Capítulo 4

Identifier Analyzer (IDA)

4.1. Introducción

En el capítulo anterior, se explicó la importancia de analizar ids ubicados en los códigos. Los ids de un programa normalmente están compuestos por más de una palabra en forma de abreviatura, por ejemplo: inp_fl_sys. Detrás de estas abreviaturas, los ids ocultan información es propia del Dominio del Problema [10, 25, 22, 16]. Desafortunadamente, las personas ajenas al código, no comprenden a simple vista la información que los ids poseen en sus abreviaturas e invierten tiempo en entenderlas. Es por esto, que las herramientas automáticas de análisis de ids son bienvenidas en el ámbito de la Comprensión de Programas (CP). Con estas herramientas se logra disminuir los tiempos de comprensión de ids y revelar la información que estos contienen en sus abreviaturas.

Dada la importancia que tienen las herramientas de análisis de ids, se tomó la iniciativa de desarrollar una llamada Identifier Analyzer (IDA). Esta herramienta le permite al usuario ingresar un archivo JAVA, luego IDA analiza los ids que están en el archivo. La forma en que IDA expande los ids, se efectuá en tres pasos: I) Extraer los ids del código de estudio. II) Aplicar una técnica de división, en donde se descompone al id 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

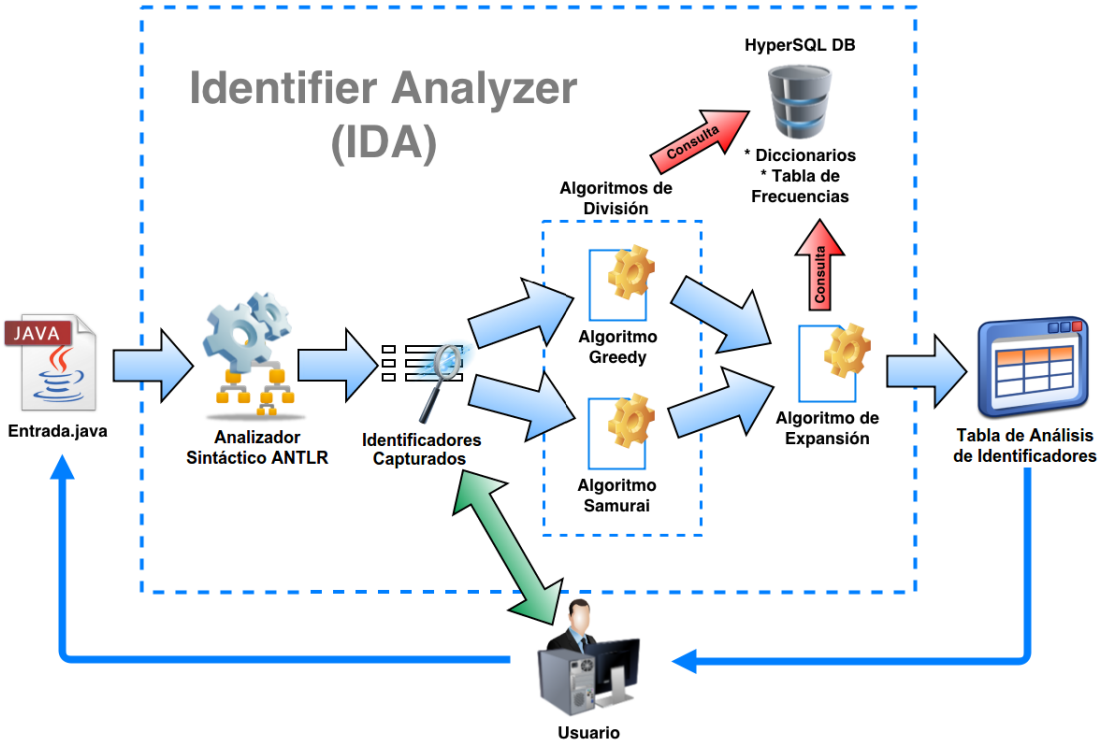


Figura 4.1: Arquitectura de IDA.

expande a palabras completas. Por ejemplo: inp fl sys → input file system. Una vez realizado los tres pasos, los resultados de las expansiones de ids se exhiben en una tabla. El objetivo de IDA es lograr que el usuario comprenda más rápidamente el propósito de los ids en los archivos JAVA, y de esta manera mejorar la comprensión del código analizado.

El correspondiente capítulo, está destinado a explicar los distintos módulos que IDA posee, y que proceso de ejecución debe realizar el usuario para analizar los ids. Al final de este capítulo, se describen algunos casos de estudio que demuestran la importancia de haber construido IDA. Todas estas explicaciones, están acompañadas con capturas de pantalla y tablas que facilitarán al lector entender el funcionamiento de la herramienta IDA. Para comenzar con la descripción de IDA, en la siguiente sección se explica la arquitectura y cuales son sus componentes principales.

4.2. Arquitectura

En la figura 4.1 se puede apreciar la arquitectura de IDA. Esta arquitectura describe tres partes principales, la primera consiste en la *extracción de datos*, la segunda trata sobre la *división de ids* y la tercera sobre *expansión de ids*. A continuación se detallan cada una de ellas.

Módulo de Extracción de Datos: Este módulo recibe como entrada un archivo JAVA que ingresa el usuario (ver Figura 4.1 Entrada.java), luego este archivo se procesa por un Analizador Sintáctico (AS) (ver Figura 4.1 Analizado Sintáctico ANTLR). El AS, extrae y almacena en estructuras internas la información estática perteneciente al código del archivo ingresado. Esta información, está relacionada con ids, literales y comentarios (ver próxima sección para más detalles). El usuario a través de la interfaz de IDA, puede visualizar esta información capturada del código por medio de tablas claramente definidas (ver figura 4.1 - Flecha Verde).

Módulo de División de Ids: Una vez completada la extracción de información, el proceso continua en el módulo de división de ids. Aquí se encuentran implementados dos algoritmos de división; uno es el Algoritmo Greedy y el otro es el Algoritmo Samurai ambos explicados en la sección **??** y **??** del capítulo anterior. Estos algoritmos reciben como entrada la información capturada en el módulo de extracción de datos (ids, comentarios, literales), y luego estos algoritmos dividen los ids del archivo JAVA (ver figura 4.1 Algoritmos de División). Los resultados de las divisiones se almacenan en estructuras internas que serán utilizadas por el módulo de expansión. Cabe recordar que estos algoritmos de división necesitan datos externos para funcionar, uno es el diccionario de palabras (en caso de Greedy) y el otro es lista de frecuencias globales de aparición de palabras (en caso de Samurai). Estos datos externos se encuentran almacenados en una base de datos embebida (ver Figura 4.1 HyperSQL DB).

Módulo de Expansión de Ids: La tercera y última parte, tiene implementado el Algoritmo de Expansión Básico de abreviaturas que fue explicado en la sección **??** del capítulo anterior.

Este algoritmo, toma como entrada los ids separados en el módulo de división de ids (tanto de Greedy como de Samurai), luego el Algoritmo de Expansión expande las abreviaturas resultantes producto de la división de ids (ver Figura 4.1 Algoritmo de Expansión). Los resultados de las expansiones se retornan en dos grupos: las expansiones provistas por el Algoritmo de división Greedy y las provistas por el Algoritmo de división Samurai.

El Algoritmo de Expansión también necesita de un diccionario de palabras, por eso se realizan consultas a la base de datos embebida (ver Figura 4.1 HyperSQL DB). Finalmente, los resultados de las divisiones y las expansiones de los ids, se muestran en una tabla (ver Figura 4.1 Tabla de análisis de identificadores).

4.3. Analizador Sintáctico

Como se explicó en la sección previa, cuando el usuario ingresa un archivo JAVA, IDA examina y extrae información estática presente en el archivo ingresado. Esta información está compuesta por identificadores, comentarios y literales. La manera en que IDA extrae esta información es a través de un Analizador Sintáctico (AS).

La construcción de este AS se llevó a cabo, primero investigando herramientas encargadas de construir AS. Se dio preferencia a aquellas que emplean la teoría asociada a las gramáticas de atributos [1]. De la investigación previamente descripta, se determinó que la herramienta *ANTLR*[1] era la que mejor se ajustaba a las necesidades antes planteadas. Esta herramienta permite agregar acciones semánticas (escritas en JAVA) para el cálculo de los atributos, en una gramática de lenguaje JAVA[2]. Estas acciones semánticas deben estar correctamente insertadas en la gramática para, por ejemplo, implementar estructuras de datos y algoritmos que capturan los ids utilizados en un programa [2]. Una vez insertadas estas acciones, ANTLR lee la gramática y genera el AS adicionando acciones que fueron programadas. De esta manera, se obtiene un AS que recolecta ids mientras examina el código.

[1] ANother Tool for Language Recognition. http://www.antlr.org
[2] http://docs.oracle.com/javase/specs/jls/se7/jls7.pdf

A su vez a estas acciones semánticas, se le agregan otras acciones que extraen comentarios y literales strings. Estos elementos son necesarios ya que sirven para los algoritmos de análisis de ids que serán explicados en próximas secciones.

4.4. Base de Datos Embebida

Como se describió en secciones previas, IDA posee una base de datos embebida. Esta base de datos utiliza una tecnología llamada HSQLDB[1]. Dado que HSQLDB esta desarrollada en JAVA, al momento de incorporarla en IDA (que está programada en JAVA) no resulto una tarea difícil. Otra ventaja por la cual se eligió esta tecnología, es que responde rápidamente las consultas. Esto es importante ya que todos los algoritmos de IDA consultan a HSQLDB. Dentro de esta base de datos embebida, se encuentran almacenadas los diccionarios/listas de palabras que IDA necesita para llevar adelante sus tareas. Estos diccionarios/listas se describen a continuación, nombrando también que algoritmo de IDA consulta cada diccionarios/listas.

Diccionario en Inglés (ispell): Contiene palabras en Inglés que pertenecen a la lista de Palabras Comando de Linux *Ispell*[2]. Se utiliza en el Algoritmo de Greedy y en el Algoritmo de Expansión (ver capítulo 3).

Lista de Palabras Excluyentes (stop-list): Esta compuesta con palabras que son poco importantes o irrelevantes en el análisis de ids[3]. Se utiliza en el Algoritmo de Greedy y en el Algoritmo de Expansión (ver capítulo 3).

Lista de Abreviaturas y Acrónimos Conocidas: Contiene abreviaturas comunes del idioma Inglés y Acrónimos conocidos de programación[4] (gif, jpg, txt). Se emplea en el Algoritmo Greedy (ver capítulo 3).

[1] Hyper SQL Data Base. http://www.hsqldb.org
[2] http://wordlist.aspell.net
[3] http://www.lextek.com/manuals/onix/stopwords1.html
[4] http://langs.eserver.org/acronym-dictionary.txt

Lista de Prefijos y Sufijos Conocidos: Posee Sufijos y Prefijos conocidos en Inglés[1], esta lista fue confeccionada por el autor del Algoritmo Samurai (ver capítulo 3). Se consulta solo en dicho algoritmo.

Frecuencias Globales de Palabras: Lista de palabras, junto con su frecuencia de aparición. Esta lista fue construida por el autor del Algoritmo Samurai[2]. Se emplea solo en dicho algoritmo, más precisamente en la función de *Scoring* (ver capítulo 3).

Cabe destacar que las listas y diccionarios que fueron descriptos poseen palabras que pertenecen al idioma Inglés, dado que los autores así lo determinaron. Por lo tanto, para que la herramienta IDA analice correctamente los ids, se deben ingresar en IDA archivos JAVA con comentarios, literales e ids acordes a la lengua Inglesa.

Habiendo descripto los principales módulos de la herramienta, en la próxima sección se explicará el proceso que debe seguir el usuario para analizar ids a través de IDA.

4.5. Proceso de Análisis de Identificadores

En esta sección, se describe el proceso de ejecución que debe seguir el usuario con la herramienta IDA, para llevar a cabo el análisis de los ids en los archivos JAVA. Se explicarán que función cumple cada elemento de IDA (ventanas, botones, paneles, etc.), y de como estos elementos ayudan al usuario a analizar los ids.

4.5.1. Barra de Menú

Al ejecutar la herramienta IDA, el primer componente de interacción es una simple barra de menú ubicada en el tope de la pantalla, los botones de esta barra son *Archivo*, *Diccionarios* y *Ayuda* (ver figura 4.2).

[1] http://www.eecis.udel.edu/~enslen/Site/Samurai

[2] Esta lista no está disponible, por ende se construyó una aproximación.

Figura 4.2: Barra de Menú de IDA

Al pulsar[1] en *Archivo* de la barra antedicha, se despliega un menú con los siguientes ítems (ver figura 4.2 - Flecha 1):

Abrir archivo(s) JAVA: Abre una ventana que permite elegir uno o varios archivos con extensión JAVA (ver figura 4.3). Los archivos seleccionados serán analizados por IDA (en la próxima sección, serán dados más detalles).

Cerrar Todo: Cierra todos los archivos JAVA abiertos actualmente en la aplicación.

Salir: Cierra la Aplicación.

Cuando se pulsa en *Diccionarios* de la barra de menú, se despliega otro menú con con los siguientes ítems (ver figura 4.2 - Flecha 2):

Ver Diccionarios: Abre una ventana, que muestra un listado de palabras en Inglés correspondiente al diccionario *ispell* (explicado en la sección anterior). La ventana antedicha, también muestra un listado de palabras irrelevantes o stoplist (explicado en la sección anterior). Esta ventana, se explica con más detalles en la sección 4.5.5.

Restablecer B.D.(Base de Datos): Genera nuevamente la base de datos HSQLDB. En caso de haber problemas con la base de datos, es útil restablecerla.

[1]El término 'pulsar' o 'presionar' se utilizará a lo largo del capítulo, significa hacer click con el puntero del ratón.

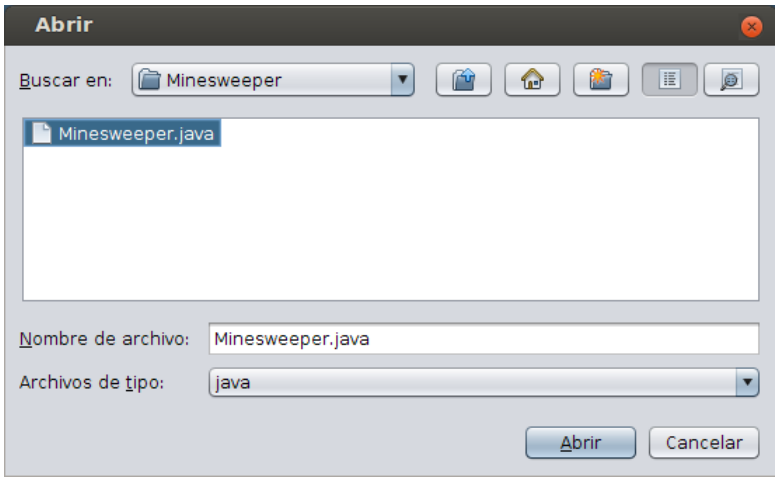


Figura 4.3: Ventana para seleccionar Archivos JAVA.

Finalmente al presionar *Ayuda* de la barra de menú, se despliega un solo botón (ver figura 4.2 - Flecha 3):

Acerca de: Brinda información sobre el autor y directores involucrados en la construcción de la herramienta IDA.

4.5.2. Lectura de Archivos JAVA

Cuando se pulsa en el botón *Abrir archivo(s) JAVA* explicado en la sección anterior (ver figura 4.2 - Flecha 1), se despliega una ventana para que el usuario elija uno o varios archivos JAVA (ver figura 4.3).

Una vez que el usuario elige el/los archivo/s, IDA utiliza un programa externo llamado JACOBE[1]. Este programa JACOBE, recibe como entrada un archivo JAVA y embellece el código fuente contenido en el archivo. Este embellecimiento se realiza para facilitar la lectura del código al usuario. En la próxima sección se describe el panel que IDA tiene para visualizar el código leído del archivo.

La herramienta IDA, además realiza un control de los archivos abiertos, impidiendo que se abra el mismo archivo más de una vez. En caso de que esto suceda, se muestra un cartel informando al usuario (ver figura 4.4). Este

[1]http://www.tiobe.com/jacobe

Figura 4.4: Aviso sobre Archivo JAVA ya abierto en IDA.

control se realiza por cuestiones de coherencia al momento de analizar los archivos.

4.5.3. Panel de Elementos Capturados

Después que el programa JACOBE embellece el código contenido en el archivo JAVA, el mismo se procesa por el AS explicado en secciones previas. Luego que el AS termina sus tareas de extracción de elementos (ids, comentarios y literales), el *Panel de Elementos Capturados* aparece (ver figura 4.5). Este panel en la parte superior posee pestañas, cada pestaña posee un rótulo con el nombre del archivo abierto correspondiente (ver figura 4.5 - Flecha 1). Es posible elegir de a varios archivos para analizar, mediante la ventana de selección de archivos (ver figura 4.3), o también se puede ir eligiendo de a un archivo por apertura de esta ventana. En caso de querer finalizar el análisis de un archivo particular y cerrar la pestaña, se puede pulsar en la cruz ubicada al lado del rótulo de cada pestaña (ver figura 4.5 - Flecha 1).

Cada pestaña en su interior posee el mismo subpanel que se divide en dos partes principales. La parte superior contiene el código leído y embellecido (por JACOBE) del archivo JAVA (ver figura 4.5 - Flecha 2). La parte inferior, muestra toda la información extraída por el AS referente a ids, literales y comentarios. Estos últimos tres poseen una pestaña cada uno (ver figura 4.5 - Flecha 3). Al pulsar sobre cada pestaña, se muestra la información clasificada correspondiente (a ids, literales y comentarios), a continuación se describe como se exhibe la información en cada una de estas pestañas.

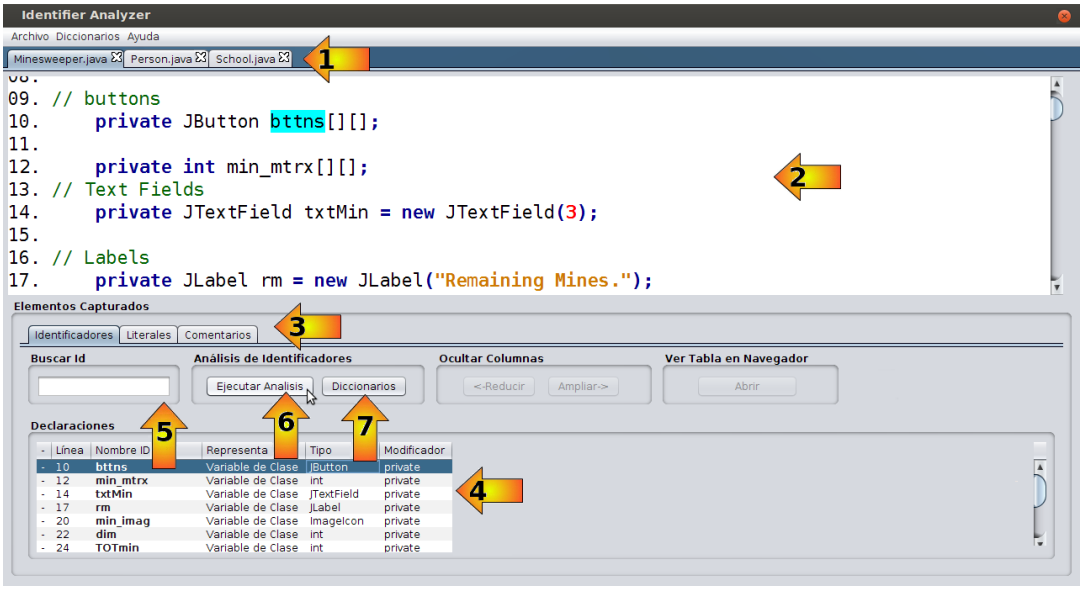


Figura 4.5: Panel de Elementos Capturados

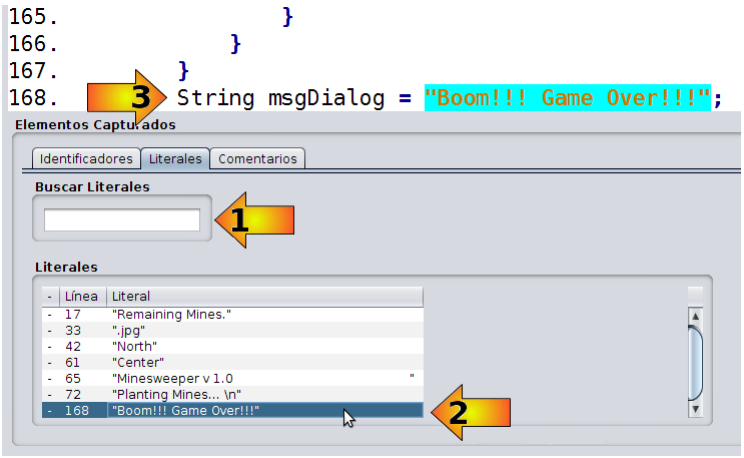


Figura 4.6: Literales Capturados

Identificadores Capturados

Al pulsar la *Pestaña de Identificadores* (ver figura 4.5 - Flecha 3), se muestra la *Tabla de Declaraciones* (ver figura 4.5 - Flecha 4). Esta tabla lista los ids declarados encontrados por el AS y cada columna se corresponde a: el número de línea donde esta declarado el id, el nombre del id, el tipo (int,

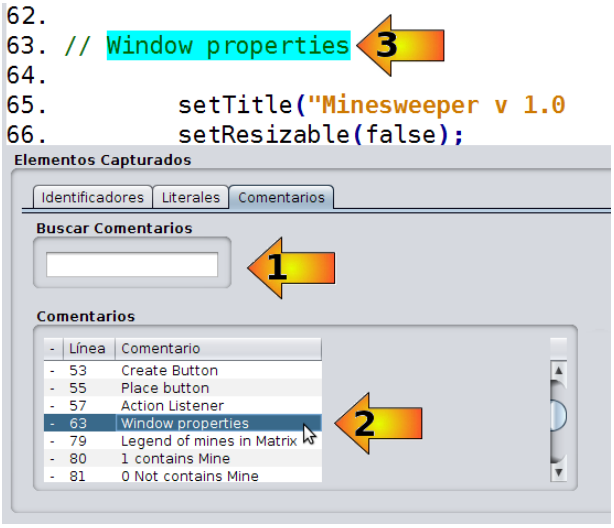


Figura 4.7: Comentarios Capturados

char, etc.), el modificador (publico, privado, protegido), lo que el id representa (variable de clase, constructor, método de clase, etc.). También la *Tabla de Declaraciones* al presionar sobre una fila, inmediatamente se resalta con color en el código del panel superior la declaración del id, según corresponda (ver figura 4.5 - Flecha 2). Cabe mencionar, que la *Tabla de Declaraciones* además posee un buscador, que realiza búsquedas por el nombre del id; para realizar las búsquedas el usuario debe escribir en el cuadro de texto ubicado dentro del recuadro *Buscar Id* (ver figura 4.5 - Flecha 5), a medida que se escriba en este cuadro de texto, se irán filtrando los resultados en la *Tabla de Declaraciones*.

Literales Capturados

Al pulsar la *Pestaña de Literales* (ver figura 4.5 - Flecha 3), se puede apreciar que contiene una tabla y un práctico buscador para agilizar la búsqueda de literales en la tabla (ver figura 4.6 - Flecha 1 y 2). Esta tabla contiene 2 columnas, número de línea del literal y el literal propiamente dicho (ver figura 4.6 - Flecha 2). Al pulsar sobre alguna fila de la tabla antedicha, automáticamente el literal correspondiente se resalta en el código (del archivo JAVA) ubicado en la parte superior del *Panel de Análisis* (ver figura 4.6 -

Flecha 3).

Comentarios Capturados

Al presionar la *Pestaña de Comentarios* (ver figura 4.5 - Flecha 3), se visualiza una tabla listando los comentarios encontrados en el archivo y un buscador de comentarios en la tabla (ver figura 4.7 - Flecha 1). Esta tabla contiene 2 columnas que corresponden, por un lado al comentario y por el otro al número de línea donde se encuentra el comentario dentro del código (ver figura 4.7 - Flecha 2). Al igual que se describió en el párrafo anterior, al presionar en una de las filas de la tabla inmediatamente se resalta en el código mostrando la ubicación del comentario seleccionado (ver figura 4.7 - Flecha 3).

Hasta aquí, solo se ha descripto como IDA exhibe la información capturada del código. A continuación, se explicará como se emplea IDA para analizar los ids. Para ello, se debe pulsar en la *Pestaña de Identificadores* (ver figura 4.5- Flecha 3), luego pulsar en el botón *Ejecutar Análisis* ubicado dentro del cuadro *Análisis de Identificadores* (ver figura 4.5 - Flecha 6), al hacerlo se abrirá el *Panel de Análisis* que será explicado en la próxima sección.

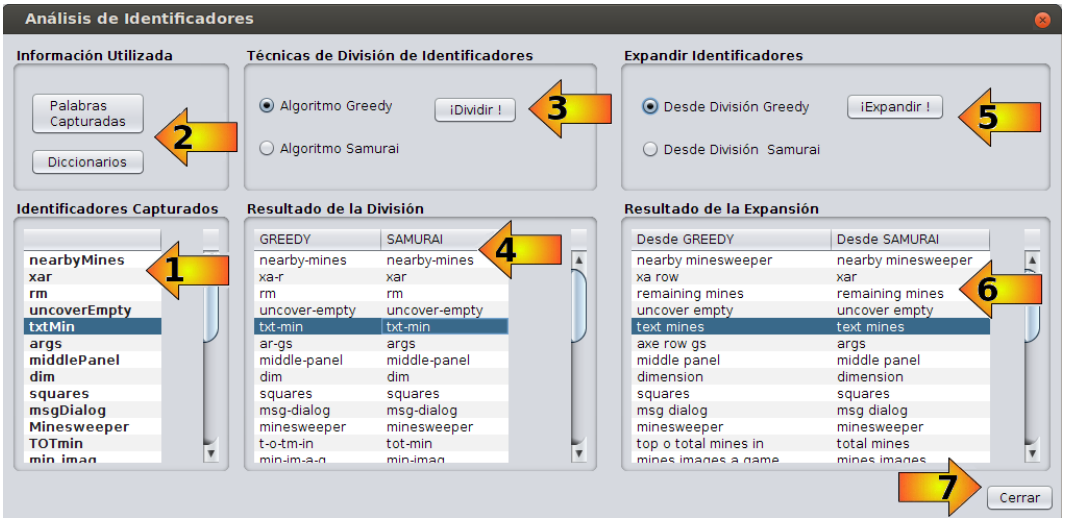


Figura 4.8: Panel de Análisis

4.5.4. Panel de Análisis

El *Panel de Análisis* (ver figura 4.8) contiene 3 partes principales, (de izquierda a derecha) la primera consiste en los ids capturados, los mismos están listados en el cuadro inferior izquierdo (ver figura 4.8 - Flecha 1), arriba de estos se encuentran dos botones *Palabras Capturadas* y *Diccionarios* (ver figura 4.8 - Flecha 2), estos brindan información al usuario sobre los datos que se utilizan para ejecutar los algoritmos de análisis. Los mismos serán explicados en la próxima sección.

Siguiendo con el *Panel de Análisis*, en el cuadro central superior (ver figura 4.8 - Flecha 3) se pueden seleccionar los dos algoritmos de división de ids (Greedy y Samurai), el botón *Dividir* del mismo cuadro ejecuta la técnica seleccionada mostrando en el cuadro inferior la tabla con los resultados. En la figura 4.8 - Flecha 4 se muestran ambas técnicas (Greedy y Samurai) ya ejecutadas y listando los resultados.

De la misma manera, en los 2 paneles subsiguientes de la derecha, permiten expandir las palabras y mostrar los resultados en el cuadro de abajo. Al presionar el botón *Expandir*, situado en el cuadro superior derecho (ver figura 4.8 - Flecha 5) ejecuta el algoritmo de expansión básico, tomando como entrada los ids divididos desde Greedy o desde Samurai, según haya seleccionado el usuario en este mismo cuadro (ver figura 4.8 - Flecha 5). Los resultados obtenidos de las expansiones (desde Greedy y desde Samurai) se muestran en la tabla ubicada en el cuadro inferior derecho (ver figura 4.8 - Flecha 6).

4.5.5. Palabras Capturadas y Diccionarios

En la sección anterior, se describieron dos botones *Palabras Capturadas* y *Diccionarios*, ubicados en el *Panel de Análisis* (ver figura 4.8 - Flecha 2). Al pulsar el primero, abre una ventana que posee dos cuadros (ver figura 4.9), el cuadro de la izquierda contiene una tabla que muestra las frecuencias correspondiente al Algoritmo Samurai (ver figura 4.9 - Flecha 1). Esta tabla tiene 3 columnas, en la primera posee tokens[1], los mismos fueron capturados

[1]Genérico utilizado por el autor, para denotar ids, palabras de comentarios y literales.

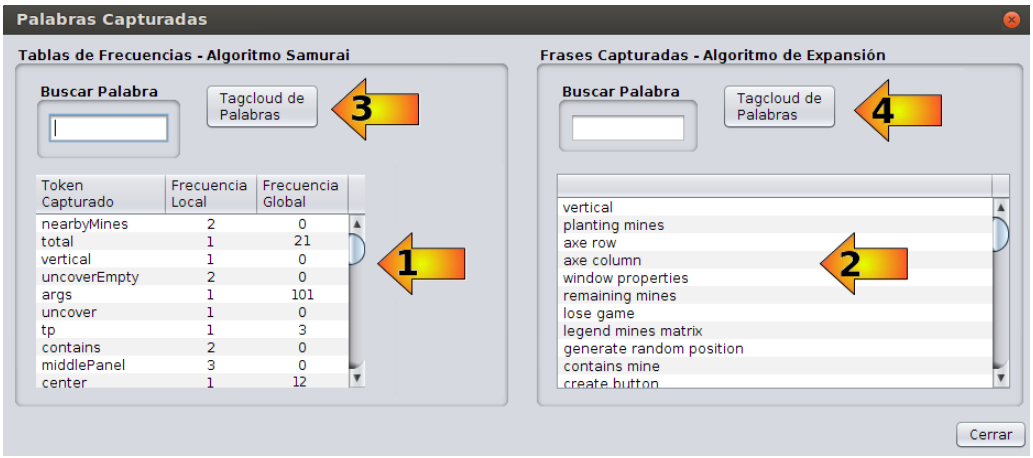


Figura 4.9: Información Utilizada para el Análisis de Ids

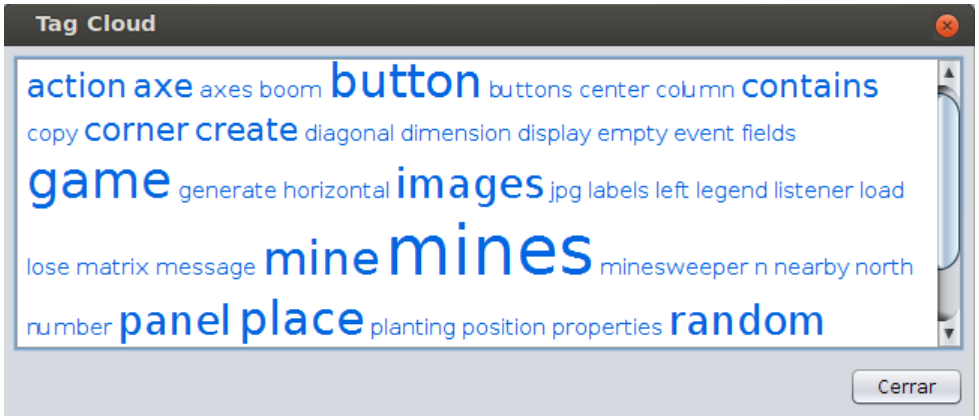


Figura 4.10: TagCloud: Nube de Etiquetas

por el AS ANTLR. La segunda columna, contiene la frecuencia local de cada token, cabe recordar que la frecuencia local se construye en función de la frecuencia absoluta de aparición de los tokens en el código del archivo actual (ver capítulo 3). La tercera y última columna de la tabla denota la frecuencia global de cada token, la misma esta predefinida en la base de datos HSQLDB (ver sección 4.4) y se armó mediante el análisis hecho por los autores del Algoritmo Samurai (ver capítulo 3).

El cuadro de la derecha, lista en una tabla las frases capturadas (ver figura 4.9 - Flecha 2), estas frases se obtienen de los comentarios y los literales strings, el Algoritmo de Expansión es el encargado de utilizarlas (ver capítulo

Figura 4.11: Panel de Diccionarios

3), ya que estas frases son posibles candidatas a expandir las abreviaturas que tiene un id ubicado en el código.

Cada uno de los botones con el nombre *TagCloud de Palabras* (ver figura 4.9 - Flechas 3 y 4) abre una ventana que contiene una *Nube de Etiquetas* (ver figura 4.10). Esta nube de palabras resalta en tamaño más grande las palabras que más frecuencia de aparición tienen (ver figura 4.10). Para generar esta nube se emplea una librería de JAVA llamada OpenCloud[1]. Esta nube de palabras ayuda a ver con mas claridad que palabras son más frecuentes en el código. Para el caso de la nube de frecuencias de Samurai (ver figura 4.9 - Flecha 3), el tamaño de cada palabra depende de la Frecuencia Local de cada token (ver figura 4.9 - Flecha 1). En el caso de la nube de las frases capturadas (ver figura 4.9 - Flecha 4), el tamaño de las palabras esta dado por el número de apariciones dentro de esta tabla de frases (ver figura 4.9 - Flecha 2).

A modo de agilizar la búsqueda de alguna palabra y/o token[1], se puede llevar a cabo utilizando los cuadros de texto con rótulo *Buscar Palabra* si-

[1]http://opencloud.mcavallo.org

[1]Genérico utilizado por el autor, para denotar ids, palabras de comentarios y literales.

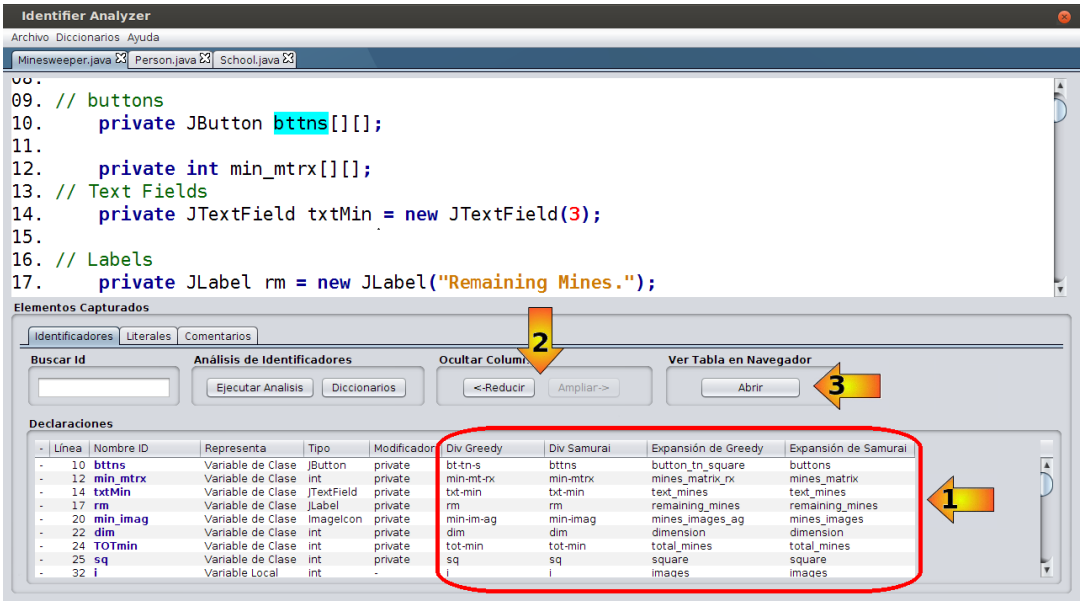
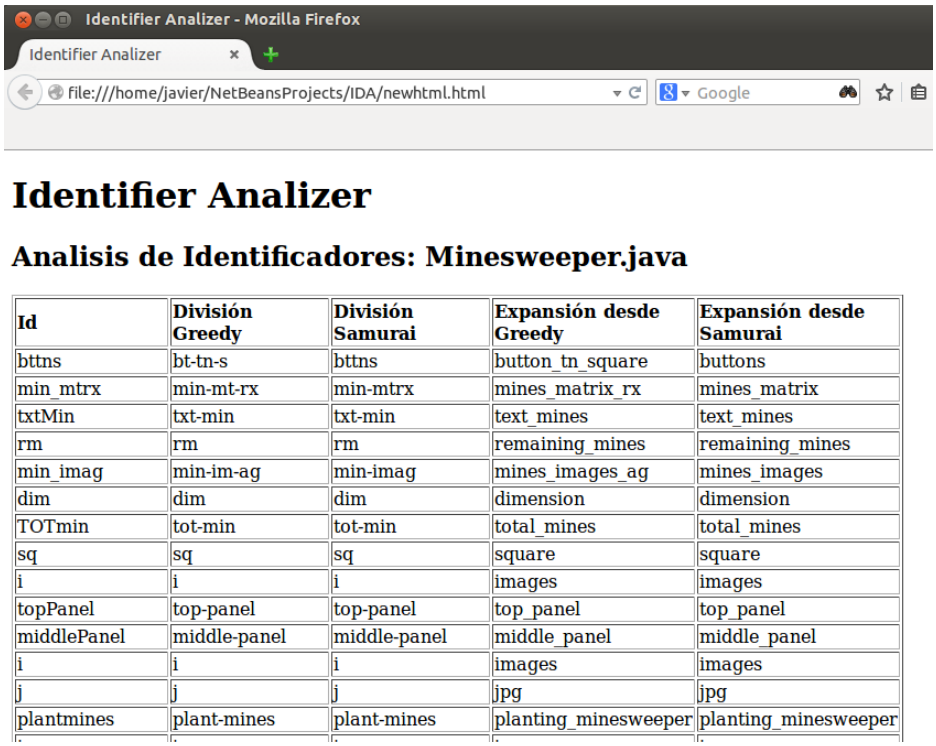


Figura 4.12: Panel de Elementos Capturados

tuados al lado de cada botón *TagCloud de Palabras* (ver figura 4.9 - Flechas 3 y 4).

Volviendo al *Panel de Análisis* si se pulsa el botón *Diccionarios* (ver figura 4.8 - Flecha 2), se abre el *Panel de Diccionarios* (ver figura 4.11). Este panel posee dos tablas, la tabla de la izquierda lista todas las palabras en inglés que tiene el diccionario del comando de Linux *ispell*, esta lista de palabras es utilizada por el Algoritmo Greedy (ver capítulo 3) y el Algoritmo Expansión Básica (ver capítulo 3). La segunda tabla de la derecha enumera las palabras que pertenecen a la stoplist o lista de palabras irrelevantes, también utilizada por los dos algoritmos antedichos. Convenientemente, ambas tablas poseen un buscador por palabras dado que el contenido de cada una es amplio (ver figura 4.11). Este *Panel de Diccionarios* puede ser invocado desde otros lugares de la herramienta IDA. Uno de ellos es desde la barra de menú (ver Figura 4.2 - Flecha 2). Otro sitio donde puede abrirse, es desde el *Panel de Elementos Capturados*, pulsando el botón que esta al lado de *Ejecutar Análisis* (ver figura 4.5 - Flecha 7).



Id	División Greedy	División Samurai	Expansión desde Greedy	Expansión desde Samurai
bttns	bt-tn-s	bttns	button_tn_square	buttons
min_mtrx	min-mt-rx	min-mtrx	mines_matrix_rx	mines_matrix
txtMin	txt-min	txt-min	text_mines	text_mines
rm	rm	rm	remaining_mines	remaining_mines
min_imag	min-im-ag	min-imag	mines_images_ag	mines_images
dim	dim	dim	dimension	dimension
TOTmin	tot-min	tot-min	total_mines	total_mines
sq	sq	sq	square	square
i	i	i	images	images
topPanel	top-panel	top-panel	top_panel	top_panel
middlePanel	middle-panel	middle-panel	middle_panel	middle_panel
i	i	i	images	images
j	j	j	jpg	jpg
plantmines	plant-mines	plant-mines	planting_minesweeper	planting_minesweeper

Figura 4.13: Tabla del Análisis de Ids en Navegador Web.

4.5.6. Nuevamente al Panel de Elementos Capturados

Una vez que los ids fueron analizados (Divididos y Expandidos) mediante el *Panel de Análisis*, el mismo debe ser cerrado presionando el botón *Cerrar* (ver figura 4.8 - Flecha 7). Esta acción, retorna al *Panel de Elementos Capturados* nuevamente (ver figura 4.12). Como se puede observar, en la tabla *Declaraciones* que detalla los ids extraídos e información asociada a estos, se le suman nuevas columnas (ver figura 4.12 - Flecha 1). Estas nuevas columnas contienen los resultados obtenidos de los algoritmos de división (Greedy, Samurai) y el algoritmo de expansión ejecutados en el *Panel de Análisis*.

Al agregar las columnas nuevas se habilita el cuadro *Ocultar Columnas* (ver figura 4.12 - Flecha 2). En el se encuentran dos botones *Reducir* y *Ampliar*. El primero de ellos a modo de facilitar la visualización, oculta las columnas que hay entre los ids y las columnas que contienen el análisis de ids (las columnas que se ocultan son: tipo, modificador, Representa), de esta

manera el usuario puede comparar más claramente las distintas divisiones y expansiones de ids. Mientras que el botón *Ampliar* restablece las columnas originales (ver figura 4.12 - Flecha 2).

Luego si el usuario lo decide, puede presionar el botón *Abrir* en el cuadro *Ver Tabla en Navegador* (ver figura 4.12 - Flecha 3). Esta acción abre el navegador web del sistema operativo, y mediante una pagina web en formato html, muestra una tabla con los resultados obtenidos del análisis de ids (ver figura 4.13).

4.6. Casos de Estudio

En esta sección se presentarán 3 casos de estudios realizados con la herramienta IDA. En cada uno de estos casos, se examinan los ids de un único archivo JAVA. A través de tablas, se irán mostrando los resultados parciales que se van obteniendo durante el proceso de análisis de los ids. Con estos casos, se pretende ostentar la utilidad de la herramienta IDA en lo que respecta al análisis de ids y mostrar que es un aporte al área de la CP.

4.6.1. Buscaminas (Minesweeper)

El programa JAVA denominado Minesweeper.java, al ejecutarlo posee el clásico y conocido juego llamado Buscaminas (Minesweeper en Inglés - ver figura 4.14). Este programa contiene un módulo de 250 líneas aproximadamente que fueron analizadas por IDA.

Al ingresarse el programa Minesweeper.java a la herramienta IDA, se da comienzo a la fase de extracción de datos y el analizador sintáctico captura información referente a los ids. Esta información la exhibe IDA al usuario en el *Panel de Elementos Capturados*, en la tabla *Declaraciones* (ver figura 4.5 - flecha 4). En la tabla 4.1 se puede apreciar esta información: la línea donde esta declarado el id, que representa en el código analizado, el tipo, el modificador .

Para hacer una descripción más detallada sobre los ids capturados, en la tabla 4.1 se puede observar que el archivo Minesweeper.java tiene ids del

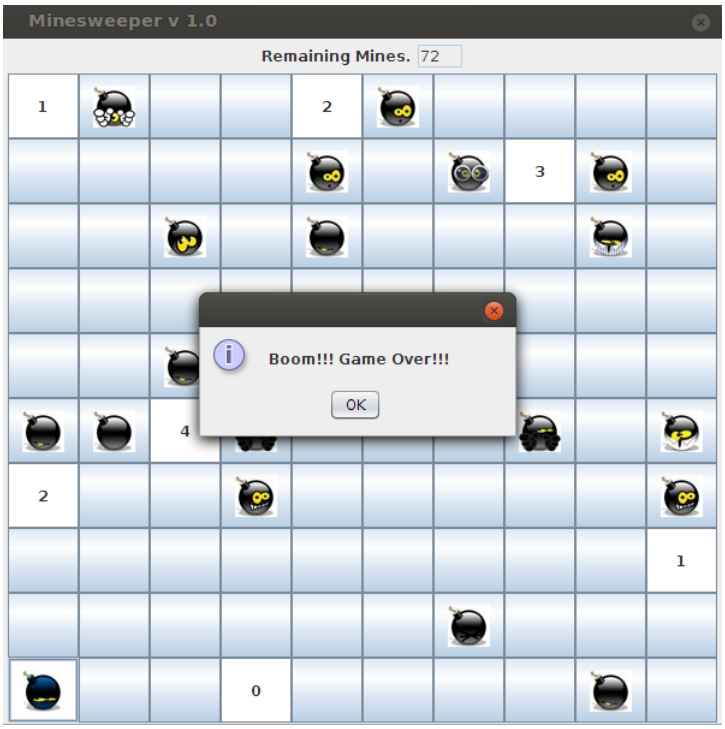


Figura 4.14: Captura del Juego Buscaminas programado en JAVA

tipo *hardwords* y *softwords* (ver capítulo 3). Algunos *hardwords* que se pueden observar son min_mtrx, TOTmin, topPanel (entre otros), ya que estos poseen una marca de separación que destacan las palabras que lo componen. Por otro lado, algunos de los *softwords* que se capturaron son ae, plantmines, bttns (entre otros).

A su vez, se capturaron los comentarios junto a la línea del código donde se ubican cada uno, los mismos se muestran en tabla 4.2. De la misma forma, los literales Strings que se extrajeron se exhiben en la tabla 4.3. En esta tabla, se puede observar que además del número de línea donde se ubica el literal. Tanto los literales y los comentarios se visualizan en IDA a través del *Panel de Elementos Capturados*, eligiendo la pestaña correspondiente (ver figura 4.5 - Flecha 3).

Es importante recordar, que los comentarios y literales de las tablas 4.2 y 4.3 son usados para construir la lista de frases que se muestra en la ventana de *Palabras Capturadas* en la figura 4.9 - flecha 4. Esta información es útil

para el Algoritmo de Expansión.

A continuación, se procede a analizar los ids, el usuario realiza esto con el *Panel de Análisis* (ver figura 4.8) aplicando las técnicas de división (Greedy y Samurai) y después la técnica de expansión de abreviaturas. Los resultados más destacados se pueden apreciar en la tabla 4.4.

Análisis de Resultados

La información mostrada en la tabla 4.4, las columnas de *Greedy* y *Samurai* muestran los resultados de división de dichos Algoritmos. En las columnas *Expansión desde Greedy* y *Expansión desde Samurai* se enumeran los resultados de haber expandido las distintas partes del id, que resultaron desde los Algoritmos Greedy y Samurai respectivamente.

Los ids analizados por la herramienta IDA (ver tabla 4.4) en lo que respecta a *hardwords*, se pueden encontrar con guión bajo min_imag, min_mtrx para el tipo camel-case uncoverEmpty, msgDialog y para el caso especial TOTmin, MINSnum (variante camel-case), entre otros. El algoritmo Greedy manifiesta irregularidades a la hora de dividir ya que siempre considera que la mayor cantidad de divisiones es la mejor opción, esto puede observarse en casos como min–im–ag, bt–tn–s, min–sn–um (ver tabla 4.4 - columna Greedy). Para el caso especial MINSnum, es interesante observar como Samurai se da cuenta de donde hacer la división, y no la considera un caso común de camel-case que la separa antes de la mayúscula seguido de minúscula (ver capítulo 3). No sucediendo lo mismo que Greedy ya supone que es del tipo camel-case haciendo la separación incorrecta min–sn–um.

En lo que respecta a softwords se aprecia la presencia de acrónimos como rm y ae (entre otros). El Algoritmo de Expansión consulta la lista de frases (ver figura 4.9 - flecha 4) conformada por comentarios y los literales capturados (ver tablas 4.2 y 4.1), al encontrar coincidencia con el literal "remaining mines" y el comentario "action event", selecciona ambos como la expansión correspondiente de rm y ae (ver tabla 4.4 - Columnas de Expansión). El resto de los softwords se puede considerar a mins, bttns, dim, aquí Greedy también acusa inconvenientes separando los ids, mientras que Samurai no los divide (ver tabla 4.4).

Para finalizar, los ids i, j son comunes en la mayoría de los códigos, y son difíciles de traducir. Por ende, el Algoritmo de Expansión busca en las tablas de Literales y Comentarios (ver tablas 4.2 y 4.3), palabras que estén dentro del *Dominio del Problema* y de esta manera tratar de darle una traducción válida en este contexto.

Línea	**Nombre ID**	**Representa**	**Tipo**	**Modificador**
7	Minesweeper	Clase	–	public
10	bttns	Variable de Clase	JButton	private
12	min_mtrx	Variable de Clase	int	private
14	txtMin	Variable de Clase	JTextField	private
17	rm	Variable de Clase	JLabel	private
20	min_imag	Variable de Clase	ImageIcon	private
22	dim	Variable de Clase	int	private
24	TOTmin	Variable de Clase	int	private
25	sq	Variable de Clase	int	private
37	topPanel	Variable Local	JPanel	–
48	middlePanel	Variable Local	JPanel	–
71	plantmines	Método de Clase	void	private
71	mins	Parámetro	int	–
102	main	Método de Clase	void	public
102	args	Parámetro	String[]	–
107	actionPerformed	Método de Clase	void	public
107	ae	Parámetro	ActionEvent	–
124	uncoverEmpty	Método de Clase	void	private
124	j	Parámetro	int	–
124	i	Parámetro	int	–
150	win	Método de Clase	void	private
159	boom	Método de Clase	void	private
172	msgDialog	Variable Local	String	–
179	nearbyMines	Método de Clase	int	private
188	MINSnum	Variable Local	int	–
179	xar	Parámetro	int	–
179	yac	Parámetro	int	–

Tabla 4.1: Identificadores extraídos por el AS ANTLR

Línea	Comentario
9	buttons
13	Text Fields
16	Labels
19	Mines images
21	Dimension
23	total mines
27	Time tp
31	load Images
36	Top Panel
47	Button panel
50	Create and place button
53	Create Button
55	Place button
57	Action Listener
63	Window properties
80	Legend of mines in Matrix
81	1 contains Mine
82	0 Not contains Mine
83	Place random mine

Línea	Comentario
85	Generate random position
91	Place mine
93	Display mines panel
107	Action Event
126	Uncover an empty square
129	Nearby Mines
136	restart game
152	Win the game
161	lose the game
165	Mines Random Images
183	x axe row
184	y axe column
186	return the number of mines
192	horizontal
199	vertical
207	diagonal
208	Top left corner
209	copy of axes
224	top right corner

Tabla 4.2: Comentarios extraídos por el AS ANTLR

Línea	Literal
17	“Remaining Mines.”
33	“.jpg”
42	“North”
61	“Center”
65	“Minesweeper v 1.0 ”
73	“Planting Mines...”
153	“You Win!!! Game Over!!!”
155	“Message”
173	“Boom!!! Game Over!!!”
175	“Message”

Tabla 4.3: Literales extraídos por el AS ANTLR

Id	Greedy	Samurai	Exp. desde Greedy	Exp. desde Samurai
Minesweeper	minesweeper	minesweeper	minesweeper	minesweeper
bttns	bt–tn–s	bttns	button tn square	buttons
min_mtrx	min–mt–rx	min–mtrx	mines matrix rx	mines matrix
txtMin	txt–min	txt–min	text mines	text mines
rm	rm	rm	remaining mines	remaining mines
min_imag	min–im–ag	min–imag	mines images ag	mines images
dim	dim	dim	dimension	dimension
TOTmin	tot–min	tot–min	total mines	total mines
sq	sq	sq	square	square
topPanel	top–panel	top–panel	top panel	top panel
middlePanel	middle–panel	middle–panel	middle panel	middle panel
plantmines	plant–mines	plant–mines	planting minesweeper	planting minesweeper
mins	min–s	mins	mines square	mines
main	main	main	main	main
args	args	args	args	args
actionPerformed	action–performed	action–performed	action performed	action performed
ae	ae	ae	action event	action event
uncoverEmpty	uncover–empty	uncover–empty	uncover empty	uncover empty
j	j	j	jpg	jpg
i	i	i	images	images
win	win	win	window	window
boom	boom	boom	boom	boom
msgDialog	msg–dialog	msg–dialog	message dialog	message dialog
nearbyMines	nearby–mines	nearby–mines	nearby minesweeper	nearby minesweeper
MINSnum	min–sn–um	mins–num	mines sn um	mines number
xar	xa–r	xar	xa row	x axe row
yac	y–ac	yac	yellow action	y axe column

Tabla 4.4: Análisis Realizado a los Ids extraídos de Minesweeper.java

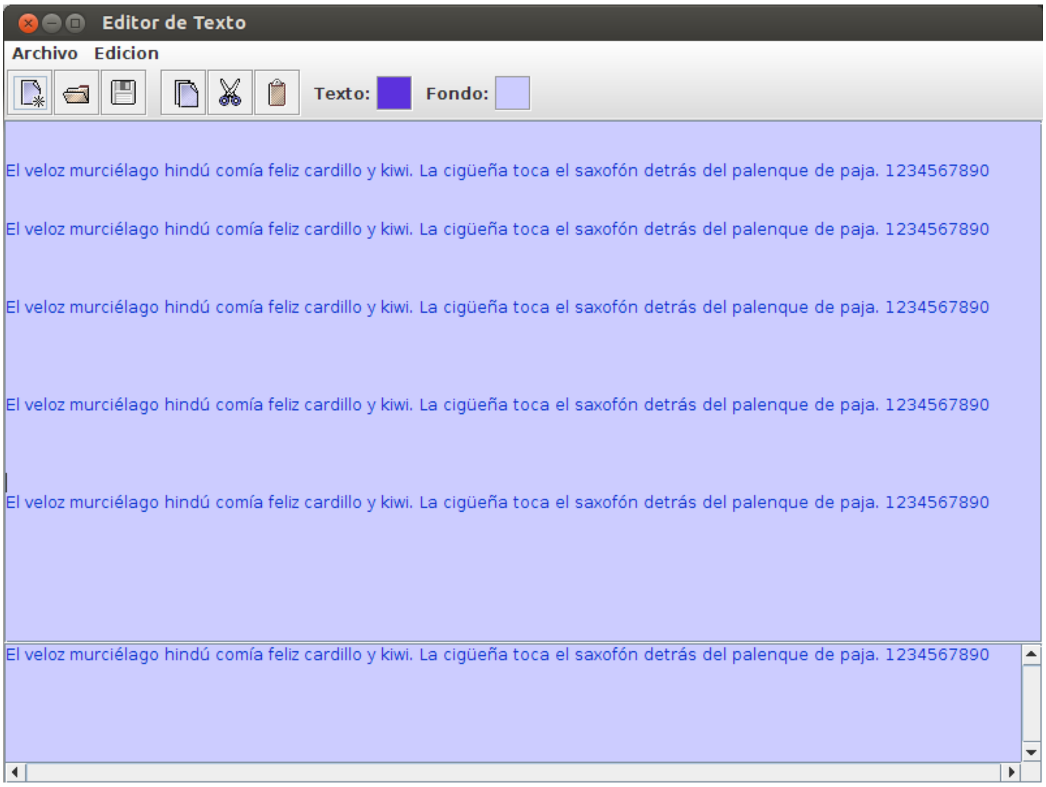


Figura 4.15: Captura del Editor de texto programado en JAVA

4.6.2. Editor de Texto

Cabe destacar que, a modo de facilitar la lectura, de aquí en adelante, por cada caso de estudio, se mostrará únicamente la tabla con resultados obtenidos producto del análisis de los ids. Se excluyen las tablas de comentarios y literales.

En el próximo caso de estudio, se seleccionó un programa escrito en JAVA que corresponde a un editor de texto (ver figura 4.15). Este editor posee las herramientas básicas para crear o modificar archivos de textos sin formato, permite cambiar la visualización de colores en las letras y en el fondo, también imprime los archivos que se abran en el. Este editor esta programado con alrededor de 500 líneas de código, las cuales están escritas en un único archivo llamado Editor.java.

A continuación, el editor de textos se analiza con la herramienta IDA, el

análisis efectuado en los ids se exhibe en la tabla 4.5. Dado que este programa posee muchos ids, los resultados que se presentan en la tabla 4.5 son los más destacados y no son la totalidad.

Análisis de Resultados

A simple vista, se puede observar en la tabla 4.5, al igual que el caso de estudio de la sección anterior, que las divisiones entre Greedy y Samurai, las de este último son las que mejor se realizan. Esto ocurre por que Greedy siempre selecciona la mayor cantidad de divisiones en la palabra como la mejor opción (ver capítulo 3). Ejemplos de divisiones mal hechas por Greedy son bckCol → b–c–k–col, btAcept → bt–ace–pt, jChoColor → j–c–ho–color, entre otros.

Sin embargo, en este caso de estudio a diferencia del anterior existen algunas divisiones en las que Samurai falla, los ids que se pueden observar son: flreader, ptjob los cuales no se dividieron como lo hizo Greedy en fl–reader, pt–job. La primer hipótesis que se maneja sobre este resultado, es que Greedy encuentra en el diccionario en Inglés las palabras reader y job. Mientras que Samurai en su función de *Score*, las palabras fl con reader y pt con job no representan puntajes (score) altos para ser divididas entre ellas (ver capítulo 3).

En lo que respecta a la expansión de ids, la mayoría de las abreviaturas resultantes de la división de ids fueron expandidas; algunos ejemplos son sel → select, tl → tool, cl → close, bck → background, entre otras. Estas palabras son expandidas por el algoritmo de expansión, gracias a los comentarios y literales capturados por el AS. Por otro lado, existen algunas abreviaturas que no se expanden como es el caso de bt a button, it a item; que forman parte de los ids bt–acept y men–it–new. La hipótesis de este comportamiento, se debe por que estas abreviaturas son tratadas como acrónimos (abreviaturas que poseen más de una palabra) dado que tienen solo 2 caracteres y no encuentra una frase (de comentarios o literales) capturada por el AS, que coincida. Como se consideran abreviaturas de más de una palabra, tampoco se puede utilizar el diccionario de palabras en Inglés para expandir.

Id	Greedy	Samurai	Exp. desde Greedy	Exp. desde Samurai
menBarFile	men–bar–file	men–bar–file	menu background file	menu background file
menItNew	men–it–new	men–it–new	menu it new	menu it new
menBarEdit	men–bar–edit	men–bar–edit	menu background editor	menu background editor
menItCopy	men–it–copy	men–it–copy	menu it copy	menu it copy
jTlBar	j–tl–bar	j–tl–bar	java tool background	java tool background
jBtSave	j–bt–save	j–bt–save	java bt save	java bt save
popUpMenu	pop–up–menu	pop–up–menu	pop up menu	pop up menu
imIcPrint	im–ic–print	im–ic–print	images icon printing	images icon printing
PRGName	prg–name	prg–name	program name	program name
jLabelColTex	j–label–col–t–ex	j–label–col–tex	java label color text exit	java label color text
bckCol	b–c–k–col	bck–col	bar copy kilo color	background color
TEXTArea	text–area	text–area	text areas	text areas
textAREAErrors	text–area–errors	text–area–errors	text areas errors	text areas errors
jScrPANtxtAr	j–scr–pant–xt–ar	j–scr–pan–txt–ar	java scrollbar pant xt areas	java scrollbar pan text areas
selCl	sel–c–l	sel–cl	select copy literalizes	select close
fc	f–c	fc	finish copy	fc
flreader	fl–reader	flreader	file reader	flreader
ioe	io–e	ioe	icon editor	invoiced
printText	print–text	print–text	printing text	printing text
ptjob	pt–job	ptjob	printing job	ptjob
pg	pg	pg	print graphics	print graphics
linNum	l–in–nu–m	lin–num	lab in nu menu	linearised numerically
i	i	i	images	images
jLbFind	j–lb–find	j–lb–find	java lb find	java lb find
jTxFind	j–tx–find	j–tx–find	java text find	java text find
jPanBut	j–pan–but	j–pan–but	java pan but	java pan but
jChoColor	j–c–ho–color	j–cho–color	java copy ho color	java choose color
btAcept	bt–ace–pt	bt–acept	bt ace printing	bt acept

Tabla 4.5: Resultados destacados del Análisis Realizado a los Ids extraídos de Editor.java

También existen casos de abreviaturas mal expandidas, el más común es bar → background, aquí simplemente el algoritmo de expansión entiende que bar es una abreviatura y la expande con un candidato fuerte que se encuentra en el listado de frases capturadas background.

La mayoría de los ids analizados (ver tabla 4.5 columnas Expansión desde Greedy y Expansión desde Samurai), se corresponden a distintos elementos de interacción que posee el programa con el usuario. De estos elementos se pueden enumerar, áreas de texto, barras de desplazamiento, menu, copiar, pegar, iconos, imágenes, colores. Estos elementos sin duda forman parte del Dominio del Problema en el programa Editor.java.

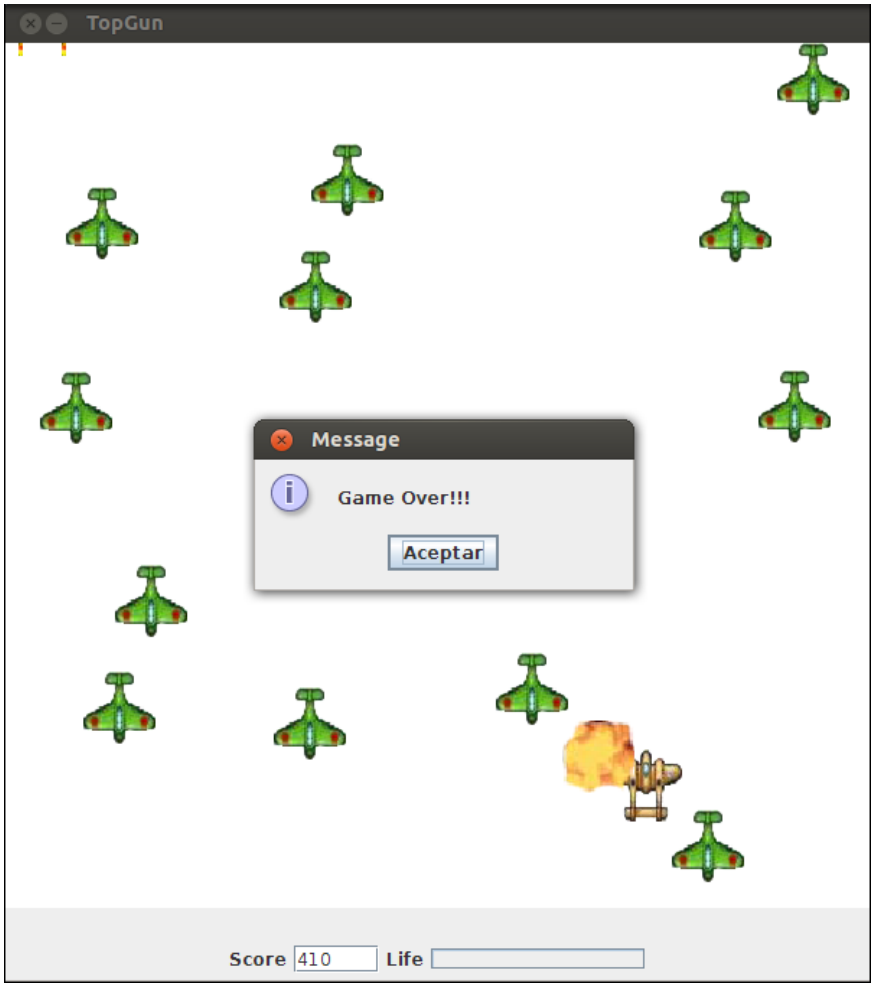


Figura 4.16: Captura del juego Top Gun programado en JAVA

4.6.3. Top Gun

Continuando con los casos de estudio, el próximo es un programa JAVA llamado TopGun.java. Cuando este programa se ejecuta aparece un juego de aviones (ver figura 4.16), en este juego el usuario conduce un avión y es manejado desde el teclado. El objetivo consiste en en derribar la mayor cantidad de aviones enemigos y un contador suma puntos por cada avión derribado. Este programa, posee un módulo de aproximadamente de 600 líneas que van a ser analizadas por la herramienta IDA.

Al igual que el caso de estudio anterior (Editor de texto), debido a que el programa TopGun.java contiene muchos ids, en la tabla 4.6 se lista el análisis de ids más relevantes de TopGun.

Análisis de Resultados

A diferencia de los casos de estudios antes vistos, en la tabla 4.6 se puede observar la presencia de ids capturados conformados por 3 palabras; algunos ejemplo de esto son: LIFEprogressBar, hitShotEnemy, hitPlaneEnemy.

En la división de ids persiste la tendencia en Greedy con respecto a Samurai, de separar mal algunos ids (ver caso de estudio anterior).

Las expansiones de ids en general están bastante precisas, cabe mencionar que los ids extraídos no presentan muchas dificultades en la traducción de las palabras completas que representan.

De los ids analizados (ver tabla 4.6 en las columnas Expansión desde Greedy y Expansión desde Samurai), las palabras que más frecuentemente aparecen están asociados al juego. De estos palabras se pueden nombrar, disparos, aviones, enemigos, imágenes, actualizar pantalla, movimiento. Estas palabras forman parte del Dominio del Problema en el que está enmarcado el programa TopGun.java.

Id	Greedy	Samurai	Exp. desde Greedy	Exp. desde Samurai
strt_but	str–t–but	strt–but	start topgun but	start but
scoLabel	sco–label	sco–label	score label	score label
scotext	sco–text	sco–text	score text	score text
lifelabel	life–label	life–label	life label	life label
LIFEprogressBar	life–progress–bar	life–progress–bar	life progress background	life progress background
init	init	init	initiate	initiate
sn	sn	sn	shoot number	shoot number
pm	pm	pm	plane movement	plane movement
shot	shot	shot	shoot	shoot
lnEne	ln–en–e	ln–ene	launch enemies enemies	launch enemies
rfshScreen	rfs–h–screen	rfsh–screen	refresh hit screen	refresh screen
shoImage	sho–image	sho–image	shot images	shot images
eneImage	en–e–image	ene–image	enemies enemies images	enemies images
bangImage	bang–image	bang–image	bang images	bang images
tg	tg	tg	topgun	topgun
hitPlaneEnemy	hit–plane–enemy	hit–plane–enemy	height planes enemys	height planes enemys
intExp	int–exp	int–exp	int exploit	int exploit
enNum	en–nu–m	en–num	enemies number movement	enemies number
getYac	get–y–ac	get–yac	get yin active	get y axe column
getXar	get–xa–r	get–xar	get xa row	get x axe row
ie	ie	ie	initiate enemies	initiate enemies
updatePlane	update–plane	update–plane	update planes	update planes
updateShot	update–shot	update–shot	update shoot	update shoot
hitShotEnemy	hit–shot–enemy	hit–shot–enemy	height shoot enemys	height shoot enemys
updatePlane	update–plane	update–plane	update planes	update planes
updateShot	update–shot	update–shot	update shoot	update shoot
keyReleased	key–released	key–released	keylistener released	keylistener released

Tabla 4.6: Parte del Análisis Realizado a los Ids de TopGun.java

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] Alfred V. Aho, Jeffrey D. Ullman, and John E. Hopcroft. *Data structures and algorithms / Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman.* Addison-Wesley Reading, Mass, 1983.

[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] Ramez A. Elmasri and Shankrant B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1999.

[15] David W Embley. Toward semantic understanding: an approach based on information extraction ontologies. In *Proceedings of the 15th Australasian database conference-Volume 27*, pages 3–12. Australian Computer Society, Inc., 2004.

[16] 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.

[17] Henry Feild, David Binkley, and Dawn Lawrie. An empirical comparison of techniques for extracting concept abbreviations from identifiers. In *Proceedings of IASTED International Conference on Software Engineering and Applications.*, 2006.

[18] 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.

[19] Fangfang Feng and W. Bruce Croft. Probabilistic techniques for phrase extraction. *Inf. Process. Manage.*, 37(2):199–220, 2001.

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

[21] Wilhelm Hasselbring, Andreas Fuhr, and Volker Riediger. First international workshop on model-driven software migration (mdsm 2011). In *CSMR '11 Proceedings of the 2011 15th European Proceedings of the 2011 15th European Conference on Software Maintenance and Reengineering (CSMR 2011)*, pages 299–300, Washington, DC, USA, März 2011. IEEE Computer Society.

[22] 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.

[23] IEEE. Ieee standard for software maintenance. *IEEE Std 1219-1998*, page i–, 1998.

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

[25] 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.

[26] D. Lawrie, C. Morrell, H. Feild, and D. Binkley. What's in a name? a study of identifiers. In *Program Comprehension, 2006. ICPC 2006. 14th IEEE International Conference on.*, page 3–12, June 2006.

[27] Dawn Lawrie, Henry Feild, and David Binkley. Syntactic identifier conciseness and consistency. In *SCAM*, page 139–148. IEEE Computer Society, 2006.

[28] Dawn Lawrie, Henry Feild, and David Binkley. Quantifying identifier quality: an analysis of trends. *Empirical Software Engineering*, 12(4):359–388, 2007.

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

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

[31] T.A. Standish. *Data structure techniques.* Computer Sciences. Addison-Wesley, 1980.

[32] 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.

[33] 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.

[34] 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.

[35] Tim Tiemens. Cognitive models of program comprehension, 1989.

[36] 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.

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