evolución y migración del software. Por esta razón, existe un área de la IS que se dedica al desarrollo de técnicas de inspección y comprensión de software. Esta área se conoce con el nombre de *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 tal que se pueda analizar disminuyendo en lo posible el tiempo y los costos* [**?**].

La CP asiste al equipo de desarrollo de software proveyendo métodos, técnicas y herramientas que faciliten el entendimiento del sistema de estudio.

Las investigaciones realizadas en el área de la CP determinaron que el foco de estudio se centra en relacionar el *Dominio del Problema* con el *Dominio del Programa* [**?**, **?**, **?**, **?**]. El primero hace referencia a los elementos que forman parte de la salida del sistema, mientras que el segundo indican las componentes del programa empleados para generar dicha salida. Esta relación es compleja de realizar y representa el principal desafío de la CP.

Una aproximación para relacionar ambos dominios consiste en elaborar la siguiente reconstrucción:

I) Armar una representación del *Dominio del Problema*. II) Construir una representación para el *Dominio del Programa*. III) Unir ambas representaciones con una *Estrategia de Vinculación*.

Para llevar a cabo los 3 pasos mencionados, se necesitan estudiar temáticas asociadas a la CP. Las más importantes son:

- Los *Modelos Cognitivos* son relevantes para la CP porque destacan como el programador utiliza procesos mentales para comprender el software.
- La *Visualización del Software* analiza las distintas partes del sistema y genera representaciones visuales que agilizan la CP.
- La *Interconexión de Dominios* trata de como interrelacionar los elementos de un dominio con otro. Es útil para la CP en la reconstrucción de la relación entre dominio del problema y el dominio del programa.
- La *Extracción de Información* en los sistemas de software, es necesaria para las estrategias de cognición, visualización, interconexión de dominios, etc.
- Al extraer gran cantidad de información se necesita la *Administración de Información*. La misma brinda técnicas de almacenamiento y acceso eficiente a la información extraída.

Todos estos temas se explican con mayor precisión en el próximo capítulo. En la siguiente sección se amplia la *extracción de la información* ya que es la antesala al *análisis de identificadores*, eje central de este trabajo final.

1.5. Extracción de Información

En la CP, la *Extracción de la Información* se define como *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.

La extracción de información estática recupera los distintos elementos en el código fuente de un programa [**?**].

Por otro lado, la extracción dinámica de información implica obtener los elementos que se utilizaron en la ejecución del sistema [**?**]. Generalmente no todas las partes del código son ejecutadas, por lo tanto no todos los elementos se pueden recuperar en una sola ejecución.

La principal diferencia que radica entre ambas técnicas, es que las estáticas se basan en la información presente en el código, mientras que las dinámicas es obligatorio ejecutar el sistema. A veces, se recomienda complementar el uso de ambas estrategias para obtener mejores resultados [?].

En este trabajo final se hace énfasis en la extracción de información estática. Es importante aclarar que la información dinámica es tan importante como la información estática, sin embargo su extracción requiere del estudio de otro tipo de aproximaciones y escapan al alcance de este trabajo.

Como se mencionó previamente, en los códigos de software abundan componentes que conforman la información estática. Alguno de ellos son, nombres de variables, tipos de las variables, los métodos de un programa, las variables locales a un método, constantes. Todos estos componentes se representan mediante los identificadores (ids). Los ids abarcan gran parte de los elementos de un código por lo tanto su análisis no debe pasarse por alto.

1.6. Análisis de Identificadores

Estudios realizados [?, ?, ?, ?] indican que gran parte de los códigos están conformados por identificadores (ids), por ende abundante información estática está representada por ellos (ver capítulo 3).

Generalmente, los ids están compuestos por más de una palabra en forma de abreviaturas. Varios autores coinciden [?, ?, ?, ?] que detrás de estas abreviaturas, se encuentra oculta información que es propia del dominio del problema.

Una vía posible para exponer la información oculta consiste en traducir las abreviaturas antes mencionadas, en sus correspondientes palabras expresadas en lenguaje natural. Para lograrlo: I) primero se extraen los identificadores del código, II) luego se les aplica técnicas de división en donde se descompone al identificador en las distintas palabras abreviadas que lo componen. Por ejemplo: inp_fl_sys $\rightarrow$ inp fl sys. Finalmente, III) se emplea estrategias de expansión a las abreviaturas para transformar las mismas en palabras completas. Siguiendo con el ejemplo anterior: inp fl sys $\rightarrow$ input file system.

Normalmente, los nombres de los ids son elegidos en base a criterios del

programador [**?**, **?**]. Lamentablemente, estos criterios son desconocidos para las técnicas que expanden abreviaturas en los ids. Una forma de afrontar esta dificultad, es recurrir a fuentes de información informal que se encuentran disponible en el código fuente. Por información informal se entiende aquella contenida en los comentarios de los módulos, comentarios de las funciones, literales strings, documentación del sistema y todos lo demás recursos descriptivos del programa que estén escritos en lenguaje natural.

Sin duda, los comentarios tienen como principal finalidad ayudar a comprender un segmento de código [**?**]. Por esta razón, se puede ver a los comentarios como una herramienta natural para entender el significado de los ids en el código, como así también el funcionamiento del sistema.

Por otro lado para poder entender la semántica de los ids, se toman 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.

Los literales string como los comentarios pueden brindar indicios del significado de las abreviaturas que se desean expandir.

En caso de que estas fuentes de información informal sean escasas dentro del mismo sistema, se puede acudir a alternativas externas como es el caso de los diccionarios predefinidos de palabras en lenguaje natural.

Las estrategias de análisis de ids, explicadas en esta sección, se han ido mejorando a lo largo del tiempo. Al principio contaban con diccionarios extensos y ocupaban mucho espacio, luego estos diccionarios se fueron reemplazando por listados de palabras que son acordes a la ciencias de la computación. Estos listados son más eficientes que los diccionarios, ya que contienen palabras más precisas y ocupan menos espacio de almacenamiento (ver capítulo 3).

1.7. Problema y la Solución

Desafortunadamente, hasta el momento no todas las estrategias de análisis de ids han sido implementadas en herramientas automáticas. A su vez, no se encontraron herramientas que integren los pasos (que fueron descriptos en la sección anterior), relacionados a la traducción de ids: I) capturar los

ids del código, II) dividir las palabras abreviadas que componen un id, III) expandir las abreviaturas en base a lo observado en comentarios/literales y/o diccionarios, etc. Tampoco se han encontrado herramientas que implementan más de una estrategia de división y/o más de una técnica de expansión de abreviaturas. De esta forma, se le brindaría al usuario en una única herramienta, la posibilidad de emplear distintas estrategias, ante la variedad de circunstancias que puedan darse en el ámbito del análisis de ids. Por lo descripto previamente, se espera mejorar los resultados obtenidos por las aproximaciones actuales.

Para abordar las iniciativas planteadas y hacer algunos aportes a la solución de estos problemas planteados, se pretende:

- Construir un analizador sintáctico en lenguaje Java que permite extraer los ids, comentarios y literales encontrados en el código fuente del sistema de estudio. La herramienta a seleccionar para realizar esta tarea es ANTLR[1].
- Armar un diccionario de palabras en lenguaje natural que sirva como alternativa a las fuentes de información informal.
- Investigar técnicas de división de ids y técnicas expansión de abreviaturas. Algunas de estas utilizan como principal recurso la información brindada en los comentarios y los literales extraídos del código. Como segunda posibilidad se recurre al diccionario.
- Implementar empleando el lenguaje JAVA algunas de las técnicas del ítem anterior en una herramienta denominada *Identifier Analyzer* (IDA). Esta herramienta también permite visualizar atributos del id (ambiente, tipo de identificador, número de línea, etc) mostrando la parte del código donde se encuentra ubicados.
- Probar la herramienta IDA con casos de estudios, para demostrar la utilidad de la misma.
- Mediante un gráfico de barras, comparar el desempeño de cada técnica implementada en IDA y sacar las conclusiones pertinentes.

[1] ANother Tool for Language Recognition - http://www.antlr.org

1.8. Contribución

El correspondiente trabajo final, es un aporte al área de CP que basa su estudio sobre como capturar conceptos del dominio del problema, a través del análisis de los ids en programas escritos en JAVA. Para lograrlo, primero se describe el estado del arte sobre las estrategias más importantes de análisis de ids. Luego, se fija atención en aquellas técnicas involucradas en la traducción de ids a palabras completas, que consideren distintas fuentes de información informal, como es el caso de los comentarios, los literales o diccionarios predefinidos de palabras. Luego, algunas técnicas antedichas se implementan en una única herramienta llamada *Identifier Analyzer* (IDA). Finalmente, se demostrará la utilidad de la herramienta IDA en el ámbito de la CP, a través de distintos casos de estudio.

1.9. Organización del Trabajo Final

El trabajo está organizado de la siguiente manera. El capítulo 2 define conceptos teóricos relacionados a la comprensión de programas. El capítulo 3 describe el estado del arte asociado a las técnicas de análisis de identificadores conocidas. El capítulo 4 trata sobre la herramienta *Identifier Analyzer* (IDA), en donde se explican los distintos módulos que la herramienta posee y que técnicas de análisis de identificadores tiene implementadas; además se describen algunos casos de estudio. En el capítulo 5 se explican las conclusiones elaboradas y se proponen trabajos futuros.