DAVIS ELIAS AROSEMENA TREJOS

GRUPOS DE TRABAJOS PARA LA ENSEÑANZA DE LA PROGRAMACIÓN BASADOS EN ONTOLOGÍA Y EN LA SEMÁNTICA DEL CÓDIGO FUENTE

Tesis presentada a la Facultad de Ingeniería en Sistemas Computacionales de la Universidad Tecnológica de Panamá para la obtención del Título de

Maestría en
Ciencias de Tecnología de la Información y Comunicación

Acesores:
Dr. Sergio Crespo
Dr. Clifton Clunie

Panamá
2012

II

Con mucho amor, dedico este trabajo final a las siguientes personas:

A mi mamá Lucila Trejos, por recibir de ella todo su amor y comprensión, apoyándome y dándome ánimos cuando lo he necesitado en la vida.

A mis hermanos(as) y sobrinos(as), por sus apoyos y cariños, por acompañarme en los momentos más importantes de mi vida, por preocuparse y ayudarme en las dificultades que se me han presentado.

A todos mis amigos, que con su cariño y aprecio forman o han formado parte de mi vida, especialmente, mi amiga Laura Herrera, por su compañía al inicio y final de este proyecto; de igual forma, a todos los que de una u otra forma me han brindado su ayuda.

la recompensa del esfuerzo debe llegar hasta la satisfacción espiritual por lo alcanzado, solo así, el dolor del sacrificio será olvidado...
Anónimo

Agradecimientos

Primeramente, agradezco a Dios por toda la fe, fortaleza y amor que me ha dado a lo largo de mi vida.

Agradezco a mis profesores asesores, Dr. Segio Crespo y Dr. Clifton Clunie, por ser más que asesores, unos guías comprometidos con que yo presentara un buen trabajo.

A toda mi familia, por estar conmigo en este largo trabajo, brindándome su compañía comprensión y apoyo, a mis hermanos Jimmy y Alina, por acogerme en sus casas mientras cursaba dichos estudios.

A mis compañeros de clases, especialmente Belén y Xavier, por sus apoyos a lo largo de este programa, en la que fueron fuente de motivación y esperanza en seguir luchando juntos y terminar exitósamente este compromiso.

A todos mis amigos y todas las personas que de una u otra forma me apoyaron, que me dieron ánimos, o simplemente me preguntaban por el avance de mi trabajo sin saber que me ayudaban en este camino, gracias a todos y que Dios me los bendiga.

Resumen

Existen muchas técnicas, métodos y herramientas para la enseñanza de la programación; eso se debe a la complejidad que representa hacer que un estudiante adquiera y aplique dichos conocimientos. La utilización de grupos de trabajo como método didáctico, puede ser efectivo dentro del aula de clase, ya que ayuda a los estudiantes a generar y adquirir nuevos conocimientos en un menor tiempo. Sin embargo, estos grupos al ser formados sin considerar ningún criterio, puede provocar un efecto negativo en el proceso de enseñanza-aprendizaje de la programación. Este trabajo propone una herramienta para la formación de grupos de trabajo basado en la semántica del código fuente (SOFORG). Esta semántica esta basada en métricas extraídas de las preferencias, estilos y buenas prácticas de programación; además de una evaluación del problema. Estas métricas permiten cuantificar cada característica de programación y así poder comparar a los estudiantes entre sí. Todo esto se obtiene a través de un análisis estático del código fuente; que pertenecen a problemas de programación que los estudiantes desarrollan, ya sea individual o grupal. De esta manera, se tendrá un registro de los estudiantes con la información extraída, esto permite evaluar la mejor formación de grupos de trabajo en un curso determinado. Las formaciones de grupos están basadas en estilos de programación, capacidades, grupo de par o con tutor; de igual forma, esta herramienta permitirá hacer una evaluación del estado de los estudiantes en un curso y así, sugerir cuál es la mejor formación de grupos, así como también ayudará a mantener una estadística del porcentaje de avance de los estudiantes, en la que se podrá evaluar la efectividad de las técnicas, metodologías y tipos de grupos utilizados por el profesor.

Palabras Claves: *Grupos de trabajos, enseñanza de la programación, estilo de programación, analizador estático de código, ontología*

Abstract

There are many techniques, methods and tools for teaching programming, this is due to the complexity that a student is to acquire that knowledge and put them into practice. The use of workgroups as a teaching method can be effective in the classroom because it helps students to generate and acquire new knowledge in less time, but these groups to be formed without taking into account some may cause an adverse effect on the teaching-learning process of programming. This paper proposes a tool for the formation of workgroups based on the semantics of source code (SOFORG). This semantics is based on metrics extracted from the preferences, programming styles and good programming practices, in addition to an assessment of the problem, and these metrics to quantify each programming features so you can compare students to each other. All this is achieved through a static analysis of source code, which belong to scheduling problems that students develop, either individually or in groups. In this way, you will have a record of students with the information extracted, it evaluates the best formation of working groups in a given course. The group formations are based on programming styles, skills, peer group or work leader, and likewise, this tool will make an assessment of students in a course and thus suggest the best group formation. Apart from the above, this tool will help keep a statistic on the percentage of students' progress, which we can evaluate the effectiveness of the techniques, methods and types used by the teacher groups, in addition to other scientific research within the process of teaching and learning programming.

Keywords: *Workgroups, teaching programming, programming styles, static Code analyzer, ontology*

Índice general

Lista de Tablas

Pág.

Lista de Figuras

Lista de Abreviatura

Pág. Página

Dr. Doctor

PhD. Doctor of Philosophy

et. al y otros

e.g. Por ejemplo

Lista de Siglas

FISC Facultad de Ingeniería en Sistemas Computacionales

UTP Universidad Tecnológica de Panamá

SOFORG Sistema para la Formación de Grupos de Trabajos

POO Programación Orientada a Objetos

POA Programación Orientada a Aspectos

IDE *Integraded Development Environment*

CAPATEC Cámara Panameña de Tecnologías y Comunicaciones

W3C *World Wide Web Consortium*

RAE Real Academia Española

XML *Extensible Marckup Language*

RDF *Resource Description Framework*

RDFS *Resource Description Framework Schema*

OWL *Ontology Web Language*

DL *Description Logics*

GIC *Gramática Independiente de Contexto*

NP *nondeterministic polynomial time*

BNF *nondeterministic polynomial time*

Glosario

1. **Programación:** es el proceso de diseñar, escribir, depurar y mantener el código fuente de programas computacionales.

2. **Código fuente:** es un conjunto de líneas de texto que son las instrucciones que debe seguir la computadora para ejecutar dicho programa.

3. **Lenguaje de programación:** es un idioma artificial diseñado para expresar computaciones que pueden ser llevadas a cabo por máquinas como las computadoras.

4. **Semántica:** se refiere a los aspectos del significado, sentido o interpretación del significado de un determinado elemento, símbolo, palabra, expresión o representación formal.

5. **Algoritmo:** es un conjunto preescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien deba realizar dicha actividad.

6. **Programación Orientada a Objetos:** es un paradigma de programación que usa objetos y sus interacciones, para diseñar aplicaciones y programas informáticos.

7. **Clase:** definiciones de las propiedades y comportamiento de un tipo de objeto concreto. La instanciación es la lectura de estas definiciones y la creación de un objeto a partir de ellas.

8. **Herencia:** es la facilidad mediante la cual la clase D hereda en ella cada uno de los atributos y operaciones de C, como si esos atributos y operaciones hubiesen sido definidos por la misma D. Por lo tanto, puede usar los mismos métodos y variables publicas declaradas en C. Los componentes registrados como ”privados”(private) también se heredan, pero como no pertenecen a la clase, se

mantienen escondidos al programador y sólo pueden ser accedidos a través de otros métodos públicos. Esto es así para mantener hegemónico el ideal de OOP.

9. **Objeto:** entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos) los mismos que consecuentemente reaccionan a eventos.

10. **Método:** algoritmo asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena tras la recepción de un ”mensaje”. Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto, o la generación de un .^{ev}entoçon un nuevo mensaje para otro objeto del sistema.

11. **Evento:** es un suceso en el sistema (tal como una interacción del usuario con la máquina, o un mensaje enviado por un objeto). El sistema maneja el evento enviando el mensaje adecuado al objeto pertinente. También se puede definir como evento, a la reacción que puede desencadenar un objeto, es decir la acción que genera.

12. **Mensaje:** una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros asociados al evento que lo generó.

13. **Propiedad o atributo:** contenedor de un tipo de datos asociados a un objeto (o a una clase de objetos), que hace los datos visibles desde fuera del objeto y esto se define como sus características predeterminadas, y cuyo valor puede ser alterado por la ejecución de algún método.

14. **Ontología:** hace referencia a la formulación de un exhaustivo y riguroso esquema conceptual dentro de uno o varios dominios dados; con la finalidad de facilitar la comunicación y el intercambio de información entre diferentes sistemas y entidades.

15. **Grupos de trabajos:** para este trabajo, se refiere al conjunto de estudiantes que trabajan en una asignación presentada por un profesor en las universidades.

16. **Parser:** en español conocido como analizador sintáctico. Convierte el texto de entrada en otras estructuras (comúnmente árboles), que son más útiles para el posterior análisis y capturan la jerarquía implícita de la entrada. Un analizador léxico crea tokens de una secuencia de caracteres de entrada y son estos tokens

los que son procesados por el analizador sintáctico para construir la estructura de datos, por ejemplo un árbol de análisis o árboles de sintaxis abstracta.

17. **Analizador Estático:** perteneciente al análisis estático de programas. Este análisis consiste en evaluar un programa a través del código fuente, es decir, evaluar sin correr el programa. Si se evalúa el programa a través de la ejecución del mismo, entonces se considera un análisis dinámico.

18. **Aplicación basada en conocimientos:**Es una aplicación informática capaz de solucionar un conjunto de problemas que exigen un gran conocimiento sobre un determinado tema.

19. **Inteligencia artificial:** La Inteligencia Artificial es una combinación de la ciencia del computador, fisiología y filosofía, tan general y amplio como eso, es que reúne varios campos (robótica, sistemas expertos, por ejemplo), todos los cuales tienen en común la creación de máquinas que pueden pensar.

Capítulo 1

Introducción

La tendencia del mundo se inclina hacia la informática, la mayoría de las áreas científicas están introduciendo algún tipo de lenguaje de programación que les permite generar nuevos conocimientos. La programación ha sido una de las actividades que demanda mayor capacidad de concentración, análisis, trabajo organizado, paciencia y dedicación; esto representa para los estudiantes nuevos, ciertas dificultades debido a la inmadurez, falta de conocimientos, capacidades para abordar el problema y otros factores que provocan problemas concretos dentro de la programación; como es el caso de la buena implementación de los lenguajes, la utilización correcta de las instrucciones, problemas en el desarrollo de la lógica, creación de algoritmos óptimos, entre otros. Por otro lado, existen otros tipos de dificultades relacionadas con el entorno social, heterogeneidad y problemas personales que afectan el proceso de programar y por ende el aprendizaje del mismo. Por éstas y otras razones, se han empleado esfuerzo para encontrar mejores metodologías, técnicas y herramientas que permita la enseñanza óptima de la programación, uno de estos métodos es la formación de grupos de trabajo dentro del aula de clase.El éxito de los grupos de trabajo para el aprendizaje de la programación dependerá de un conjunto de factores que influyen directa e indirectamente (Cox y Fisher, 2009); muchos de estos factores tienen que ver con las características personales que cada individuo posee; por lo que hay que considerar ciertos criterios y dependencias a la hora de formar dichos grupos con el fin de evitar resultados contraproducentes en el aprendizaje del estudiante.

Este trabajo tiene como objetivo presentar una herramienta para la formación de

grupos de trabajo basado en análisis estático de código fuente (SOFORG), en la que se extrae y almacena información de estilos de programación, preferencias y buenas prácticas a través del análisis estático de los programas que los estudiantes desarrollan; de igual forma, en caso de que existiese un modelo ontológico del ejercicio, SOFORG proporciona una evaluación sugerida del problema en base a modelos preestablecidos en una ontología. El resultado de cada análisis estático será almacenado en forma de métricas por cada estudiante; de esta manera, existirá un registro completo de los ejercicios que fueron aplicados en grupo o individual. Para la formación de grupos, la herramienta evalúa el estado actual de los estudiantes en un curso determinado, y así sugerir la mejor opción de formación de grupos; estos grupos pueden ser basados en capacidades, estilos de programación, grupo de par, grupos con tutor. Aunque existen muchos aspectos a considerar dentro de la formación de grupos para la programación, en este trabajo solo se tomará en cuenta las preferencias y estilos de programación, las capacidades individuales y el estado grupal del salón de clases que gira entorno a los parámetros mencionados.

La estructura de éste trabajo comprende un capítulo introductorio que presenta las generalidades del problema y el objetivo que se pretende alcanzar; el segundo capítulo, incluye las teorías y los conocimientos más importantes acerca del tema a tratar, en la que se habla de la enseñanza de la programación, una breve descripción de las dificultades, deficiencias y errores de los estudiantes a la hora de programar; se presenta de igual forma, un estudio de los grupos de trabajo para la enseñanza de la programación, los beneficios y problemas, aparte de los posibles tipos de grupos que son utilizados y sus respectivas ventajas y casos de uso; conceptos de compresión de programas, analizadores estáticos y ontología son descritos también en este capítulo. En el capítulo tercero se presenta el marco metodológico, el detalle de la arquitectura de la herramienta es incluido en el capítulo cuarto, donde se describe las métricas de los estilos de programación y preferencias que serán analizadas, aparte de las buenas prácticas; de igual forma se incluye todos los elementos de SOFORG con su descripción, también los métodos y algoritmos empleados para el desarrollo del mismo. El ambiente de trabajo de la herramienta, es descrito en el capítulo quinto, incluye impresiones de pantalla de la interfaz gráfica y las principales características de la herramienta. El capítulo sexto evalúa la propuesta con un caso de estudio y se observará el comportamiento de la misma. Por otro lado, el séptimo proporciona las conclusiones y recomendaciones de la investigación.

1.1. Título de la Investigación

"GRUPOS DE TRABAJO PARA LA ENSEÑANZA DE LA PROGRAMACIÓN BASADOS EN ONTOLOGÍA Y EN LA SEMÁNTICA DEL CÓDIGO FUENTE"

1.2. Antecedentes

La enseñanza de la programación ha sido una tarea tediosa debido a que es complejo para los profesores crear en los estudiantes, modelos mentales correctos, para que puedan emplear los componentes de un lenguaje de programación. En las universidades de Panamá, las carreras de ingeniería y algunas licenciaturas, requieren el aprendizaje de la programación, ya que puede representar una herramienta para la generación de conocimientos; en las carreras de computación, las exigencias en cuanto a la capacidad de programación que deben tener los estudiantes es mucho mayor. Aunque los lenguajes de programación han evolucionados conjuntamente con los IDEs (Ambiente de Desarrollo Integrados), sigue siendo una tarea compleja el escribir líneas de código.

Desde el inicio en que los lenguajes de programación fueron llevados a las universidades, se han empleado esfuerzos para la formulación de metodologías y herramientas que sirvan de soporte en el proceso de enseñanza-aprendizaje de la programación; tal es el caso de Truong et al. (2009), que presentan un framework para el análisis estático, en donde es usado para las prácticas de los estudiantes, permitiendo escribir mejor sus programas; esto representa una asistencia al profesor en la clase, ya que le permite comprender la situación real del estudiante. Este framework usa métricas de ingeniería de software y comparaciones de modelos para evaluar los programas de los estudiantes y en caso de encontrar errores, notifica al estudiante y sugiere una posible solución.

De igual forma, existen otras investigaciones como las que presentan Kaila et al. (2005), donde tienen una herramienta para la visualización de programas en ejecución, en donde los estudiantes pueden entrar a una plataforma y desarrollar ejercicios; esta herramienta evalúa y almacena los resultados del mismo, conjuntamente con la información del estudiante que la desarrollo. Morimoto et al. (2006) y Juedes (2005)

son autores que también han desarrollado herramientas de apoyo al profesor en este proceso. Existen técnicas que plantean mejorar la habilidad del estudiante en la programación a través de buenas prácticas que le permitan desarrollar aptitudes en la comprensión de programas, así como lo plantean Gray et al. (2005); por otro lado, Black (2006) propone un método de enseñanza *Cognitive Apprenticeship Learning* que ayuda al estudiante a crear sus propios conocimientos llevándolo a una situación en el que se encuentre frente a una dificultad, esto lo estimulará a crear sus propias soluciones ya sea en grupo o individual.

La utilización de grupos de trabajos, en muchas de las herramientas y metodologías presentadas, representa grandes beneficios, ya que éstos, van ligados a las bondades que se tiene de la utilización de los grupos de trabajo. En los grupos de trabajos se dá una mayor generación y aprendizaje de conocimientos, conjuntamente con un tiempo menor necesario para realizar las tareas asignadas en el grupo; esto se debe a que como el estudiante tiene un contacto directo con otra(s) persona(s), se comparte conocimiento y se logra aclarar dudas en conjunto, permitiéndoles desarrollar sus habilidades con mayor efectividad, ya que pueden tener el respaldo de sus compañeros y así alcanzar el aprendizaje significativo o permanente. De esta manera, el trabajo en grupo dentro del aula de clase es como si el estudiante tuviese un tutor o asesor de manera constante, que es representado por el resto del grupo. Muchas veces se logra que estos grupos trabajen de manera cooperativa, es decir, se desarrolla una mayor coacción y se obtienen mejores resultados en donde los estudiantes se preocupan por el aprendizaje de sus compañeros. Por otro lado, los grupos de trabajo promueve el desarrollo de aptitudes para la colaboración, comunicación, relación intrapersonal y otros.

No siempre se espera buenos resultados cuando se utilizan grupos de trabajo, ya que puede darse casos tales como: la tarea de programación recae sobre uno o varios estudiantes, exista cierta incompatibilidad de trabajo dentro de los grupos, todos los miembros del grupo presentan capacidades muy por debajo de lo necesario para realizar la tarea, no se ha desarrollado el trabajo colaborativo por lo que existe poca comunicación y coordinación. Todas estas dificultades causan el fracaso de los grupos de trabajo, debido a que los estudiantes se retrasan o no avanzan en su aprendizaje; por consiguiente, la aplicación de ésta técnica puede llegar a ser contraproducente; debido a esto, existen investigaciones que han abordado esta problemática, tal es el caso de de Faria et al. (2006), ellos proponen formar grupos de estudiantes para el

aprendizaje colaborativo de la programación en el que la formación de grupos está basado en los estilos de programación. Otra investigación, se presenta en Castro et al. (2008), en la que hacen un estudio sobre cómo mejorar la preparación del estudiante de programación para que pueda trabajar en grupos; ellos proponen un *Progression Learning Scheme*, que es un esquema de progreso para la enseñanza de la programación a través de ciertos tipos de asignación y su forma de desarrollar; yendo de lo individual a lo grupal.

Como se observa, existen varias investigaciones que plantean como facilitar el proceso de enseñanza-aprendizaje de la programación; sin embargo, pocas están orientadas a la problemática de formar grupos de trabajos tomando en cuenta ciertos criterios, esto provoca que no se le saque el máximo provecho a las herramientas y metodologías presentadas.

1.3. Planteamiento del problema

El poco rendimiento y capacidades de los estudiantes en el área de la programación es una problemática que afecta las empresas a la hora de contratar personal egresados de las universidades; por lo tanto, esto repercute en las sociedades que intentan acoger un ambiente tecnológico, como es el caso de Panamá; ya que las mayorías de las áreas del conocimiento científico y tecnológico giran en torno a la utilización de lenguajes de programación. Según estudio realizado por Hewitt (2010), el cargo de Analistas y Programadores, representa el de mayor importancia en agregar nuevos empleados entre las empresas panameñas, de igual forma, es el cargo con mayor dificultad en encontrar personal capacitado y es el de mayor deserción laboral. Por otro lado, información proporcionada por la Universidad Tecnológica de Panamá (UTP) [1], indica que de un total de 799 estudiantes que entraron al curso de Desarrollo de Software II (curso básico de lenguaje Java), más del 40 % no aprobaron la materia, este porcentaje incluye cursos del año 2009 y 2010 de todos los centros regionales. De esta manera, se observa una problemática nacional, ya que por un lado, el mundo laboral demanda más especialistas en programación capacitados, y por otro,

[1] Información proporcionada por la Ing. Yadira Vargas de la Dirección de Tecnología de Información y Comunicación (DITIC). Universidad Tecnológica de Panamá. Noviembre-2011. Vea información completa en el Anexo 1.

la baja matricula en estas carreras y los pocos que cursan dichas carreras, presentan ciertas dificultades en las materias de programación, como se observo en la materia de Desarrollo de Software II en la UTP.

Abordar estas problemáticas no es una tarea fácil, si bien es cierto que se han empleado esfuerzos para remediarlo, no se han obtenido los resultados esperados; la formación de grupos de trabajo en las clases de programación puede ayudar en dicha problemática; sin embargo, al no considerar ciertos criterios no se obtienen resultados significativos, debido a que existen problemas que se originan dentro del trabajo en grupo, tales como: la tarea de programación recae sobre uno o varios estudiantes, exista cierta incompatibilidad de trabajo dentro de los grupos, todos los miembros del grupo presentan capacidades muy por debajo de lo necesario para realizar la tarea, no se ha desarrollado el trabajo colaborativo por lo que existe poca comunicación y coordinación. Por lo tanto, se plantea el problema de que en Panamá no existe un sistema que dé soporte al profesor en la enseñanza de la programación, capaz de facilitar la formación de grupos de trabajos como método didáctico; todo esto, tomando en cuenta ciertos criterios y consideraciones que optimizaran el resultado del trabajo en grupo.

1.4. Objetivos

1.4.1. Objetivo General:

- Crear una herramienta que facilite la formación de grupos de trabajo para el proceso de enseñanza-aprendizaje de la programación.

1.4.2. Objetivos Específicos:

- Estudiar los temas asociados con el proceso de enseñanza-aprendizaje de la programación y la utilización de los grupos de trabajo como método didáctico.
- Extraer la semántica de los programas de los estudiantes a través del análisis estático del código fuente.

- Cuantificar y almacenar en forma de métricas, los estilos, preferencias y capacidades de programación, que son extraídas de la semántica del código fuente.
- Implementar una estructura ontológica para la evaluación académica del código fuente de los estudiantes.
- Elaborar reglas de decisión para la formación de grupos de trabajos basado en la información que ya ha sido almacenada.

1.5. Justificación

Estudios muestran que utilizar como método didáctico, la formación de grupos de estudiantes en el proceso de enseñanza-aprendizaje de la programación, es de mucha utilidad y beneficio, pues le brinda al profesor un soporte en su labor, en la que el estudiante aprende de manera colaborativa, generando nuevas ideas y adquiriendo conocimientos en un menor tiempo; esto permitirá el desarrollo del aprendizaje significativo en medio de las interacciones producidas en el grupo. Por otro lado, la formación de grupos en la clase, no solo estimula el aprendizaje de la programación, sino el desarrollo de aptitudes para la colaboración, comunicación, relación intrapersonal y otros. De esta manera, el resultado de esta investigación permitirá proponer una herramienta que dé soporte al profesor en la enseñanza de la programación, a través de la ayuda en la formación de grupos de trabajos basados en criterios y consideraciones; tomando en cuenta criterios y consideraciones a la hora de formar grupos los estudiantes pueden generar y adquirir aun más, nuevos conocimientos en la programación; uno de los estudios que evidencia ésta teoría es el de de Faria et al. (2006), en la que forman a los estudiantes basados ciertas características tales como, capacidades y estilos de programación (como ya fue explicado). De esta manera, se podrá obtener grupos mejor compenetrados, ayudando a subsanar los problemas del trabajo grupal (3.2.7).

De igual forma, esta herramienta facilitará el análisis de la información extraída por parte del profesor, en la que pudiera ser útil para la identificación de deficiencias basadas en buenas prácticas de los estudiantes (e.g. inicialización de variables, variables que no han sido declaradas private o protected, otros), que en la mayoría de los casos se vuelve una tarea tediosa para el profesor; también ayudará a mantener

una estadística del porcentaje de avance de los estudiantes que permitirá evaluar la efectividad de las técnicas, metodologías, modelos de ejercicios y tipos de grupos utilizados por el profesor; además de otras investigaciones científicas dentro del proceso de enseñanza-aprendizaje en la programación. Todo esto contribuirá a crear un ambiente de investigación dentro de las universidades de Panamá en donde se genere conocimiento regional; esto dará como resultado que se creen soluciones a problemas específicos dentro de la enseñanza de la programación y así mejorar las competencias de los estudiantes que podrán desempeñarse en el exigente mercado laboral.

1.6. Alcances

El resultado de la investigación será la presentación de una herramienta de escritorio, dirigida a los profesores de programación básica, para ser utilizada como un sistema de soporte a la decisiones; esto quiere decir que, el resultado de esta herramienta sugerirá la mejor opción de una posible formación de grupos en la labor docente; de esta manera, la herramienta comprende un prototipo funcional en la que se incluye solo las funciones principales del mismo y será útil en las universidades en la que existan cursos de programación básica en Java. Esta herramienta analizará estáticamente sólo, códigos en lenguaje Java en la que se incluye el paradigma de programación POO y no otros como POA.

Por otro lado, aunque existen muchas variables a considerar dentro de la formación de grupos, esta investigación solo contemplará las de formación de grupos basados en estilos de programación, preferencias de estructuras y niveles de capacidades de los estudiantes; de igual forma, se considerará el estado actual del curso completo en los criterios mencionados, para así sugerir el mejor tipo de formación.

Capítulo 2

Metodología de la investigación

Este capítulo describe el conjuntos de pasos y procedimientos que integran la metodología a seguir, para llevar a cabo la investigación propuesta; de igual forma, se describe los procedimientos de evaluación, esto involucra las hipótesis que se desean comprobar.

2.1. Metodología general

Para la elaboración de esta propuesta, se contempla una metodología a seguir que será enumerada a continuación:

1. Esta metodología propone la definición de un problema y los objetivos que se intentan alcanzar con el mismo, ya descrito en el capítulo 1; de esta manera, se hace un estudio de lo acontecido dentro del área de la enseñanza de la programación y las herramientas que sirvan como apoyo en la labor del profesor.

2. Una vez se presente el estado actual que gira entorno a la investigación, se procede a hacer un estudio de los principales conceptos y tecnologías más importantes que pertenecen al tema a tratar, tales como: un estudio de la situación actual de los estudiantes que aprenden programación, en donde se identificarán

sus errores comunes, deficiencias y dificultades; además de evaluar algunas metodologías y herramientas relacionadas a la enseñanza de la programación; analizar aspectos de la formación de grupos de trabajo y su efecto en la enseñanza de la programación; proponer reglas de decisión y escenarios para la formación de grupos de trabajos.

3. Una vez se haya estudiado los conceptos necesarios, se procede a identificar los parámetros de análisis provenientes del código fuente de los estudiantes, ésto, basado en las necesidades y requerimientos que se pretenden alcanzar con esta investigación.

4. A partir de ésto, se empleará una metodología de desarrollo de software, el cuál se denomina: Proceso de Desarrollo por Etapas, detallado en la sección 2.2; aquí se dividirá el proceso de construcción del prototipo funcional en tres grandes fases; que contemplará un módulo de programa por cada fase. Se utilizará lenguaje de programación Java, Gramática Independiente de Contexto (GIC) y OWL; además de otras herramientas y frameworks que serán descritas más adelante. A continuación se detalla la división del desarrollo de la herramienta:

   a) **Primer módulo:** contempla la construcción del analizador estático de código fuente, conjuntamente con la estructura de datos de los componentes del código y de las métricas a evaluar; también comprende la creación de los modelos ontológicos de cada ejercicio de programa y su respectiva validación.

   b) **Segundo módulo:** comprende la extracción de las métricas y preferencias, que posteriormente serán almacenados, además de extraer los componentes del programa en OWL, que serán unidos con la ontología base de los modelos ontológicos; ésto dará como resultado la presentación de la evaluación del código

   c) **Tercer módulo:** representa la formación final de los grupos de trabajos en la que se crea se plantea un algoritmo para la búsqueda de la mejor opción, ésto evaluando la complejidad del mismo; también este módulo consiste en la elaboración de las posibles reglas de decisión y los principios lógicos para la formación de los mismos.

Se plantea la construcción de ciertos diagramas de ingeniería de software, conjuntamente con la documentación de los procedimientos y algoritmos más importantes, además de presentarse las vistas de las interfaces gráficas del mismo.

5. Una vez concluida la herramienta, se presenta un esquema de utilización donde se incluye la descripción de los casos de estudios, opciones disponibles y operaciones existentes que el profesor puede realizar.

6. La documentación de la herramienta nos llevará a realizar el caso de estudio, conjuntamente con la evaluación y validación de la herramienta; para dicha validación, se propondrá una evaluación formal, con estudiantes de programación, la metodología de evaluación se detalla en la sección 2.3; de esta manera, la validación nos llevará a comprobar las hipótesis que serán propuestas en la sección 2.4.

2.2. Metodología de desarrollo por etapas

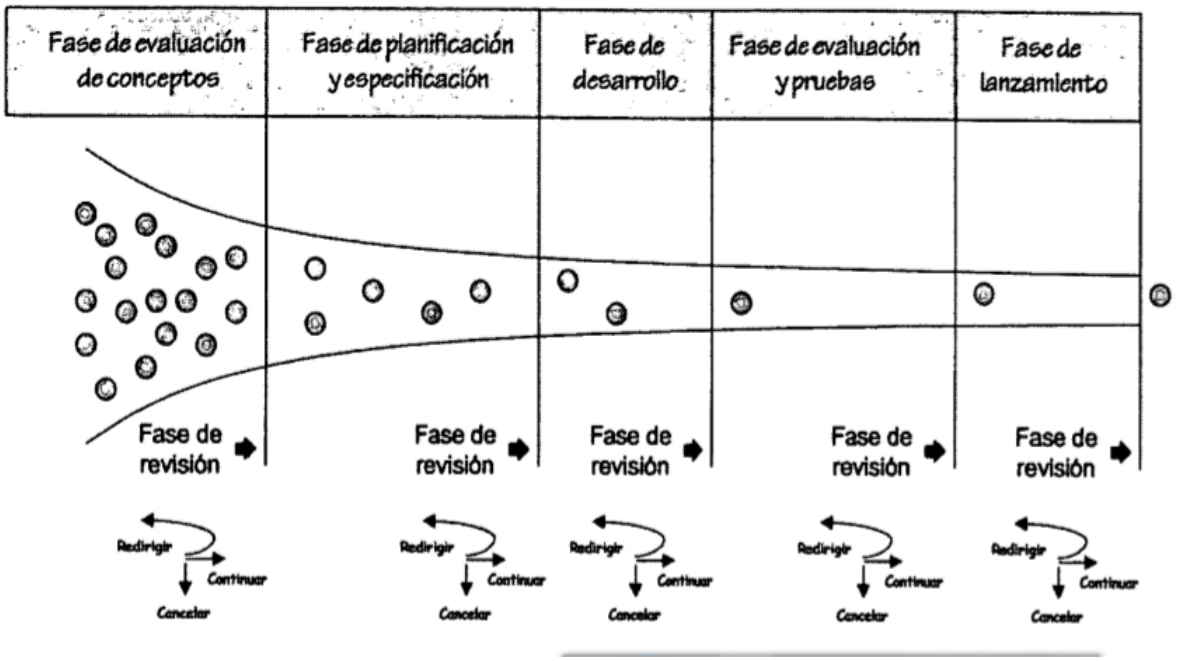
El proceso de desarrollo por etapas es uno de los modelos empleado en la ingeniería de software para el desarrollo de software ágil, en donde se busca minimizar el tiempo y los riesgos en el proceso de desarrollo de software, ya que con esta metodología, el ciclo de vida del proyecto se dá de manera iterativa, es decir, se aplica todas las etapas del modelo al módulo que se esta desarrollando.

2.2.1. Etapas del proceso de desarrollo

Las etapas del proceso de desarrollo pueden variar ligeramente en función del tipo de producto a desarrollar, sin embargo se pueden señalar una serie de características comunes para todo tipo de empresa y productos. A continuación la descripción de las etapas o fases de dicho proceso basado en Mena et al. (1999):

1. **Fase de evaluación de conceptos:** Los conceptos de esta etapa consisten en evaluar en el menor tiempo posible las diferentes oportunidades y comenzar el proceso de desarrollo también en el menor tiempo posible. El resultado de esta

Figura 2.1: Representación esquemática de las fases del proceso de desarrollo por etapas.



Fuente: Mena et al. (1999)

etapa es una evaluación del concepto y de las oportunidades del nuevo producto que es presentado al comité de revisión, quién decidirá si es conveniente asignar un equipo y los recursos necesarios para desarrollar ese concepto. En esta fase suelen intervenir un reducido número de personas dirigidas por un líder que es quien asume la responsabilidad del proyecto, no existiendo todavía un verdadero equipo multifuncional de desarrollo.

2. **Fase de planificación y especificación del producto:** Los objetivos de esta etapa son los siguientes: Definición del producto; identificación de las ventajas competitivas del producto; identificación de sus funciones básicas; determinación de la factibilidad del proceso de desarrollo; verificación, con mayor grado de exactitud y detalle, de las estimaciones realizadas en la fase de concepto; planificación del proceso de desarrollo hasta la culminación del proyecto.

3. **Fase de desarrollo:** El objetivo de esta etapa es desarrollar un producto de acuerdo con el programa aprobado en la fase previa. En esta fase se lleva a cabo el diseño detallado del producto y la mayor parte de las actividades de desarrollo. Simultáneamente se puede comenzar el desarrollo del proceso de

fabricación, diseño de las pruebas a realizar al producto, planificación del servicio al cliente, etc. Es en esta fase donde se consumen la mayor cantidad de recursos. La aprobación del comité de revisión al finalizar esta fase indica que el producto está listo para la fase de evaluación y pruebas y que el plan de pruebas establecido es el adecuado.

4. **Fase de pruebas y evaluación:** En esta etapa se procede a realizar las pruebas y simulaciones necesarias al producto, realizándose los preparativos necesarios para el comienzo de la producción y el lanzamiento al mercado del nuevo producto. Esta fase concluye cuando el producto ha superado las pruebas, el proceso de fabricación es el adecuado y se ha redactado un plan de lanzamiento del producto. El comité de revisión decide en este punto si el producto está listo para su lanzamiento, autorizando su fabricación, comercialización, distribución inicial y servicios de apoyo necesarios. Es en este momento donde la empresa se asegura de que el producto resultante cumple con los requisitos de calidad establecidos por la empresa y demandados por el mercado.

5. **Fase de lanzamiento:** Incluye la producción a gran escala, el lanzamiento del producto y su distribución inicial. La fase de revisión al final de esta etapa verifica el éxito de estas actividades, evaluando la aceptación del producto por el mercado. También se decide sobre posibles modificaciones y rediseños de productos y procesos, para adaptarlo mejor a la demanda del mercado. Esta revisión generalmente se realiza a los seis meses de lanzamiento del producto.

Para el desarrollo de esta propuesta de investigación, se empleará esta metodología con ciertas adecuaciones, ya que se trata de un proyecto académico y no comercial; sin embargo, se seguirán las fases por módulo desarrollado, como se presento anteriormente.

2.3. Metodología de evaluación

Para la evaluación de la herramienta, se presenta una metodología que será descrita por pasos a continuación:

1. Se seleccionará grupos de estudiantes que cursen materias de programación básica en Java, en cualquiera universidad local en Panamá.

2. Se les pedirá que desarrollen, de manera individual, 2 problemas de programación; uno, para operaciones básicas de Java y el otro, que requiera operaciones en POO. Estos problemas deberán ser desarrollados en el salón de clase, uno por cada día en la que los estudiantes deben incluir en el código, en forma de comentario, su ID y nombre; aparte de el nombre de su profesor y curso.

3. Los estudiantes deberán enviar su código fuente en archivos con extensión *java*, a un correo electrónico que permita contenerlos todos.

4. Una vez que se tengan todos los ejercicios desarrollados de los estudiantes, se procede a evaluarlos en la herramienta y almacenar la información extraída en formato de métricas; todo esto haciendo referencia a cada estudiante participante que ya se encuentran almacenados en el sistema previamente.

5. Luego, se procede a generar los grupos de estudiantes que han participado tomando en cuanta el curso al que pertenecen. El tipo de grupo a generar es el de basado en estilos de programación.

6. Los grupos generados se guardan, luego se evalúan las métricas utilizada y el algoritmo implementado para comprobar que ha sido los grupos correctos en la generación. El algoritmo implementado da como resultado datos finales que son las distancias de las métricas de cada estudiante en los grupos, por lo que ésto permitirá ver la de menor distancia y así comprobar el correcto.

Dicha evaluación permitirá comprobar la efectividad de la herramienta en la generación de grupos y así evaluar las hipótesis que han de ser formuladas en la siguiente sección.

2.4. Hipótesis de la investigación

Tomando en cuenta que la investigación presentada, propone la implementación de una herramienta para la formación de grupos de trabajos y así mejorar el proceso de

enseñanza de la programación, se propone, a continuación, dos hipótesis que deben ser evaluadas para dicha investigación:

H_0: El sistema SOFORG no mejora, el proceso de enseñanza de la programación.

H_1: El sistema SOFORG mejora el proceso de enseñanza de la programación.

La primera representa la hipótesis nula de la investigación, en la que se tiene como paradigma, que no existe una herramienta como la que se propone, en la que le dé un soporte en la labor docente ayudando en el proceso de enseñanza de la programación dentro del aula de clases; y en contraposición, la segunda, representa la hipótesis alternativa donde se indica que SOFORG mejora el proceso de enseñanza de la programación; en caso de que no se logre validar la primera, el resultado de la investigación adoptará la hipótesis alternativa.

Capítulo 3

Fundamentos teóricos y tecnológicos

El siguiente capítulo presenta una recopilación del material bibliográfico necesario para llevar a cabo esta investigación; en la que se abordan temáticas como la enseñanza de la programación y los principales problemas que se suscitan en este proceso, conjuntamente con las herramientas existentes que den soporte al mismo; de igual forma, se estudiará la utilización de grupos de trabajo como método didáctico en el proceso de enseñanza-aprendizaje de la programación, sus principales beneficios y desventajas; por otro lado, conceptos teóricos pertenecientes a analizadores estáticos de códigos, ontología y comprensión de programas serán planteados.

3.1. Enseñanza de la programación

La computación ha envuelto la vida de las personas; en donde la programación ha emergido como un esencial requisito para los científicos, matemáticos e ingenieros, así como los informáticos; por lo que el aprendizaje de la programación, utilizando diferentes lenguajes, es de competencia para la mayoría de las áreas del conocimiento; ya que permite la generación de información valiosa para sus estudios; de esta manera, cada día, se emplean esfuerzo en beneficio del aprendizaje significativo del mismo.

El perfil de una persona que programa debe ser el siguiente: tener un entendimiento y dominio del problema, conocimiento rudimentario de los conceptos de pro-

gramación, comprender por lo menos un lenguaje de programación lo suficiente como para leer y escribir instrucciones, comprender la lógica de cómo resolver un problema, además de estar apto para buscar y corregir “bugs” o errores de programación; tomando en cuenta esto, la programación no es una tarea fácil y esto se empeora en aquellos estudiantes que encuentran en el proceso de aprendizaje.

La programación es una de las actividades que demanda mayor concentración, capacidad de análisis y de ser autodidacta, memoria, paciencia, dedicación, organización, disciplina, distribución de las actividades; y, dependiendo de la magnitud del proyecto de software, tener sentido de colaboración y la capacidad de seguir flujos de trabajos. Las altas exigencia en cuento a los atributos mencionados hacen que cualquier factor que rodea al estudiante pueda influir positiva o negativamente en la actividad de programar. En el siguiente punto se describe las dificultades en el aprendizaje de la programación.

3.1.1. Dificultades en el aprendizaje de la programación

Existen muchas dificultades por la que el estudiante pasa a la hora de aprender a programar; estas dificultades están basadas en factores. Hay tres tipos de factores que según Cox y Fisher (2009), pueden influir en los estilos de programar; por consiguiente, en el proceso de aprendizaje de la programación; estos factores son: tarea específica a desarrollar, la situación que rodea la tarea y los factores internos y externos del individuo que realizará la tarea; en donde la tarea y la situación puede variar dependiendo de la percepción del individuo. Situación está formada por factores como: tiempo disponible, importancia del proyecto, cultura organizacional, ambiente, grupo de trabajo. Tarea encierra todas las características relacionadas con las especificaciones de un programa; por ejemplo, lenguaje, ambiente de desarrollo, complejidad, requerimientos. Individual, se refiere a los factores que abarca las características personales de cada programador; los factores externos son los relacionados con la educación, entrenamientos, conocimientos previos, experiencias; los internos, aquellos factores que incluyen en el individuo sin que se tenga control; por ejemplo, género, coeficiente intelectual, personalidad; la personalidad indica aquellos rasgos que no son visible pero tienen su influencia en el aprendizaje, tales como: creatividad, escrupulosidad, tolerancia, flexibilidad, amabilidad y otros. Estos factores pueden influir

negativa y positivamente en el aprendizaje de la programación; un estudiante que es agrupado con otro, que es, poco tolerante y posee un nivel mayor de conocimientos, puede provocar un sesgo en el estudiante al ver que su compañero realiza gran parte de la tarea sin consultarle ni explicarle. El factor género puede influir ya que las mujeres tienden a solicitar mayor ayuda a los profesores que los estudiantes varones; por lo que los profesores pueden prestar mayor atención a las dificultades de una estudiante que las de un estudiante varón; por otro lado, una estudiante podría sentirse cohibida y subestimarse al ambiente o cultura que la rodea en donde las actividades de la computación pertenecen a los hombres. De esta manera, un conjunto de factores pueden influir en otro y convertirse en un verdadero problema a la hora de programar; por ejemplo, un estudiante con capacidades cognitivas menores, algo introvertido, en donde posee pocas experiencias y conocimiento del tema de la programación o de algún lenguaje específico; al asignar, el profesor, una tarea que involucra mayor demanda de conocimiento, podría comprometer el aprendizaje del estudiante al no comunicar sus dificultades o pedir ayuda; el profesor, desconociendo las dificultades, no le brindará mayor motivación ni un reforzamiento necesario para el estudiante.

Por otro lado, se afirma que hay diferencias en el aprendizaje de cuál es el primer lenguaje de programación, ya que puede ser uno imperativo y otro orientado a objeto; esto se debe a que cada estudiante se hace una imagen de las condiciones y requerimientos lógicos de los programas, esto puede ser contraproducente si el estudiante no logra encajar fácilmente con el primer lenguaje enseñado; por lo que la diferencia de los lenguajes puede afectar la diferencias en el razonamiento de los estudiantes.

En cuanto al profesor, éste influye directamente en el aprendizaje del estudiante; ya que él podrá utilizar cualquiera metodología cognitiva para la enseñanza de la programación; algunas metodologías son detalladas en la investigación de Black (2006). Por lo tanto, el papel del profesor es muy importante, ya que este no solo debe proporcionar prácticas y aplicar ejercicios; sino también, brindarle al estudiante sesiones de instrucciones y teoría en la que se aborden temas de conceptos y utilización de técnicas; ésto, muchas veces se ve afectado por la diferencia de niveles de conocimiento en los estudiantes, en la que el profesor desconoce o no sabe cómo abordarlos. Por otro lado, la realización de prácticas de programación por parte de los estudiantes ayudará a acelerar el aprendizaje; sin embargo, estos deben estar acompañados de la figura del profesor, para que asesore y atienda los problemas y dudas que surjan posteriormente en los estudiantes; si, al momento en que un estudiante se encuentre

con una dificultad en el desarrollo de su práctica y no es asistido por el profesor, éste podría frustrarse y perder el interés por el tema o emplear técnicas y conceptos incorrectos;por lo que el profesor se convierte en una dificultad para el aprendizaje de la programación en los estudiante.

Algunas dificultades del aprendizaje relacionadas con factores de los estudiantes fueron mencionados; sin embargo, Morimoto et al. (2006) plantea la existencia de otros factores que rodean el profesor y afecta su papel, y por consiguiente, afecta el proceso de aprendizaje. Estos factores son: dificultades para entender y conocer la situación del estudiante en tiempo real y el no poder entender las tendencias del estudiante cuando realiza una pregunta o comete un error en tiempo real .

3.1.2. Deficiencias y errores comunes de los estudiantes

Debido a la naturaleza de la programación, existen muchas deficiencias y errores en la que los estudiantes incurren con frecuencia; esto se debe a que es difícil para el estudiante identificar sus propios errores y si lo hace, aun más corregir sus deficiencias y no volverlas a cometer.

De manera general, los estudiantes poseen deficiencias en el análisis de los problema, implementación de las funciones de un lenguaje, comprensión de los requerimientos, identificación de las variables necesarias, manejo de métodos, estructuración de las declaraciones lógicas, expresión de las ecuaciones matemáticas, diseño e implementación de las estructuras de decisión y repetición, entre otras; a medida que el estudiante avanza en conocimientos y se incluyen nuevos conceptos, pudieran aparecer las deficiencia en la aplicación de algoritmos recursivos, estructuras anidadas, array de múltiples dimensiones, programación orientada a objetos, creación de clases y manejo de parámetros, listas enlazadas, TDAs y otras.

Los errores más comunes de los estudiantes pueden ser clasificados, atendiendo a su nivel de abstracción, en: errores léxicos, sintácticos, semánticos y lógicos.

Los *errores léxicos* son aquellos en el que uno o más símbolos no son reconocidos por el token del lenguaje; ejemplo de estos errores:

- Error de escritura al invocar alguna función o utilizar palabras reservadas; ejemplo, SWITSH en vez de SWITCH.
- Al declarar el nombre de variable utilizando símbolos como #1, II°, espacios.
- Utilizar como operadores, otros símbolos que no son reconocidos por lenguajes; por ejemplo: $\%$, $\times$, $\div$.

Los *errores sintácticos*, son aquellos en donde la utilización de instrucciones son incorrectas por falta de símbolos que la componen, ubicación errónea o símbolos que se encuentran de más; ejemplo comunes de estos errores en los estudiantes:

- La falta del punto y coma al finalizar una instrucción en aquellos lenguajes que lo requieran.
- En la declaración de estructuras, los estudiantes pueden cometer el error de olvidar cerrar la estructura, utilizando $\{$, END, NEXT; en los lenguajes requeridos; de igual forma, otros símbolos y palabras reservadas pueden ser repetidos o ubicarlos en un lugar incorrecto, dos ELSE dentro de una estructura IF.
- Utilización de “ ”en vez de ’ ’ y viceversa en aquellos lenguajes que requieran dar valores a los String y cadena de caracteres.
- No incluir operadores, a la hora de declarar sentencias lógicas o utilizar el operador incorrecto como = en vez de == en algunos lenguajes que lo requieran.

Los *errores semánticos*, profundizan en el significado del conjunto de instrucciones; en donde los símbolos son aceptados por el token y las estructuras declaradas correctamente; pero con errores relacionados con la inclusión o ausencia de otras instrucciones; ejemplos comunes de estos errores en estudiantes según:

- No inicializar las variables ni vectores.
- Declarar variables y métodos sin utilizar o, declararlas dos veces.
- En Java, al comparar dos cadenas de tipo String se utiliza la instrucción `...if(cadena1.equals(‘‘prueba’’))`; sin embargo, los estudiantes pueden declarar `...if (‘‘cadena1’’ == ‘‘prueba’’)`.

- Al declararse una variable o una función que reciba como parámetros, un tipo de valor específico; el estudiante puede hacer que ésta, reciba un valor de otro tipo como: `int x = 3.4`; o, que se ejecute una división por una variable con valor 0 que no ha sido controlada.
- Desbordamiento de vectores o estructuras multidimensionales; en donde el estudiante los recorre sin considerar su capacidad de almacenamiento declarada; o hacer referencias a un punto null dentro de una estructura, como listas enlazadas, registros, vectores.

Los *errores lógicos*, son originados cuando el conjunto de instrucciones de los alumnos, hacen que el programa se comporte de una manera no deseada, yendo en contra de los objetivos. Existen muchos casos en el que el estudiante incurre en estos errores, algunos de los ejemplos más comunes:

- No incluyen “break” en las declaraciones de los CASE.
- Intercambio de operadores en las declaraciones lógicas, ejemplo: en un simple algoritmo de ordenación el estudiante cambia el operador $>$ por $<$ o viceversa, al momento de comparar los valores a ordenar, éste cambia el sentido de ordenamiento.
- Declaraciones condicionales redundantes o diferente a lo esperado; como por ejemplo: `...if((i` $<$ `j)&&(j` $>$ `i))`, otro ejemplo, cuando el estudiante desea realizar un ciclo de 10, pudiera hacer la declaración de esta manera: `...for (i=1; i<10; i++)`.
- Error de sustracción inversa, `x = y - x;` no es lo mismo que `x = x - y;`.
- Errores en el orden de las asignaciones u operaciones; ejemplo: el estudiante no guarda el valor de una variable necesitada y modifica la variable, al ser utilizada posteriormente lleva el valor nuevo.

De esta manera, los errores léxicos y sintácticos son más fáciles de identificar y corregir, ya que están soportados por la mayoría de los IDEs en el análisis estático; sin embargo, estos errores son pocos frecuentes una vez el estudiante empieza a adquirir conocimientos. Algunos errores semánticos pueden ser identificados en tiempo de

ejecución al detenerse el programa por algún error; éstos son difíciles de identificar y los lógicos aun más; por lo que requieren la atención del profesor y la utilización de herramientas especializadas.

También, existen errores específicos de la programación orientados a objetos, en el que los estudiantes pueden incurrir. A continuación se presentan algunos de éstos:

- No se incluyen clases, elementos de datos, atributos, métodos necesarios o fueron incluidas, pero no son necesarias.
- Una clase que debió ser identificada como subclase de una superclase no lo fue o identifican la herencia en otras subclases innecesariamente.
- Una clase que se debió haber definido como abstracta fue definida como una clase concreta.
- No es usado el polimorfismo en los casos necesarios.

Tomando en cuenta lo antes mencionado, la analogía de causa y efecto que representa el aprendizaje de la programación con respecto a los programas producidos, queda de esta manera: Dificultades $\Longrightarrow$ Deficiencias en el estudiante $\Longrightarrow$ Errores de programación $\Longrightarrow$ fallas de algoritmos $\Longrightarrow$ fracaso del programa; en donde el fracaso de un programa es el resultado de una falla de algoritmo; éste a su vez es el resultado de los errores de programación que a su vez, es el resultado de las deficiencias de los estudiantes y éste, de las dificultades que lo rodean. Existen muchos casos en el que no necesariamente se sigue esta analogía, ni uno es la causa de otro.

3.1.3. Herramientas para la enseñanza-aprendizaje de la programación

Como ya hemos visto, existen muchas metodologías y herramientas que dan soporte en el proceso de enseñanza-aprendizaje de la programación; a continuación se detallan algunas de las muchas existentes.

- Truong et al. (2009), presentan un framework para el análisis estático, en donde es usado para las prácticas de los estudiantes, permitiendo escribir mejor sus programas; al darle asistencia al profesor en la clase, esto le permite comprender la situación real del estudiante. Este framework usa métricas de ingeniería de software y comparaciones de modelos para evaluar los programas de los estudiantes y en caso de encontrar errores, notifica al estudiante y sugiere una posible solución.

  El framework es desarrollado para evaluar programas en Java y está integrado al Environment for Learning to Program (ELP) [1]. El ELP es un ambiente interactivo en la Web utilizado por los estudiantes para el aprendizaje de la programación; los estudiantes ingresan y el sistema les proporciona ejercicios a desarrollar, esos ejercicios se encuentran almacenados en una base de datos; al culminar el problema, ELP permite compilarlo; en caso de que no exista error, el sistema permite que el estudiante descargue la versión del programa final en archivo JAR para su ejecución. Al integrar el framework en el ELP; el framework aporta al sistema, características que le permiten identificar errores semántico y lógico, a través de un análisis estático del código fuente. También, este le proporciona al estudiante una posible solución a su error; éste análisis es basado en algoritmos específicos, ya almacenados; por lo que no acepta códigos fuentes, de problemas, que el sistema desconozca. Cuando se le presenta al estudiante un problema a resolver; es porque el sistema contiene modelos de las posibles soluciones; y en base a esos modelos se compara la solución del estudiante en busca de alguna semejanza; en caso de que el estudiante proporcione una solución correcta y que el sistema no reconozca, este tiene la opción de redirigir su solución a un profesor; y de ser aceptada, formaría parte del grupo de modelos. Otra característica es la de poder evaluar la calidad del programa en base a métricas de ingeniería de software, la complejidad ciclomática es medida a través del número de caminos independientes lineales, que contenga un modelo del programa; de esta manera, el framework facilita al profesor la evaluación de sus alumnos.

  Las métricas de ingeniería de software y el análisis estructural de corrección operan sobre la estructura Abstract Syntax Trees (AST) que es representada a través de XML. De esta manera, la solución del estudiante y del modelo, son compara-

[1] Environment for Learning to Program. Queensland University of Technology. Disponible en: http://www.elp.fit.qut.edu.au/. Accesado en: Abril-2010

dos después de ser llevados a la estructura AST y haber pasado por un proceso de transformación en donde se extrae el algoritmo principal de ambos; también en este estado, la solución del estudiante es evaluada y enviada al profesor.

- Morimoto et al. (2006), presentan un sistema para la ayuda de los profesores de programación; en donde sus estudiantes formulen preguntas y cometan errores en las prácticas de programación de manera frecuente. El sistema extrapola las causas y los errores sintácticos de compilación, para ser presentado como resultado al profesor en tiempo real. De esta manera, el profesor tiene una herramienta que le permita monitorear los errores y el número de compilaciones por estudiante con el fin de orientarlos en un problema identificado. El sistema se basa para la detección de errores en un repositorio; donde permite comparar con compilaciones antiguas y errores comunes; esto ayudará al profesor a comprender la situación del estudiante y su tendencia al realizar preguntas y cometer errores. Las funciones principales son:

  - Presentación de la situación de los estudiantes al profesor: número de compilaciones, ocurrencia de error.
  - Extrapolación de los errores de causas y los errores de sintaxis.
  - Presentación de los errores extrapolados al profesor con el que puede ver los errores comunes.
  - Almacena el historial de las compilaciones que son captadas en tiempo de ejecución.

  La configuración del sistema presenta los siguientes módulos: presentación de la situación de los estudiantes, extrapolación de los errores, presentación de las tendencias de error, colección de los errores compilados, módulo de adquisición de errores de compilación; también posee dos base de datos que almacena el historial de los errores de compilación y contiene los errores comunes.

- De igual forma, Juedes (2005) proponen un software denominado Web-based grading Project (WBGP), éste software crea una página Web por cada estudiante que incluye un portafolio de sus proyectos con los resultados de las pruebas y calificaciones; además, genera reportes de los tipos y frecuencia de comentarios en el código y de las correcciones de los proyectos realizados.

3.2. Grupos de trabajos

La importancia de los grupos de trabajo se hace evidente en los proyectos de mayor tamaño donde es necesario cumplir, de manera colaborativa, un objetivo previsto; también, son útiles en la minimización del tiempo y esfuerzo requerido para llevar a cabo dicha tarea. Por lo tanto, un grupo de trabajo es considerado exitoso si cumple con los objetivos por el que fue formado; de esta manera, si dentro de los objetivos de un grupo está la realización de una tarea dentro de un tiempo determinado y la tarea es realizada fuera del tiempo límite, este se considera como un grupo no exitoso, debido a que se incumplió el objetivo.

3.2.1. Grupos vs. equipos

Dentro del área de trabajo colaborativo y cooperativo surgen los conceptos que dan significado al mismo: grupo y equipo, respectivamente. Existan muchas diferencias entre los mismo (ver apéndice A); pero, de forma general, ambos describen a un conjunto de personas que interaccionan entre sí para llevar a cabo una tarea o cumplir con un objetivo. Indistintamente de las características de dichos conjuntos de personas, se puede decir que el concepto de equipo es mayormente asociado a entidades empresariales y deportivas; y los grupos, a entidades académicas, públicas y civiles, sin importar que, en un momento determinado, un grupo tenga, parte o todas, las características de equipo y viceversa, en las entidades mencionadas. Por lo tanto, para efecto de esta investigación, se utilizará la palabra grupo para referirnos al conjunto de estudiantes que trabajan colaborativamente o cooperativamente en un aula de clase.

3.2.2. Grupos de trabajo como método didáctico

Basado en Lucero (2004), podemos decir que el aprendizaje es un proceso individual que puede fortalecerse a través de actividades de colaboración, en la que el individuo desarrolle habilidades personales y de grupo.

“Los constructivistas que apoyan la teoría dialética de Vygotsky del aprendizaje y el desarrollo, opinan que el trato social es importante para el aprendizaje porque las fun-

ciones mentales superiores (como el razonamiento, la comprensión y el pensamiento crítico) se originan en las relaciones sociales y luego son internalizadas por los individuos. Los estudiantes pueden realizar tareas mentales con apoyo social antes de que puedan hacerlas por sí solos; así, el aprendizaje cooperativo les proporciona el apoyo social y el andamiaje que necesita para avanzar en su aprendizaje." [2]

De esta manera, los grupos de trabajo en el aula de clases ayuda a los estudiantes a obtener mayor conocimiento, esto se debe a que los alumnos se convierten en tutor del otro, generando conocimientos en un tiempo menor y cada miembro del grupo aporta sus propios conocimientos que son complemento del otro. El éxito de los grupos de trabajo dependerá de un conjunto de factores que influyen directa e indirectamente. De esta manera, cuando en un grupo se logra desarrollar habilidades mixtas (aprendizaje y desarrollo personal y social), donde cada miembro del grupo es responsable tanto de su aprendizaje como del de los restantes miembros del grupo; entonces se dice que se ha alcanzado el éxito al utilizar los grupos de trabajo como método didáctico, en la que se dá discusiones entre los estudiantes al momento de explorar conceptos deseados o posibles soluciones, dando paso a las combinaciones de situaciones e interacciones sociales que contribuyen al aprendizaje significativo personal en el grupo; y así, se logrará resultados palpables a través del nivel deseado de conocimiento y rendimiento, en donde se neutralizan las dificultades y frustraciones producidas.

No necesariamente se obtiene aprendizaje colaborativo con la consecución de un objetivo, sin embargo puede ser un indicativo a evaluar; entre más un grupo de trabajo tenga las características de equipo, mayor será el aprendizaje de los estudiantes; estas características se pueden observar en el Apéndice A.

3.2.3. Grupos de trabajo para la enseñanza de la programación

Desarrollando un aprendizaje en ambientes colaborativos a través de la consecución de habilidades mixtas, se puede aprovechar la formación de grupos, como método didáctico en la enseñanza de la programación. Los estudiantes se forman en grupo para resolver alguna asignación de programación, en donde se discute sobre el tema

[2]Woolfok, A., "Psicología Educativa", en *De los grupos a la cooperación*, México, 1999, p.350

y una posible solución; luego se desarrolla, ya sea dividiendo los trabajos en subtrabajos para cada miembro o trabajando junto en un mismo modulo.

De esta manera, se crea indirectamente el efecto de que cada estudiante se convierte en el tutor del otro (como se dijo anteriormente); al poder consultar con sus compañeros diferentes inquietudes o bien, observar en conjunto el trabajo desarrollado por el resto. Existen diferentes tipos de grupos que pueden ser formados dentro del aula de clase; que, acompañado de técnicas didácticas, estrategias cognitivas o metodologías completas; pueden hacer mucho más efectiva la utilización de dichos grupos. La obtención del éxito de los grupos para la enseñanza de la programación, no es una tarea trivial; por lo que, en algunas ocasiones no es aprovechado como método didáctico, ya que existen diferentes factores que afecta el desempeño de la misma y por lo tanto su éxito. Se han empleado esfuerzos a través de investigaciones para aplicar, de manera efectiva, la formación de grupos dentro del aula de clase, especialmente en el área de la programación; sin embargo, dentro de este entorno existen factores propios que entorpecen o ayudan en este proceso; el nivel de colaboración o cooperación es un factor que influye directamente en la culminación y buen desarrollo de un trabajo asignado por el profesor; por otro lado, éste es afectado por una diversidad de factores, que en su mayoría, están asociados al ser humano; esta situación lo convierte en un problema complejo.

Cubranic y Storey (2005), presentan diferentes estrategias cognitivas aplicadas a los tipos de grupos; en donde se obtiene que su efectividad varia con respecto a la estrategia aplicada por tipo de grupo. Debido a que la utilización de grupos de trabajos es uno de los métodos didácticos más utilizados para la enseñanza de la programación; se busca constantemente la optimización de su uso. La investigación de de Faria et al. (2006), proponen la formación de grupos de estudiantes para el aprendizaje colaborativo de la programación; la formación de estos grupos está basado en los estilos de programación. En este análisis, se utilizo una herramienta llamada *Program Quality Assessment (PQA-C)* que determina un porcentaje o valoración en base a un conjunto de métricas, en donde los estudiantes de mayor puntaje forman grupo y los de menores o intermedio los demás grupos. Las métricas con la que es basado el estudio, atienden a las capacidad de los estudiantes que son mencionadas a continuación:

- Longitud del nombre de las variables.
- Porcentaje de variables definidas como constante.

- Longitud de los módulos (funciones y procedimientos).
- Número de módulos
- Porcentaje de indentación, líneas de comentario y espacios en blanco.

Tales métricas indican que una puntuación mayor perteneciente a un estudiante, posee mayores capacidades que otro con una puntuación menor; este estudio plantea que la conformación de los grupos de estudiantes con las mismas capacidades, hacen que su aprendizaje y producción sea más efectiva, ya que genera colaborativamente más conocimientos que si fuera con capacidades diferentes.

Por otro lado, la formación de grupos en la clase, no solo estimula el aprendizaje de la programación, sino el desarrollo de aptitudes para la colaboración, comunicación, relación intrapersonal y otros; ya que, se ejercita estas aptitudes por el nivel de requerimiento de los mismos, al desarrollar este tipo de trabajo en grupo. El resultado de las investigaciones de Castro et al. (2008), presentan un *Progression Learning Scheme*; este es un esquema de progreso para la enseñanza de la programación, a través de ciertos tipos de asignación y su forma de desarrollar; yendo de lo individual a lo grupal. Este esquema está basado en estudios realizados con estudiantes del curso de programación de la Universidad Federal de Amazonas. Presenta 6 fases que va del trabajo individual al grupal, cada fase está delimitada por actividades y asignaciones a lo largo del curso. La fase 1 de preparación los estudiantes individualmente resuelven problemas sencillos en los laboratorios y clarifican en la metodología; la fase 2 consiste en una solución individual de un problema determinado que debe ser tomado en cuenta posteriormente; en la fase 3 los grupos de trabajo empiezan decidiendo la mejor solución construida individualmente en la fase 2; la fase 4 se trabajará de manera grupal en donde el profesor define las tareas y el grupo define los actores para cada actividad; en la fase 5, cada grupo se hace responsable de la definición de las tareas para un problema específico; y finalmente en la fase 6, se desarrolla tareas en el que la colaboración y trabajo grupal se asemeja a una situación de trabajo real. De esta manera, se propone un modelo de seguimiento basado en la formación de grupos en la que se le asigna comportamientos específicos dependiendo del estado de progreso de los estudiantes.

3.2.4. Tipos de grupos para la enseñanza de la programación

Existen distintas técnicas que pueden implementar la utilización de los grupos de trabajo para la enseñanza de la programación; estas técnicas se resumen en los diferentes tipos de grupos que pueden ser formados y que son descritos a continuación, basados en Chong et al. (2010); de Faria et al. (2006); Scott et al. (1994).

a)- ***Grupo homogéneo basado en estilos de programación (EP):*** Conformado por estudiantes que presentan iguales estilos de programación. Este tipo de grupo permitirá una mejor integración y adaptación en las tareas asignadas, al compartir los estudiantes, las mismas preferencias y estilos de programar. Estos estilos de programación tienen que ver con la preferencia de herramientas y métodos al programar.

b)- ***Grupo heterogéneo basado en estilos de programación (NEP):*** Está conformado por estudiantes con diferentes estilos de programación. Puede ayudar en el reforzamiento de la habilidades de colaboración entre los estudiantes; ya que al no compartirse las mismas preferencias los estudiantes se ven obligados a hacer uso de sus facultades de comunicación y tolerancia.

c)- ***Grupo heterogéneo basado en deficiencias (ND):*** Conformado por estudiantes con diferentes deficiencias, puede ser útil para compartir conocimientos en un trabajo específico; en donde cada miembro será el complemento del otro en la solución de un problema, ya que la debilidad de uno es la fortaleza del otro.

d)- ***Grupo homogéneo basado en capacidades (CP):*** Para este trabajo, las capacidades son definidas como habilidades cognitivas que permiten aplicar los conocimientos adquiridos en el desarrollo de un problema de programación determinado. La formación de este tipo de grupo involucra formar estudiantes con similares capacidades cognitivas; esto ayudará a mejorar el rendimiento de los estudiantes dentro del grupo, ya que se elimina el problema de que uno de los integrantes tiene capacidad inferior representando una carga para el otro.

e)- ***Grupo heterogéneo basado en capacidades (NCP):*** Este tipo de grupo se conforma con estudiantes de diferentes capacidades. Esta formación puede ser utilizada con la formación de grupo Par (P), ya que ayudará a nivelar los conocimientos de aquellos estudiantes que tengan capacidades inferiores, en la que se fuerza a

obtenerlos al contar con un compañero con capacidad superior trabajando en este tipo de grupo (P).

f)- ***Grupo basado en estudiantes tutores (T):*** Conformado por estudiantes en donde uno o más hacen el papel de tutor; puede ser ventajosa su utilización debido a que los estudiantes tutores brindan una mejor ayuda a sus compañeros. Los estudiantes reciben de sus tutores un esclarecimiento de sus dudas y explicaciones personalizadas o más centradas de las que un profesor podría dar. Son útiles en aquellos salones de clase donde la mayoría de los estudiantes poseen pocas o menores capacidades con relación al resto del grupo, permitiéndoles que puedan ir obteniendo poco a poco mayor conocimientos del tutor en un ambiente colaborativo.

g)- ***Grupo de par (P):*** El trabajo en par utilizando una misma computadora comprende a dos estudiantes sentados frente a la misma, en donde uno toma el papel de *Driver* y el otro de *Navigator*; ambos pueden dialogar en una posible solución, pero el *Driver* es aquel encargado de escribir el código en la computadora y el *Navigator* de revisar y monitorear dicho código ya escrito con el objetivo de encontrar algún tipo de error. Por lo que se logra un avance significativo en el proceso de enseñanza-aprendizaje. Este tipo de grupo puede ser útil en la generación de nuevos conocimientos en un tiempo relativamente menor. Su optima utilización dependerá de que ambos estudiantes posean igual estilo de programación e iguales capacidades.

h)- ***Grupo basados en características personales (CP):*** Estos grupos se forma tomando en cuenta la personalidad de los estudiantes, esto se realiza a través de evaluaciones que pueden ser cuestionarios o simple entrevistas de los profesores. Es muy poco utilizado debido a las dificultades que se tiene para la clasificación, ya que se necesita conocimientos psicológicos para el mismo. Tiene mucha utilidad cuando se quiere evitar conflictos personales y que los estudiantes no pierdan el tiempo en conversaciones personales y juegos de amigos.

i)- ***Grupo basado en la elección de los estudiantes (EE):*** Son aquellos grupos conformados por estudiantes que han decidido formar parte del mismo y no ha sido impuesto por un profesor; generalmente los estudiantes buscan los compañeros con los que tienen mejor relación o los que tienen capacidades superiores o similares que les permita asegurar el éxito de la asignación. Tiene ventajas debido

a que se facilita las interacciones grupales y la coordinación de tareas debido a las buenas relaciones existentes. Sin embargo, la formación de grupos de este tipo trae como consecuencia un desbalance en la generación de conocimientos ya que generalmente los grupos quedan conformados de manera que los de mayor capacidad permanezcan en uno y los de menor quedan rezagados al formar sus propios grupos.

j)- ***Grupo con integrantes aleatorio (AP):*** Estos grupos están conformados por estudiantes que han sido generado de manera aleatoria, sin tomar en cuenta ningún tipo de criterio. Su principal ventaja es que los estudiantes quedan completamente distribuido de tal manera que ayuda a desarrollar en ello las habilidades necesarias para trabajar en grupo; sin embargo, la formación de este tipo grupo puede ocasionar que los estudiantes se encuentren distribuidos de manera desigual en cuanto a sus capacidades y deficiencias; esto provoca que ciertos grupos avancen en sus actividades y otros se vean entorpecidos por no existir un balance general en dicha formación.

De igual forma, se puede formar grupos tomando en cuanta dos o más de los tipos de formaciones ya mencionadas, por ejemplo: grupo par basado en estilo de programación semejantes. Muchas veces son considerados como un tipo de formación Mixto (Scott et al., 1994).

3.2.5. Competencias para el trabajo grupal

Como se ha mencionado el trabajo en grupo para la enseñanza de la programación, a parte de requerir ciertas habilidades cognitivas, requiere de otras habilidades pertenecientes al desarrollo del trabajo grupal. Estas habilidades se describen como un conjunto de aptitudes y valores que permiten facilitar la interacción de un estudiante con los demás; desarrollando así un aprendizaje mucho más robusto dentro del ambiente colaborativo del trabajo en grupo.

Se dice que un estudiante posee aptitud para el trabajo en grupo cuando puede relacionarse de manera efectiva con sus compañeros, en donde el sentido de compañerismo, respeto, tolerancia, compromiso grupal y valoración del trabajo de los

demás, predominen en dichas relaciones. Esto da como resultado una mayor comunicación y colaboración que permita obtener los beneficios del trabajo en grupo; beneficios que se resumen en la consecución de los objetivos de rendimiento y aprendizaje del contenido por parte de los estudiantes.

Basado en lo anterior, el trabajo grupal sería un fracaso si los estudiantes no poseyeran dichas aptitudes o valores; por lo que es necesario desarrollarlos en ellos. Esto se logra a través de actividades que permitan hacer conciencia y despierten la importancia en el mismo; ya que uno de los problemas es que los estudiantes no valoran ésta actividad debido que se creen auto-suficiente en la generación de conocimientos y convivir con otros en estos trabajos representa una pérdida de tiempo. Por otro lado, ciertos estudiantes poseen sentimientos de menosprecio dentro del grupo por lo que sienten que su trabajo es innecesario; empeorado por las actitudes erróneas de estudiantes que no aceptan opiniones de otros. También puede ocurrir que el salón se encuentre dividido debido a la existencia de ciertas diferencias, dando como resultado que un grupo de estudiantes no conviven con otros ya sea por problemas personales o por antipatía. Todo esto empeora el desarrollo de éstas aptitudes y por consiguiente el fracaso de los grupos.

Actividades como charlas de motivación, en la que se aclaren dudas y se explique las bondades de trabajar en grupo; por otro lado, presentar asignaciones en la que se conformen grupos y se evalúe la efectividad de los mismos, además de otras asignaciones en la que los estudiantes presenten una sola solución dentro del salón de clases; ésta actividad requerirá que el profesor tome el papel de líder y asigne subtareas grupales, aparte de forzar en el salón la discusión de posibles soluciones y despertar en ello una identidad de pertenencia al proyecto. De igual forma, éstas aptitudes pueden ser desarrollada mientras se le asigna trabajos grupales y a la medida que ellos desarrollen dichas habilidades generarán mayores conocimientos al ser más efectivo dentro de los grupos de trabajo.

3.2.6. Beneficios de los grupos de trabajos en la enseñanza de la programación

La utilización de grupos de trabajos para la enseñanza de la programación puede traer grandes beneficios. El principal impacto recae en el aprendizaje significativo del

estudiante, en donde éste gana experiencias a través de la interacción con sus compañeros, intercambiando conocimientos y recibiendo la influencia positiva del grupo en lo que se refiera al material de estudio de programación. Pero no solo trae beneficios en el proceso de enseñanza-aprendizaje del estudiantes, sino que ayuda a desarrollar aptitudes positivas que les permiten ser mejor como persona; por otro lado, trae beneficio a los profesores en cuanto a la obtención de sus objetivos pedagógicos de manera más efectiva.

A continuación se presenta una lista de los beneficios que involucra el utilizar grupos de trabajos para la enseñanza de la programación:

- El estudiante adquiere en un tiempo menor, mayor experiencia y conocimiento acerca de los temas específicos de la programación.
- Creación de un modelo o esquema mental correcto de la programación al entender su filosofía, en donde se mejora significativamente los flujos de trabajos por la enseñanza brindada de un estudiante a cada otro.
- Los estudiantes pueden profundizar mejor en temas específicos de su interés, en la programación.
- Se desarrollan proyectos de mayor envergadura, ya sea por ser más complejos o extensos.
- Se obtiene mayor eficiencia en la corrección de errores, ya que se tiene varias personas buscando una solución; como se dá en el caso de utilizar programación por par en un solo computador, en donde el *Navigator* u otros se dedica únicamente a evitar y corregir errores.
- Reforzamiento de los conocimientos al momento en que un estudiante necesite explicar un tema específico a otro miembro del grupo.
- Se desarrolla otras aptitudes positivas.
- Mayor seguridad en sí mismo y satisfacción del trabajo realizado, esto permitirá un aumento en la autoestima del estudiante.
- Los estudiante desarrollan la capacidad de reponerse fácilmente a las críticas personales y retroalimentación de sus trabajos; desarrollando una actitud de aceptación frente a dicha situación.

- Se desarrolla aptitudes para la organización, asignación de tareas y otras aptitudes relacionadas con el liderazgo; en aquellos estudiantes que recae el papel de líder o jefe de grupo.
- Entre más experiencia de trabajo en grupo, los estudiantes se encuentran más aptos para desenvolverse en el mundo laboral.

3.2.7. Problemas en la enseñanza de la programación utilizando grupos de trabajos

Los beneficios expuestos con anterioridad, no siempre se hacen evidentes al utilizar los grupos de trabajos, ya que existen problemas y dificultades que afectan notablemente el proceso de enseñanza-aprendizaje de la programación, como ya fue explicado. Se mencionan los problemas más importantes:

- Los estudiantes no trabajan bien en grupo, es decir, no están acostumbrados a trabajar colaborativamente; dando como resultado la falta de comunicación, prefiriendo realizar su propio trabajo individual.
- Si dentro del grupo existe una definición errónea de un tema específico, este puede ser aprendido por todo los miembros.
- Los estudiantes no denuncian irregularidades de trabajo al profesor por diversas razones.
- Típicos comportamientos en el que poco estudiantes hacen todo el trabajo, algunos hacen lo suficiente como para sobrevivir, y otros no hacen nada.
- Los estudiantes pueden tener diferentes problemas personales, familiares o sociales que le afectan en el desempeño del grupo.
- Los miembros pueden tener personalidades incompatibles dentro de los grupos de trabajo; ocasionando problemas de comunicación o coordinación.
- Se puede crear fricción y diferencias dentro del grupo; empeorado por las diferencias de personalidades.

- Diferencias de culturas, etnias y genero; que provoca dificultad en la comprensión del estado de ánimo de la persona; por lo que puede ocasionar distracción en el proceso de aprendizaje en grupo.

- No se posee suficiente conocimientos para desarrollar las tareas asignadas.

- Mala distribución de los grupos, en la que no se toma en cuenta aspectos especiales como capacidades, dificultades, personalidad de los estudiantes.

- La asignación propuesta por el profesor puede no ser apta para el trabajo grupal o es demasiado pesado para la experiencia y cantidad de miembros.

- Los miembros no se conocen bien, ocasionando un atraso en las actividades.

- Se puede desarrollar un ambiente de rivalidad entre los miembros o entre los demás grupos dando paso al egoísmo y al tratar de ser el mejor, descuidando aquellos detalles que permiten que los miembros perciban nuevos conocimientos.

Todos estos problemas causan que los estudiantes tengan mayores dificultades, creando en ellos un sentimiento de frustración, esto da como resultado una disminución en el proceso de aprendizaje colaborativo, en donde se pierde la capacidad de coordinación, de acordar posibles soluciones y de cumplir con las responsabilidades, entre otras.

3.3. Comprensión de programas

Basadas en Taherkhani et al. (2008), el enfoque de la comprensión de programas puede ser clasificado en tres categorías: comprensión de funcionalidades, análisis de los estilos de programación, reconocimiento y clasificación de algoritmos; descritos en la siguientes secciones.

3.3.1. Comprensión de funcionalidades

Esta se centra en la necesidad de conocer cuál es la función de un programa a través de un análisis dinámico; donde se le ingresan las entradas y se analiza las salidas; esta tarea puede ser llevada a cabo con análisis estático, pero resulta una tarea difícil; también pueden ser comparadas con códigos o programas modelos en la que se determina si un programa cumple con los requerimientos propuestos desde el principio.

3.3.2. Análisis de los estilos de programación

Según lo investigado por Cox y Fisher (2009), la diferencia de estilos de programación utilizados entre los programadores va a depender de ciertos factores como la personalidad del programador, nivel de educación, ambiente laboral en la organización, ambiente socio-económico y político; todo esto va influenciando la elección de los estilos de programación. Algunos aspectos o elementos, en donde los estilos de programación se hacen evidentes son descritos a continuación:

- **Nombres de variables, controles y procedimientos:** el estudiante puede elegir cualquier nombre a utilizar, este puede representar o describir la funcionalidad de la herramienta o bien elegir, para su programa, un conjunto de nombres que describa la secuencia y número de variables utilizadas, que le permita llevar un inventario de las mismas. En este aspecto se incluye también, el nivel de descripción de la representación de los nombres utilizados; es decir, qué tan largos son.
- **Inicialización de variables:** este aspecto, en ocasiones puede ser opcional; sin embargo, para algunos programadores inicializar todas las variables al momento de su declaración o utilización es de vital importancia en la eliminación de registros basura de memoria.
- **Utilización de constantes:** la utilización de constantes puede ayudar a que, a la hora de realizar modificaciones de valores, se haga solamente en las constantes; sin embargo, muchos programadores no prefieren utilizar constantes o simplemente ignoran este estilo.

- **Indentación y espacios en el código:** se emplea para describir líneas de códigos subordinadas; como por ejemplo, al declarar un ciclo FOR anidado dentro de otro la utilización de espacios y sangrías puede hacer que se tenga múltiples representaciones de la misma; que la utilización de una dependerá de la elección del programador.

- **Documentación de las líneas de código:** los programadores para el mejor entendimiento de ellos u otros programadores utilizan la documentación, en donde describen funcionalidades o comportamientos de líneas de código. El nivel de documentación esta bajo la consideración del programador y el grado de complejidad del programa.

- **Implementación de ciclos y estructuras de decisión:** algunos algoritmos admiten la utilización de varias modalidades de ciclos, en el caso del FOR y del WHILE las funcionalidades que brindan pueden ser iguales; sin embargo, algunos programadores prefieren la utilización de ciclos WHILE únicamente. De igual forma, cuando se tiene muchas preguntas de decisión algunos programadores optan por la utilización de SWITCH-CASE para representarlo, en el caso de otros, utilizan solo el IF-ELSE.

- **Algoritmos inductivos y recursivos:** el estudiante que domine los conceptos de los procesos inductivos y recursivo en un programa; puede preferir uno de los dos para sus aplicaciones.

- **Procedimientos y funciones:** es decisión del estudiante el nivel de acoplamiento y el incluir una o varios procedimiento dentro de una misma función; la mejor implementación de las mismas dependerá del nivel de conocimiento que posea el programador.

- **Creación de clases:** el estudiante puede elegir si crea archivos por separado para cada clase del programa o agrupar las clases en un mismo archivo; se puede determinar el orden de creación de las clases, si es creada la clase *main* primero o cualquier otra para un caso especifico, esto será siempre y cuando se indique el orden de creación.

- **Utilización de herencia y polimorfismo:** en POO la utilización de herencia y polimorfismo puede ayudar a estructurar mejor el programa; sin embargo, en algunos casos su implementación puede ser opcional. De esta manera, los niveles

de uso de estas propiedades, para un caso específico, pueden representar un estilo de programación.

3.3.3. Reconocimiento y clasificación de algoritmos

Este problema de comprensión de programa implica la necesidad de conocer de antemano la funcionalidad del programa; existen algoritmos que pueden ser más óptimos en un caso que en el otro, por lo que la elección de un algoritmo adecuado (por ejemplo, algoritmos de ordenación *quicksort*, *mergesort*, *insertion*, etc.) dependerá de los requerimientos del programa a crear y el conocimiento de complejidad de los estudiantes.

3.4. Analizadores estáticos de código fuente

El análisis estático de código fuente es una técnica utilizada en la comprensión de programas para la evaluación y análisis de programas sin ser ejecutados. Ésta técnica extrae información relevante acerca de la estructura del programa, estilos de programación, errores semánticos y otros.

3.4.1. Enfoques de los analizadores estáticos

Según Taherkhani et al. (2008), el análisis estático tiene varios enfoques y se dan de diferentes maneras:

- **Enfoque basado en conocimiento** Este enfoque esta basado en un conocimiento base que se encuentran predefinido en planes almacenados. Para entender el programa, el código fuente es comparado con el plan o diseño y si este coincide se puede deducir que realiza la función predefinida.
- **Enfoque de evaluación por similitud** Se basa en el análisis de las estructura de los programas y estilos de programación en vez del reconocimiento de funcionalidades. Este enfoque puede ser dividido en dos subcategorías: enfoque de

evaluación de atributos, en donde se contabiliza los atributos o métricas en un programas y al ser comparados se puede obtener la similitud de los programas; y el enfoque basado en estructura, que compara las estructuras del código.

- **Enfoque de ingeniería inversa** Consiste en analizar el código fuente y extraer información que será expresada de manera documentada con la finalidad de entender su funcionalidad o la estructura del programa, es útil para el entendimiento de programas complejos y su comparación subjetiva.

3.4.2. Herramientas y aplicaciones de los analizadores estáticos

El análisis estático de código fuente tiene muchas aplicaciones dentro de la ingeniería de software, pero su principal aplicación es la de detección de bugs o errores de programación; pueden mencionarse algunas herramientas como: PMD [3], es un analizador estático gratuito de Java orientado a programadores; contiene un conjunto de reglas para la detección de posibles bugs, códigos sin usar, expresiones repetidas y otros; de igual forma, FindBugs [4] tiene la misma orientación que el anterior en la detección de errores de programación. CheckStyle [5] es un analizador estático enfocado al estilo y formato del código.

Otras aplicaciones pueden ser: extracción de vistas, gráficas y patrones del código fuente de programas orientados a objetos, detección de algoritmos a través de métricas, evaluación de los estilos de programación; sin embargo, también es utilizado como técnica dentro de algunas herramientas para la enseñanza-aprendizaje de la programación, tal es el caso de la propuesta de Truong et al. (2009) que ya fue descrita en la sección 3.1.3; Mengel y Yerramilli (1999), proponen una evaluación de la calidad de los programas a través de ciertas métricas de ingeniería de software; la investigación de Taherkhani et al. (2008), proponen una herramienta que identifica algoritmos utilizados por los estudiantes y evalúa la capacidad del mismo.

[3] PMD, disponible en: http://pmd.sourceforge.net/
[4] FindBugs - Find Bugs in Java Programs, disponible en: http://findbugs.sourceforge.net/
[5] CkeckStyle, disponible en: http://checkstyle.sourceforge.net/

3.5. Ontología

Este término surge como rama de la filosofía, según la RAE (2001), se define como parte de la metafísica que trata del ser en general y de sus propiedades trascendentales; es decir, que se encarga de la descripción de una entidad tomando en cuenta sus propiedades y la interacción de los mismos; sin embargo, este concepto no fue utilizado en la informática, sino hasta la década de los 80 por la Inteligencia Artificial. A partir de aquí se a empleado en la representación de conocimientos a través de reglas lógicas y expresiones formales. Existen muchas definiciones de ontología en la informática, sin embargo la siguiente definición, a nuestra consideración, se ajusta más a la descripción de la aplicación de ontología en este trabajo. Basado en Noy y McGuinness (2005) se define como: “una descripción explicita y formal de conceptos en un dominio de discurso (clases (a veces llamadas conceptos)), propiedades de cada concepto describiendo varias características y atributos del concepto (slots (a veces llamados roles o propiedades)), y restricciones sobre los slots (facetas (algunas veces llamados restricciones de rol)). Una ontología junto con un conjunto de individuos de clases constituyen una base de conocimiento”.

3.5.1. Aplicaciones

Existen muchas aplicaciones en la que puede ser usada la ontología, sin embargo, Noy y McGuinness (2005) las cataloga en 5 grandes áreas en las que se hace evidente su utilización:

- Compartir el entendimiento común de la estructura de información entre personas y agentes de software.
- Permitir la reutilización de conocimiento de un dominio.
- Explicitar suposiciones de un dominio, que subyacen bajo una implementación, permite cambiar esas suposiciones fácilmente si el conocimiento del dominio cambia.
- La separación del conocimiento del dominio del conocimiento operacional.

- Analizar el conocimiento de un dominio.

De esta manera, las aplicaciones de la ontologías en la actualidad, son en su mayoría, en el mundo de la Web, en la que son utilizadas para la descripción de portales, mejorar búsquedas y descubrimientos, utilización de los servicios, gestión de la información en sitios, colecciones multimedia y en *Ubiquitous computing* tienen su aplicación; también es muy utilizado en el diseño de documentación, que luego será fácil la interpretación por diversos programas; creación de procedimientos y especificaciones de instalación (Heflin, 2010).

3.5.2. Lenguajes ontológicos

Los lenguajes ontológicos han evolucionado desde su aparición a principios de los 90, actualmente dichos lenguajes se apoyan en estándares orientados al intercambio por red, como HTML y XML; estos lenguajes son: RDF, RDF Schema, SHOE, XOL,OIL,DAML+OIL y OWL (Gómez-Perez y Fernández, 2004). Esta sección se enfatizará en el lenguaje ontológico OWL, ya que es el que será utilizado en el presente trabajo.

OWL es un lenguaje para la ontología Web que está diseñado para ser usado por aplicaciones que necesiten procesar contenidos de información, que pueda ser presentado con fácil entendimiento a los usuarios. OWL facilita a las maquinas la interpretación de los contenidos, está soportado por XML, XML Schema, RDF y RDF Schema que proporcionan vocabulario adicional que permite una mejor semántica (Heflin, 2010).

OWL está conformado por tres sublenguajes que van en aumento con respecto a la capacidad de expresión. Estas son, OWL Lite, OWL DL, OWL Full (McGuinness y van Harmelen, 2010). OWL Lite proporciona un conjunto de instrucciones capaces de describir clases y subclases; además, brinda la capacidad de expresar propiedades a los objetos y a los datos almacenados, que pueden relacionarse entre las clases y subclases existentes; por otro lado, OWL soporta la características de *minCardinalidad, maxCardinalidad y Cardinalidad*, en donde solo se puede especificar 0 ó 1. OWL DL y OWL Full presentan un nivel de expresión mayor, ya que acepta números mayores que 1 en las propiedades de cardinalidad; también, permite especificar las clases

que requieren estrictamente un valor ya sea de una instancia o dato. Puede especificar los miembros requeridos por clases y su respectivos valores estrictos. OWL DL se diferencia de OWL Full en que el primero no soporta la utilización de clases como caso de otra clase, mientras que el Full si. Por lo tanto, OWL Lite es un OWL DL y un OWL DL es un OWL Full, en el que OWL DL representa el de mayor restricción y el OWL Full el de mayor expresión y libertad sintáctica.

Capítulo 4

Arquitectura de software propuesta

La arquitectura de software toma en consideración las necesidades que se quieren satisfacer con el proyecto y las facilidades que brindan las tecnologías disponibles; de igual forma, las restricciones que pueden surgir.

En este capítulo se presenta la arquitectura de software de SOFORG, en la que se describe los componentes y las interacciones que existen entre los mismos; además de brindar los detalles de funcionamiento para las operaciones más importantes de la herramienta, se considerarán algunas vistas o modelos representados en diagramas y esquemas; de igual forma, se presentarán los algoritmos o métodos más importantes empleados para cumplir con los objetivos del mismo.

4.1. Generalidades de la herramienta

Basado en los objetivos propuestos de este trabajo, se define la herramienta SOFORG como un sistema para la formación de grupos de trabajo basado en ciertos parámetros e información extraída del código fuente de los estudiantes. Esta herramienta representa un apoyo en la labor docente, por lo que debe ser de uso exclusivo de los mismos; esto implica un software de tipo escritorio con base de datos central (Arquitectura Cliente-Servidor), en la que cada profesor alimente un mismo repositorio. Haciendo referencia a una arquitectura de tres capas (Presentación, Lógica de

Negocio, Datos), esta herramienta ejecuta en las máquinas clientes, la Presentación y la Lógica del negocio, dejando la capa Datos del lado del servidor; esto se debe a la complejidad de las operaciones que debe ejecutar la herramienta, ya que si se centraliza podría ocasionar la caída del servidor. Por otro lado, se utilizará lenguaje Java para la programación del mismo debido a la compatibilidad que existe con otras herramientas y librerías a utilizar que serán detallada en la siguiente sección.

De esta manera, la herramienta debe capturar en primera instancia, toda la información del estudiante a través del análisis estático del código fuente, este se realizará a través de un analizador sintáctico-léxico e interprete, que emitirá los resultados en forma de métricas para luego ser almacenada; parte de estas métricas es la evaluación del código que representa un problema o ejercicio asignado por el profesor, para esta evaluación se utilizará lenguajes ontológicos y una librería para su gestión. De esta manera, se construirá una aplicación basada en conocimiento utilizando ontologías, esto permitirá la evaluación del código fuente de manera más eficiente, en la que se tendrá una ontología base e instancias del mismo que inferirán conocimiento y así determinar dicha evaluación.

Otra parte del módulo de la herramienta, consiste en tomar la información de los parámetros almacenados (métricas de estilos de programación, evaluación) y generar los grupos por curso existente, haciendo referencia a un profesor; se proponen diferentes tipos de grupos a formar y dependerá de los parámetros que han sido extraídos del código, por lo que existirá una limitante en los tipos de grupos que pudieran ser formados. Por otro lado, como funcionalidad complementaría, se propone que SOFORG pueda realizar reportes de diferentes tipos con el objetivo de comparar y analizar el rendimiento de los estudiantes, métodos de enseñanza de los profesores y efectividad de los ejercicios aplicados.

4.2. Herramientas utilizadas

Para el desarrollo e implementación de SOFORG se considera la utilización de algunas tecnologías o herramientas. A continuación se presenta algunas de ellas.

4.2.1. JavaCC

JavaCC (Java Compiler Compiler) es un generador de analizadores sintácticos de código abierto para el lenguaje de programación Java. JavaCC genera un parser para una gramática presentada en notación BNF. A diferencia de Yacc, JavaCC genera analizadores descendentes (top-down), lo que lo limita a la clase de gramáticas LL(K) (en particular, la recursión desde izquierda no se puede usar).[1] A pesar de sus limitaciones frente a otros analizadores, JavaCC es aplicado en esta herramienta debido a que la generación del parser es en código Java, permitiendo ejecutar códigos en el mismo lenguaje ya que se obtiene un interprete de los mismos. La versión utilizada en este proyecto es la ver. 5.0.

4.2.2. Protégé

Protégé es una plataforma de código abierto que proporciona un conjunto de herramientas para la construcción de modelos de dominios y aplicaciones basadas en conocimientos utilizando ontologías. Permite la creación y edición de ontologías modelándolas gráficamente o a través de marcos, en la que se despliega propiedades que pueden ser editadas; por otro lado, las ontologías pueden ser exportadas en una variedad de formatos incluyendo RDF(S), OWL y XML Esquema.[2] En este proyecto, se utiliza la ver. 4.1 para el modelado y diseño de la ontologías necesarias en el módulo de evaluación del código fuente.

4.2.3. JENA

JENA es un framework de código abierto para la construcción de aplicaciones de Web Semántica. Este proporciona una colección de herramientas y librerías en Java para el desarrollo de aplicaciones de web semántica, datos enlazados, herramientas y servidores. Tiene soporte para la manipulación de lenguajes RDF, RDFS, OWL;

[1] Java.net. Java Compiler Compiler - The Java Parser Generator. Accesado: Mar-2011. Disponible en: `http://javacc.java.net/`

[2] Stanford Center for Biomedical Informatics Research. The Protege Ontology Editor and Knowledge Acquisiton System. Accesado: Mar-2011. Disponible en: `http://protege.stanford.edu/`

además de realizar consultas con SPARQL. Incluye un motor de inferencia basado en reglas lógicas, que se ejecuta sobre las ontologías creadas o importadas.[3] Una vez que se crea las ontologías en Protégé, utilizando OWL, se integran al resto del proyecto con la utilización de JENA ya que es accedido en forma de librerías dentro del mismo. La versión a utilizar es la ver. 2.7.0.

4.2.4. NetBeans

Es un ambiente de desarrollo integrado para el lenguaje Java, que permite crear aplicaciones Web y de escritorios. NetBeans es de código abierto y se encuentra auspiciado por la empresa Oracle. [4] Netbeans será utilizado en dicho proyecto para el desarrollo del mismo, debido a su amigable interfaz gráfica, cuya versión es ver. 7.0.1.

4.2.5. MySQL

Es un sistema de gestión de bases de datos relacional, multihilo y multiusuario, desarrollado como software libre, pero es incorporado como producto privativos para las empresas que desean comprarlo. [5]

4.3. Elementos de SOFORG

En la figura 4.1, se presenta un esquema de elementos que conforman la arquitectura de SOFORG, de igual forma se visualiza la interacción que existen entre los mismos. A continuación se describe cada elemento y su funcionalidad.

[3]The Apache Software Foundation. Jena - A Semantic Web Framework for Java. Accesado: Jul-2011. Disponible en: `http://incubator.apache.org/jena/`

[4]netbeans.org. Netbeans IDE. Accedido: Mar-2011. Disponible en: `http://www.netbeans.org`

[5]Oracle Corporation. MySQL. Accedido: Mar-2011. Disponible en: `http://www.mysql.com`

Figura 4.1: Esquema de elementos de SOFORG.

4.3.1. archivos.java

Comprende el código fuente que el estudiante desarrollo en lenguaje Java, la herramienta puede aceptar un conjunto de archivos de este tipo. Uno de ellos (el principal) debe contener en forma de comentario el ID del estudiante y el código de la prueba. Ejemplo: `/* @ID: 9-394-345 @codigo: E0003 */`. Si la asignación fue en grupo, se coloca una coma (”,”) y se escribe las demás ID.

4.3.2. Analizador Estático - C

Analiza y proporciona los estilos de programación del estudiante basado en el código fuente proporcionado. Recibe los archivos `.java` que son analizados en forma de cadena de caracteres y busca patrones que determinaran dichos estilos de programación, todo esto basado en las métricas que se describirá en el punto 4.6.1.

4.3.3. Analizador Estático - GLC

Este elemento tiene como entrada los archivos `.java`, en la que se analizará su código fuente; a diferencia del anterior, éste analiza con Gramática Independiente de Contexto, utilizando JavaCC, que es descrito en el punto 4.2.1. De esta manera, este elemento proporciona el resultado de: buenas prácticas y preferencias de estructuras (como se detallo con anterioridad); también, en caso de que existiese un modelo OWL del ejercicio del estudiante, éste extrae la información de los componentes del programa, en forma de instancias OWL y lo envía al *Analizador OWL*.

4.3.4. Repositorio General

Una base de datos en MySQL (descrito en el punto 4.2.5) que contiene los datos generales de los estudiantes, profesores, ejercicios y otros datos de importancia en la herramienta. Para el caso de los ejercicios, éstos almacenan información del nombre, descripción de la asignación y la referencia al archivo OWL; ésto, en caso de que existiese un modelo ontológico.

4.3.5. Analizador OWL

El *Analizador OWL* recibe la información de las instancias OWL (proporcionada por el *Analizador Estático -GLC*); luego, por referencia especificada en el código fuente, se carga el modelo ontológico base de las posibles soluciones y se analiza comparándolo con el programa del estudiante, todo esto utilizando las librerías proporcionadas por JENA descrito en el punto 4.2.3. Al final, se obtendrá el resultado de la evaluación del ejercicio desarrollado por el estudiante. Cabe destacar que no todos los ejercicios analizados tendrán su modelo ontológico para ser evaluados, por lo que el profesor debe incluir la evaluación de forma manual, si este fuese el caso; de esta manera, no siempre se hará uso de este elemento. Por otro lado, los ejercicios que tengan un modelo base, pero no la solución del estudiante, debieran ser agregados a la ontología base.

4.3.6. Repositorios de cada análisis

El resultado obtenido de cada análisis es almacenado en una base de datos relacional, se utiliza, de igual forma, MySQL (descrito en el punto 4.2.5). Dicho resultado es representado a través de métricas y hará referencia a uno o varios estudiante, al ejercicio desarrollado, conjuntamente con el profesor y curso.

4.3.7. Generador de grupos

Este elemento contiene un conjunto de clases que permiten la generación de grupos. Utiliza un algoritmo que verifica todas las posibles formaciones de los estudiantes en grupos, y así seleccionar los mejores grupos basados en las opciones de formación que el profesor seleccionó. Las opciones y los tipos de grupos que pueden ser formados, son descritos en la sección 4.8.

4.4. Clases y componentes

Como se explico con anterioridad, todas las operaciones de la capa de Lógica de Negocio están contenidas en una misma computadora, por lo que todas las clases propuestas para SOFORG operan juntas y se encuentran relacionadas en un mismo proyecto. En la figura 4.2, se presenta el primer diagrama de clases en la que se observa las relaciones de las clases propuestas. Se observa que se tiene una clase llamada *principal* que es la que inicia la ejecución del programa, esta clase utiliza *analisisResultForm* y *generadorGruporForm*; que proporcionan la función de ejecutar las operaciones del análisis estático y la generación de los grupos respectivamente. La clase *generadorGruporForm* contiene con conjunto de métodos y atributos como es observado, en la que accede a la información almacenada y, dependiendo de las opciones seleccionadas para la formación del grupo, se genera los grupos atendiendo a dichos parámetros. De igual forma, la clase *analisisResultForm* llama a la clase *Analizador_Ind*, esta clase es la encargada del análisis estático con GIC y utiliza la clase *evaluadorOWL* para brindar el resultado de la evaluación del código; dicha clase, utiliza las librerías de JENA y ejecuta las operaciones sobre las ontologías base. Cabe

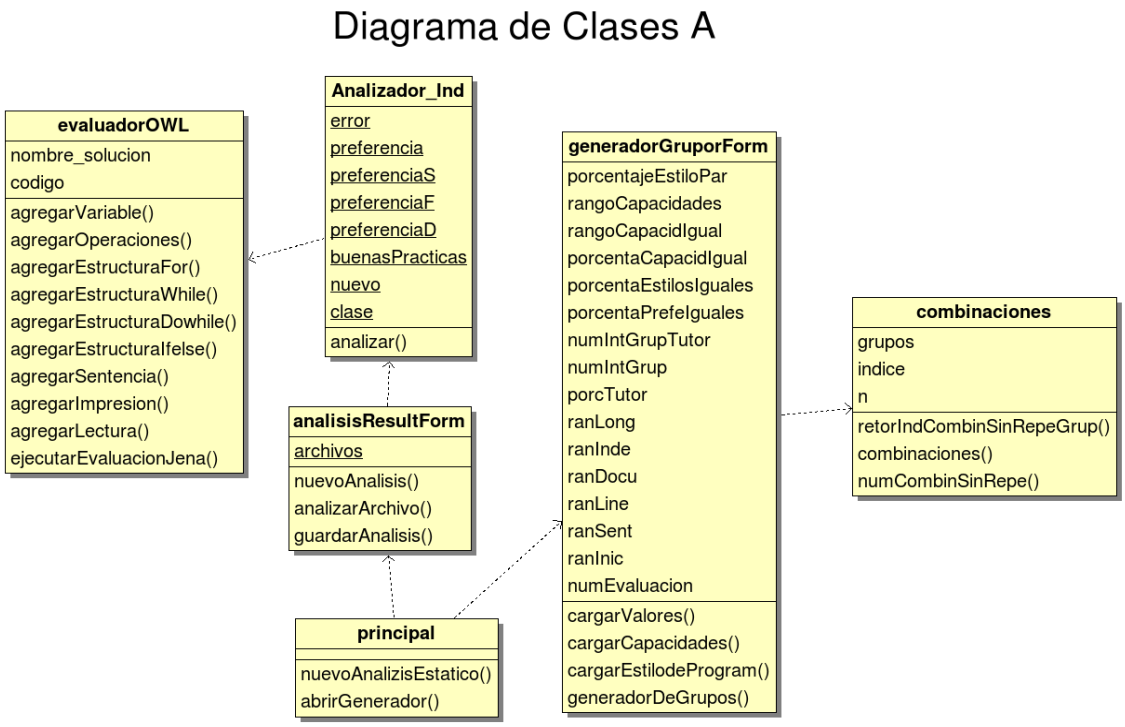


Figura 4.2: Diagrama de clases A.

destacar que la clase *Analizador_Ind* es la clase principal generada por JavaCC y representa el parser conjuntamente con 4 clases más; estas 4 clases no han sido incluidas en el diagrama debido a su poca relevancia.

La clase *estiloProgramacion* es instanciada por *analisisResultForm*, debido a que ésta, proporciona la operación de analizar estáticamente el código fuente, este análisis es realizado en forma de cadena de caracteres en la que devuelve en forma de métricas los resultados obtenidos; todo esto es observado en el diagrama de la figura 4.3. Este diagrama presenta las demás relaciones de la clase *Analizador_Ind* con otras; estas son la clase *preferenciaCodigo* y *instanciaClase*. La clase *preferenciaCodigo* brinda la funcionalidad de determinar las preferencias del estudiante para el código proporcionado, retornando el resultado a la clase *Analizador_Ind*; por otro lado, la clase *instanciaClase* determina las sugerencias de buenas prácticas en el código. El resto de las clases representan una estructura de datos para el almacenamiento de los componentes del código y así determinar de manera más sencilla, dichas sugerencias de buenas prácticas; esta estructura de datos es invocada y gestionada por la clase *instanciaClase*, en la que se puede observar atributos y métodos para agregar

clases, atributos, métodos, constructores y otras operaciones de búsqueda; existe una clase llamada *clase*, otra *atributo*, *constructor* y *método*; todas estas administran la información de dichos componentes; sin embargo, existen otras que administran las propiedades de los mismos y heredan funcionalidades de otras como se observa en la figura; estas clases son *entidad*, *entidadStaticFinal*, *entidadAbstract*.

En la figura 4.4, se observa el diagrama de componentes para las clases propuestas, en la que se aprecia mejor las relaciones y operaciones que se desean ejecutar; se observa que el proyecto esta dividido en 3 paquetes principales, el paquete *finalproyectmasterdegree*, *Analizador_ind* y *estPrefeBuenProgramacion*; existen otros paquetes que no se encuentran representando en dicho diagrama, debido a que representan operaciones menores o librerías de apoyo; sin embargo, se puede observar que las dos principales funcionalidades representadas como componente es el de analizador estático y el de generador de grupos; estos componentes requerirán las funciones de los demás componentes.

4.5. Modelo relacional de la base de datos

SOFORG implementa una base de datos relacional en la que almacena datos que fueron descritos en los puntos 4.3.4 y 4.3.6, donde ser describían los elementos de la herramienta. La figura 4.5, representa el modelo de entidad-relación de SOFORG, donde se observan tablas para el almacenaje de los datos de los profesores y estudiantes, además de almacenar otros datos como el resultado de los análisis, grupos, cursos y pruebas aplicadas a los estudiantes. Cabe destacar que la tabla *Estudiante*, tiene una relación directa de muchos a muchos con la tabla *Analisis*, en vez de estar relacionada con *Prueba*; esto se debe a que se mejora el proceso de búsqueda a la hora de extraer los datos de las métricas de los estudiantes. Se observa también, una tabla denominada *config*, que almacena los valores de las variables internas que pueden ser editadas por el profesor, esto se describe más adelante en la sección 4.9.

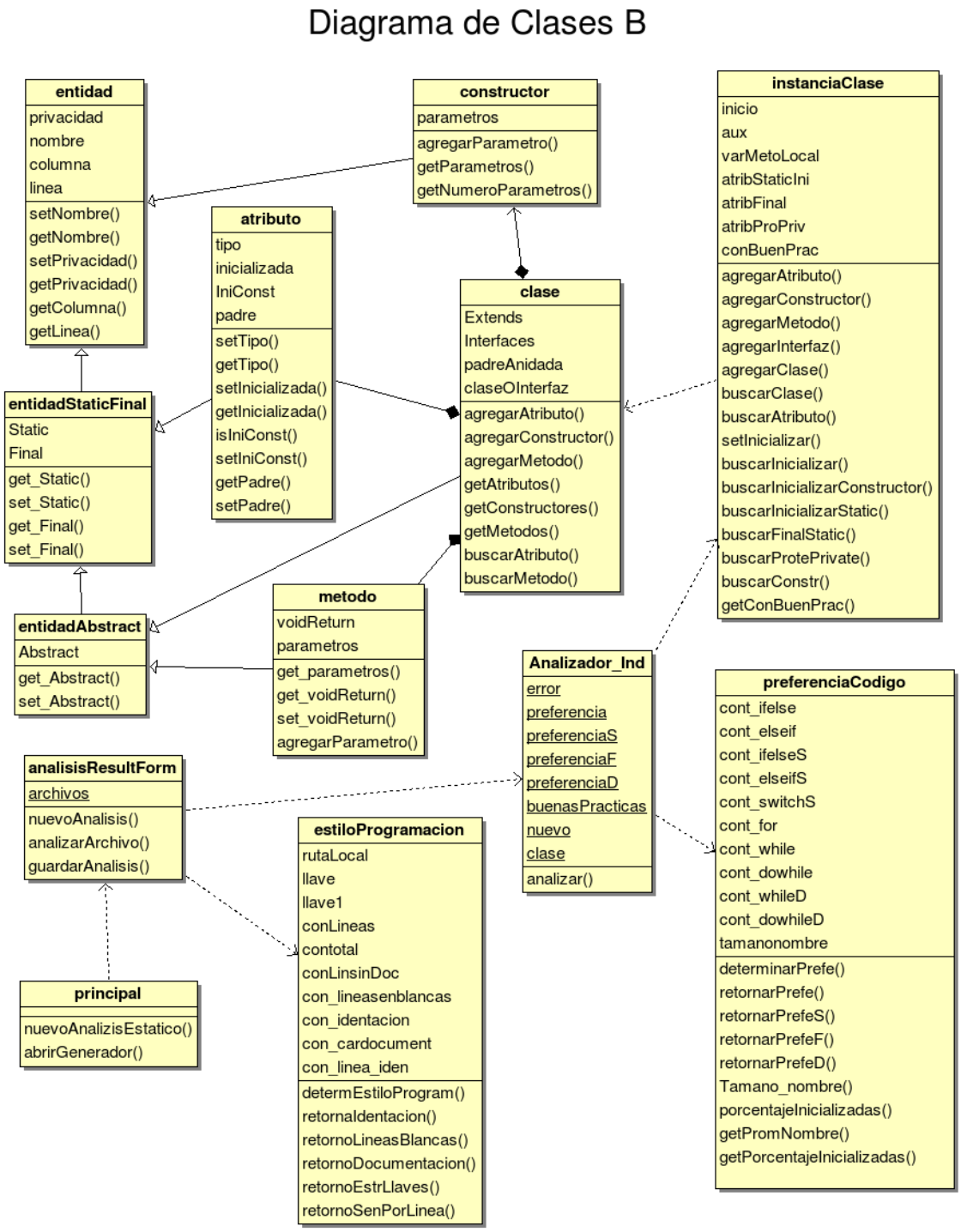


Figura 4.3: Diagrama de clases B.

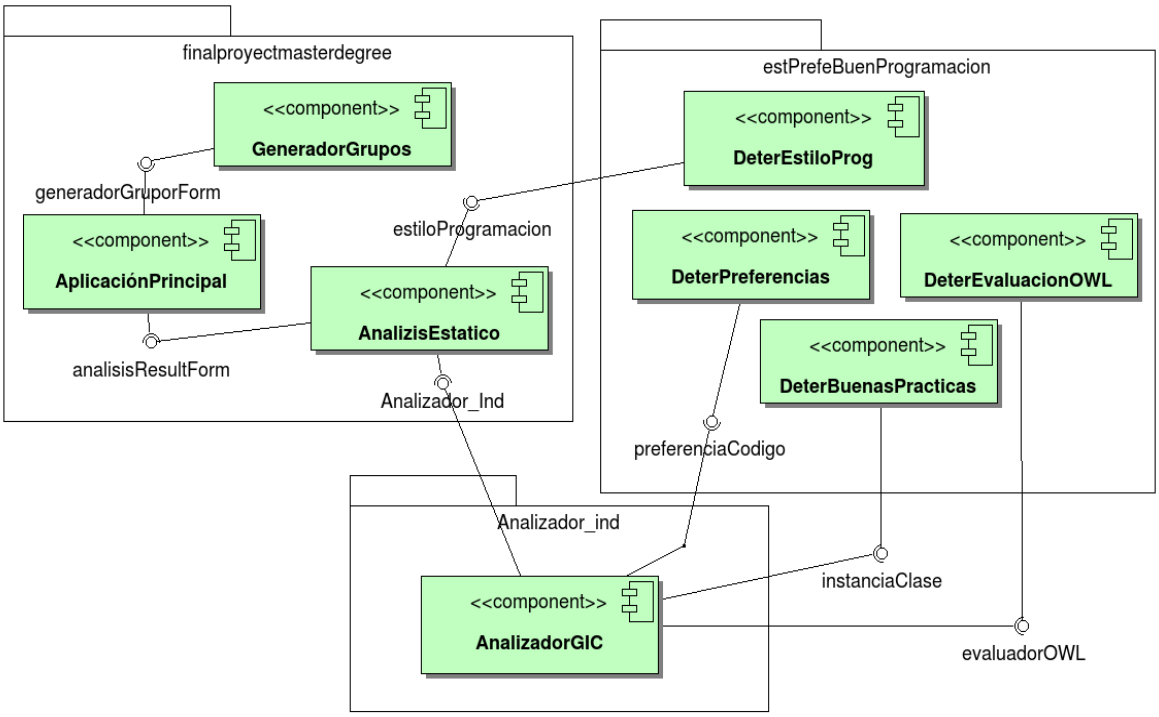


Figura 4.4: Diagrama de componentes de SOFORG.

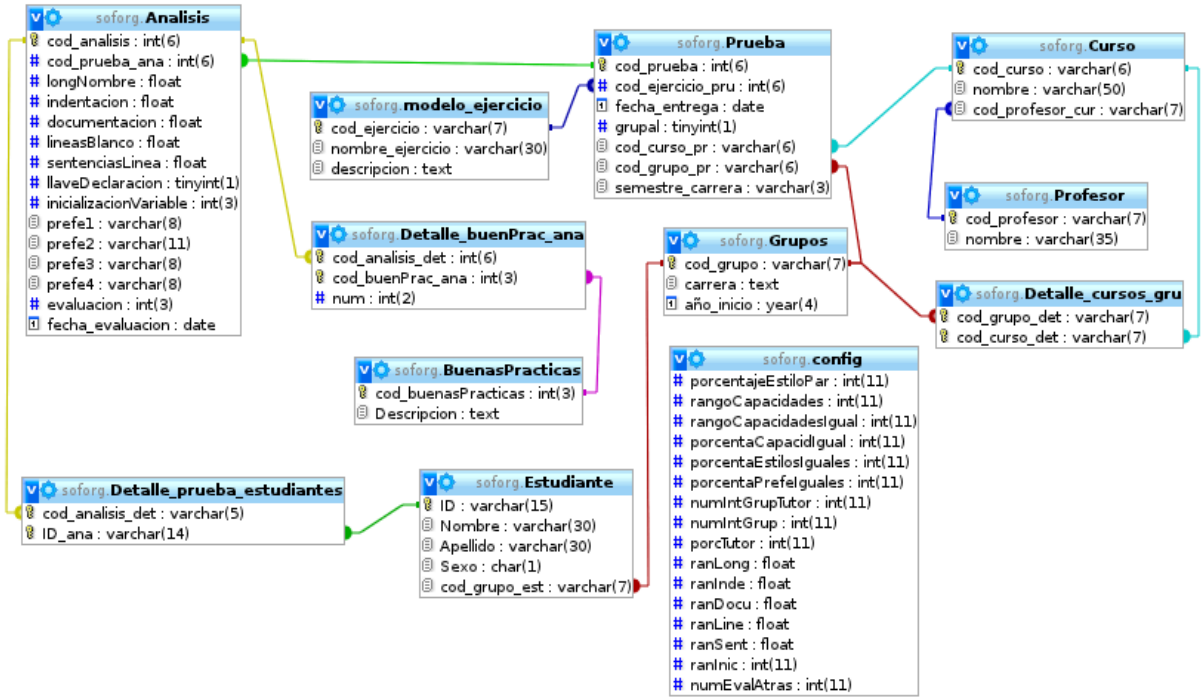


Figura 4.5: Modelo entidad-relación de la base de datos de SOFORG.

4.6. Propuestas de resultados del análisis estático

El resultado del análisis estático representa la información extraída del código fuente de los estudiantes, esta información es expresada a través de métricas individuales de cada aspecto a evaluar. Está dividida en 4 grupos: estilos de programación, preferencias de estructuras, buenas prácticas y una posible evaluación del resultado del estudiante; que serán descritos a continuación.

4.6.1. Estilos de programación

Los estilos de programación son independientes de la funcionalidad final del programa, por lo que representa la apariencia y formato que cada programador le da al código fuente. Existen muchos estilos de programación debido a que éste dependerá de la personalidad y los hábitos del programador Cox y Fisher (2009). Existen empresas de desarrollo de software que tienen estilos de programación preestablecidos en la que cada programador debe regirse; sin embargo, a los estudiantes de programación básica no se les exige ser regidos por un estilo específico (sí por buenas prácticas), ya que no existe un estándar de cuál es el mejor estilo de programación, y regirlos por uno, atentaría contra la comodidad en la utilización del lenguaje y por ende, en el proceso aprendizaje.

El análisis de los estilos de programación en SOFORG será útil en el conocimiento individual de la forma en que programa cada estudiante, esto permitirá al profesor la creación de grupos de trabajo de iguales estilos; ya que por estudios previos, se conoce que puede mejorar el rendimiento en dichos grupos de Faria et al. (2006). Para el análisis de los estilos de programación se utiliza el enfoque basado en conocimiento, descrito en secciones anteriores. Basados en Haahr (2010); Wheaton (2010), se mencionan los estilos de programación más importantes con sus respectivas métricas:

a)- ***Longitud de los identificadores (LI):*** toma en cuenta los nombres de variables, métodos, clases, interfaces y paquetes; en la que extrae un promedio general de la longitud de las mismas. Se propone la siguiente ecuación para ser medida:

$$PL = \frac{NC}{NTN} \tag{4.1}$$

dónde: *NC* - el número total de caracteres de todos los identificadores; *NTN* - el número total de identificadores encontrados en el código; *PL* - es el promedio de longitud de los identificadores.

b)- ***Indentación (I):*** es el espacio de sangría que existe al inicio de cada línea de código, es utilizado para mejorar la vista y lectura del mismo. La siguiente ecuación es propuesta para su medida:

$$PI = \frac{\sum_{l=1}^{L}\sum_{i=1}^{I}(NE_i)}{\sum_{l=1}^{L}(N_l \cdot 4)} \tag{4.2}$$

dónde: *L* - es el número total de líneas de código; *I* - es el número de espacios de indentación por cada línea de código l; NE_i - es el número de espacio de indentación total del código fuente; $N_l \cdot 4$ - es el nivel de anidamiento de cada línea de código (0,1..) multiplicado por 4; *PI* - es la relación que existe entre el NE_i y $N_l \cdot 4$. Esta última sumatoria se debe a que se toma como referencia una indentación de 4 espacio de más, por cada nivel de anidamiento, ejemplo: línea 2 en nivel 0 = 0 espacio de indentación..., línea 14 en nivel 2 = 8 espacios de indentación. Dicho estilo de indentación está presente en algunos IDE de java como Netbeans.

c)- ***Llave en la estructura (CB):*** dicho estilo de programación indica que una llave (generalmente para declarar estructuras, clases, métodos) puede estar en la misma línea en que se declara la estructura o bien, en la siguiente línea. Ejemplo: en la figura 4.6, se observa que el código A, presenta la llave en la misma línea de la declaración de la estructura *for*y el código B en la siguiente línea. Generalmente, los estudiantes que utilizan una de las dos formas lo hacen en todo los casos dentro del código, por lo que simplemente se buscara la mayor incidencia y el análisis de éste estilo dará como resultado Sí o No.

d)- ***Líneas en blancos (BL):*** representa los espacios en blanco entre líneas de código y le dan una mejor vista y formato al mismo. Muchos estudiantes varían la cantidad de líneas en blanco, especialmente para separar estructuras o líneas de códigos de una sección. La métrica que se propone para éste estilo, está descrita

A

```
for (int k=1;k<10;k++) {
  //código
}
```

B

```
for (int k=1;k<10;k++)
{
  //código
}
```

Figura 4.6: Códigos que describen la declaración de las llaves.

```
if(var<2){return var;}else{return 0;}
```

Figura 4.7: Líneas de código inmersos en una misma línea.

en la siguiente fórmula:

$$PB = \frac{\sum_{b=1}^{B}(LB_b)}{\sum_{l=1}^{L}(L_l)} \tag{4.3}$$

dónde: *B* - posibles repeticiones de líneas en blanco dentro del código; LB_b - número total de líneas en blanco; L_l - es el número total de líneas que posee el código; *PB* - es la relación del número total de líneas en blanco entre el total de líneas que posee el código.

e)- ***Sentencias de código por línea (SCL):*** describe la cantidad o número de líneas de código finalizadas en ";" o bien, una parte o todo de una estructura que pueden estar en una misma línea. Ejemplo: la figura 4.7, representa líneas de códigos que normalmente irían en varias líneas y se encuentran en una sola. Los estudiantes de programación básica no programan de ésta manera, sin embargo existen estudiantes que puedan seguir un estilo de programación que si los empleen o simplemente descubran que si es posible unir dichas líneas. De esta manera, se

propone la siguiente fórmula para su medida:

$$PC = \frac{\sum_{l=1}^{L}(L_l)}{\sum_{c=1}^{C}(LR_c)} \tag{4.4}$$

dónde: *C* - es el posible número de líneas de códigos de referencias; LR_c - es el *c* veces líneas de referencias existente en el código; *PC* - es la relación de líneas de códigos totales entre líneas de códigos de referencias.

Las líneas de código de referencias pueden ser de 3 tipos y están basadas en el formato predeterminado de algunos IDEs de java como Netbeans: 1. Sentencias terminadas en ";", 2. Declaraciones de estructuras, métodos o clases seguidas de su llave, 3. Llaves de cierre. Ejemplo: el código de la figura 4.7, tiene 5 líneas de referencias.

f)- ***Documentación de las líneas de código (D):*** los estudiantes de programación básica no tienen como hábito la documentación de sus códigos; esto se debe a que les parece aburrido o innecesaria su utilización; sin embargo, poco a poco pueden ir descubriendo las bondades de la misma, a la medida que programen aplicaciones complejas. Para la medida de dicho estilo de programación se propone la siguiente fórmula:

$$PD = \frac{NCD}{\sum_{s=1}^{S}(LS_s)} \tag{4.5}$$

dónde: *NCD* - es el número total de caracteres de documentación dentro del código; *S* - número posible de líneas de código sin contar líneas de documentación solas; LS_s - es el número total de líneas de código sin documentación representado por las *s* posibles veces.

g)- ***Inicialización de variables (IV):*** inicializar las variables puede ser opcional; sin embargo, en algunos lenguajes como C, inicializar todas las variables al momento de su declaración es de vital importancia en la eliminación de registros basura de memoria. Algunos profesores exigen a sus estudiantes la inicialización de todas sus variables, aun utilizando lenguajes nuevos como Java. Esto se debe a que se ha creado como hábito utilizar esta buena práctica; otra razón, es el hecho de que

se evita errores de programación. Para algunos estudiantes inicializar todas las variables, aunque algunas no sean necesarias, es ser precavidos; para otros, es trabajo demás. Para la evaluación de este estilo, se propone extraer el porcentaje total de las variables inicializadas.

4.6.2. Preferencias de estructuras

Las preferencias en la programación son catalogadas como estilo de programación; sin embargo, para este trabajo, se ha dejado en una clasificación separada para su mejor apreciación. Dentro de la programación existen múltiples maneras de resolver un problema, siendo eficiente de igual forma. Estas múltiples maneras pueden ser: estructuras de decisión y repetición, métodos recursivos o inductivos y funcionalidades propias del lenguaje. La utilización de una de ellas, fuera de evaluar la eficiencia del algoritmo, se basa en la preferencia o gusto del programado.

Algunos algoritmos admiten la utilización de varias modalidades de ciclos, en el caso del FOR y del WHILE las funcionalidades que brindan pueden ser iguales; sin embargo, algunos estudiantes prefieren la utilización de ciclos WHILE únicamente. De igual forma, cuando se tiene muchas preguntas de decisión, algunos estudiantes pueden optar por la utilización de SWITCH-CASE para representarlo, en el caso de otros, utilizan solo el ELSE-IF o simplemente el IF-ELSE anidados. La herramienta SOFORG solo evaluara las preferencias de las estructuras de repetición y decisión, ya que son las preferencias en la que un estudiante de programación básica, puede incurrir. Por lo que se propone 4 tipos del mismo: Ifelse_elseif, Ifelse_elseif_switchcase, for_while_dowhile, while_dowhile.

a)- ***Ifelse_elseif (IE):*** en esta categoría, los algoritmos que empleen estructuras de decisión IF anidados dentro de un ELSE, pudiera reescribirse como un ELSE-IF. Por ejemplo: en la figura 4.8, se presenta dos códigos cuya funcionalidad es la misma, el código A expresado en estructura *if-else* anidados y el código B en forma de *else-if*. Los algoritmos que cumplan esta características y dependiendo de la estructura empleada, SOFORG indicará la preferencia en el análisis estático del código; si no existen apariciones de estas estructuras, entonces no se dará ningún resultado.

A

```
if (edad <= 10) {
     System.out.println("Es un niño");
 } else {
     if (edad <= 18) {
        System.out.println("Es un adolescente");
     }
```

B

```
if (edad <= 10) {
     System.out.println("Es un niño");
 } else if (edad <= 18) {
        System.out.println("Es un adolescente");
 }
```

Figura 4.8: Códigos de estructuras de decisión.

b)- ***Ifelse_elseif_switchcase (IES):*** en esta categoría entran los algoritmos en la cual, pudieran emplearse la estructura SWITCH-CASE; estos algoritmos tienen la misma forma que la anterior (un IF anidado en un ELSE), pero se diferencia en la declaración lógica. Todas las declaraciones lógicas en esta categoría, deben comprobar igualdad. Ejemplo: if (edad == 18)... Toda estructura SWITCH-CASE tiene su transcripción a IF-ELSE anidados o ELSE-IF. De esta manera, si se encuentra un SWITCH-CASE, SOFORG directamente dará como resultado una preferencia en la utilización de SWITCH-CASE sobre las demás.

c)- ***For_while_dowhile (FWD):*** Todo ciclo FOR tiene su representación en WHILE, pero no todo WHILE al FOR y todo ciclo WHILE puede ser reemplazado por un DO-WHILE y viceversa; esto dependerá de que dichas estructuras presenten sentencias con operadores lógicos $>$, $>=$, $<$, $<=$. Ejemplo de esta categoría: en la figura 4.9, se observa tres códigos con funcionalidad igual, utilizando diferentes estructuras de decisión.

d)- ***While_dowhile (WD):*** En esta categoría, se encuentra todos los demás casos en la que se emplee estructuras de repetición WHILE o DO-WHILE fuera de las condiciones expresada en la categoría anterior. Para esta categoría, dichas estructuras de repetición pueden emplear en su declaración lógica: igualdades, llamadas a funciones, varias declaraciones lógicas.

A

```
for(i=0;i<100;i+5){
    System.out.println("No. "+i);
}
```

B

```
i=0;
while(i<100) {
   System.out.println("No. "+i);
   i=i+5;
}
```

C

```
i=0;
do{
   System.out.println("No. "+i);
   i=i+5;
}while(i<100);
```

Figura 4.9: Códigos de estructuras de repetición.

4.6.3. Buenas prácticas

De igual forma, SOFORG evaluará el código fuente para sugerir buenas prácticas. En programación, las buenas prácticas representa el conjunto de patrones o estilos de programación que el estudiante debe aplicar para mejorar el rendimiento y la mantenibilidad de los programas, además de evitar posibles errores de programación. Esta información será útil para identificar posibles deficiencias, especialmente en la POO, a partir de la evaluación de aquellas buenas prácticas que el estudiante no ha aplicado. El análisis de las buenas prácticas es del enfoque basado en conocimiento, al igual que el de las preferencias de estructuras, explicado en el punto anterior. A continuación se presentan buenas prácticas que SOFORG puede evaluar:

a)- ***Atributos que no han sido inicializados en el constructor:*** Es una buena práctica de la POO, donde los atributos no deben ser inicializados en el constructor de su respectiva clase; sin embargo, solo aplica a atributos que no han sido declarados *static* o *final* y a los atributos que pertenecen a clases abstractas o interfaces.

b)- ***Atributos de tipo public static sin inicializar en su declaración:*** Todos los atributos de tipo *public static* deben ser inicializados en su declaración, sin importar que esté inicializado en un constructor. Esto se debe a que pudiera producir error de programación al ser utilizado por otra clase y no ha sido inicializado.

c)- ***Atributos de tipo final que deben ser declarados static:*** Cuando un atributo es declarado *final* y no *static*, esto provocará que cada instancia de la clase a la que pertenece dicho atributo, guardara un registro en memoria del mismo valor. Al ser declaro *static*, simplemente existirá un solo registro para todas las instancias. De esta manera, esto ayudará en el mejor aprovechamiento de la memoria.

d)- ***Atributos que no han sido declarados private o protected:*** Como buena práctica de POO, todos los atributos, excepto los declarados *final* o *static* y los atributos de las interfaces, deben ser declarados *private* o *protected*; por lo que deben ser accedidos a través de métodos dentro de la clase.

e)- ***Clases que no tienen constructor:*** Cada clase que no tenga constructor, por defecto, Java internamente le crea un constructor vacío; sin embargo, para efectos de mantenibilidad cada clase creada debe tener su propio constructor, aunque no sea necesario inicializar atributos. Las clases que están libre de ésta buena práctica son las que tienen método *main*, clases que son abstractas y las interfaces.

f)- ***Métodos que pueden ser declarados static:*** Los métodos estáticos son aquellos que se pueden utilizar sin instanciar su clase. La utilización de estos métodos mejora el rendimiento de los compiladores y programas al consumir menos memoria, ya que no se crean copias del método por cada instancia. De esta manera, representa una buena práctica en programación; por lo que SOFORG analizará el contenido de cada método y comprobará que todos los atributos y métodos que accede sean estáticos.

g)- ***Clases anidadas que pueden ser declaradas static:*** La utilización de las clases estáticas son similares a la utilización de los métodos de este tipo; por lo que representa una buena práctica su declaración. SOFORG tiene como criterio el acceso a otros métodos y atributos estáticos de la clase padre, para ser sugerida su declaración.

4.6.4. Evaluación académica del códigos

Proporcionar una posible evaluación del programa de los estudiantes, es de vital importancia en el análisis estático del mismo, ya que ayudará al profesor en dicha tarea. SOFORG utiliza el enfoque de evaluación por similitud (descrito en secciones anteriores) para el análisis de este resultado, ya que utiliza modelos preestablecidos de los ejercicios que el estudiante desarrolla; de esta manera, el resultado del estudiante es comparado con dichos modelos. Éstos modelos contienen las posibles soluciones de un ejercicio, en una estructura ontológica escrito en lenguaje OWL. Los ejercicios serán evaluadas en base a 100 %, por lo que cada parte de la estructura del problema tendrá una ponderación, por ejemplo: si un ejercicio implementa un ciclo, éste puede tener una ponderación de 13 % del total 100; las operaciones dentro del ciclo tendrán otro porcentaje y así sucesivamente. Si la declaración lógica del ciclo es errónea, las operaciones anidadas también lo serán y en total, se perderán todos los puntos del ciclo y las operaciones, aunque éstos se encuentren bien. Por otro lado, la posición y secuencia de las estructuras y operaciones dentro del programa tendrán otra ponderación.

No todos los problemas que se le presentan a los estudiantes pudieran tener modelos preestablecidos en OWL; esto dependerá de la complejidad, la utilización de funciones especiales de Java e interfaces gráficas. Por lo tanto, en caso de que no existiese un modelo, el profesor ingresará dicha evaluación a SOFORG, en donde será almacenado con el registro completo del respectivo análisis. Los descripción del diseño y desarrollo de esta sección de la herramienta, serán abordados en un capítulo aparte (Capítulo 5).

4.7. Determinación del análisis estático

La herramienta SOFORG analiza el código fuente de dos formas: 1. A través de GIC, y 2. A través de cadena de texto línea por línea; de esta manera, se dan dos evaluaciones del código fuente secuencialmente y se emite los resultados del mismo. La clase *analisisResultForm* es la encargada de gestionar todo el análisis estático dentro de la herramienta, y a partir de allí, se hacen los llamados a las demás clases que

ejecutaran las operaciones de los diferentes resultados, como fue explicado en el punto 4.4. En los siguientes puntos, se describirá cómo se determinará los resultados del análisis estático, tales como: estilos de programación, preferencias y buenas prácticas, que fueron expuestos en la sección 4.6; en cuando a los detalles de la determinación de la evaluación del código, se abordará dicho punto en el capítulo 5.

4.7.1. Determinación de los estilos de programación

Como fue mencionado anteriormente, los estilos de programación son determinados en la clase *estiloProgramacion*, observada en el diagrama de la figura 4.2. Esta clase analiza el código fuente en forma de cadena de texto (tipo de dato `String`), en la que se captura la información línea por línea, permitiendo contabilizar los estilos de programación de una manera más eficaz. Al terminar el escaneo del código fuente se procede a realizar el cálculos de las métricas para cada estilo, utilizando las fórmulas ya propuestas detalladas en el punto 4.6.1. Existen ciertas consideraciones específicas en el cálculo de los estilos de programación que serán detallados a continuación:

- En la longitud de los identificadores además de tomar en cuenta el nombre de las variables, clases, paquetes, métodos e interfaces; toma en cuenta las variables que han sido declaradas como índices en una estructura de repetición.
- Para el caso del cálculo de la indentación, la herramienta debe calcular aparte de los espacios al inicio de la líneas, los niveles de profundidad de las estructuras, es decir, cuando existen códigos y estructuras anidadas. Estos niveles son calculados para determinar los espacios de indentación de referencia como ya fue explicado en el punto 4.6.1. Para el cálculo de estos niveles se cuenta las líneas de código perteneciente a un nivel sin importar su secuencia, es decir que al final se tendrá para el nivel 0 un número de líneas, para el nivel 1 otro número y así sucesivamente.
- No se toma en consideración las líneas de documentación solas, para el cálculo de los espacios de indentación.
- La llave en la estructura no toma en cuenta ni la llave de la declaración de la primera clase, ni la llave del método `main`; esto se debe a que todo los IDEs proporcionan la clase con su método `main` ya declarados en el código.

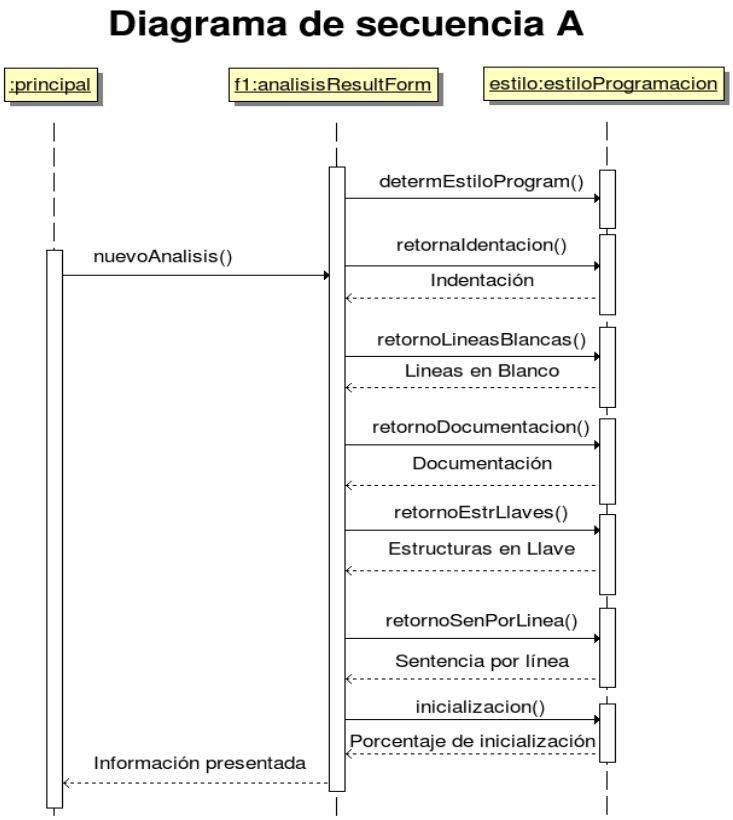


Figura 4.10: Diagrama de secuencia A para la determinación de los estilos de programación.

- En el caso de la documentación de las líneas de código no tomara en cuenta la documentación perteneciente a la información que el estudiante debe suministrar (cédula, código), como se presento en el punto 4.3.1. De esta manera, la herramienta al evaluar una línea de código que se encuentra en estado de documentación y encuentra cadenas como "`@ID:`" o "`@codigo:`", ésta no contabiliza dicha línea.

En la figura 4.10, se presenta el diagrama de secuencia para la determinación de los estilos de programación, en la que se ilustra mejor el orden lógico y las clases involucradas. El resultado de las métricas evaluadas es expresado en números decimales con 2 dígitos de decimales, salvo por los estilos de inicialización de variables y llaves en la estructura, que son expresados en decimales y sí o no respectivamente, como fue explicado anteriormente en el punto 4.6.1.

4.7.2. Determinación de las preferencias de estructuras

La determinación de las preferencias de los estudiantes esta a cargo de las clases *Analizador_ind* y *preferenciaCodigo* (representadas en la figura 4.3). La primera, como ya se mencionó en el punto 4.4, analiza el código fuente a través de GID, para identificar las estructuras empleadas por los estudiantes; de esta manera, cada vez que se identifica el tipo de estructura y a cuál categoría de preferencias pertenece (como fue explicado en el punto 4.6.2), se envía dicha información a la clase *preferenciaCodigo*. En esta clase, se suman las preferencias por categoría y al final del análisis se emite un resultado según el de mayor incidencia en la respectiva categoría; en caso de que no exista estructura para una categoría específica el resultado será nulo. La figura 4.11, muestra el diagrama de secuencia para este caso, donde se visualiza mejor lo explicado.

La herramienta SOFORG toma en consideración que algunas estructuras pueden ser declaradas como resultados de llamadas a métodos, este tipo de declaración son consideradas como preferencias en las categorías donde no se encuentre ni el SWITCH ni el FOR, para los casos respectivos.

4.7.3. Determinación de las buenas prácticas

En comparación a las anteriores, existe un mayor grado de complejidad en la determinación de las buenas práctica, debido a que se involucran muchas más clases en la misma. La clase *Analizador_ind* envía los parámetros de los componentes identificados en el código fuente, a la clase *instanciaClase*; esta clase controla la estructura de datos que almacena y manipula toda la información de los componentes de la POO que han sido aplicados en el código fuente, tales como: clases, objetos, atributos, métodos, constructores, interfaces; para cada componente, existe una clase que la gestiona además de otras que permiten ser heredadas. Todo esto es observado en el diagrama de la figura 4.3.

Como ya se mencionó en el punto 4.6.3, la herramienta SOFORG presenta en forma de sugerencias, las buenas prácticas que los estudiantes no han aplicado, en su mayoría relacionadas con la POO. A continuación se mencionan algunas consideraciones para la determinación de las buenas prácticas:

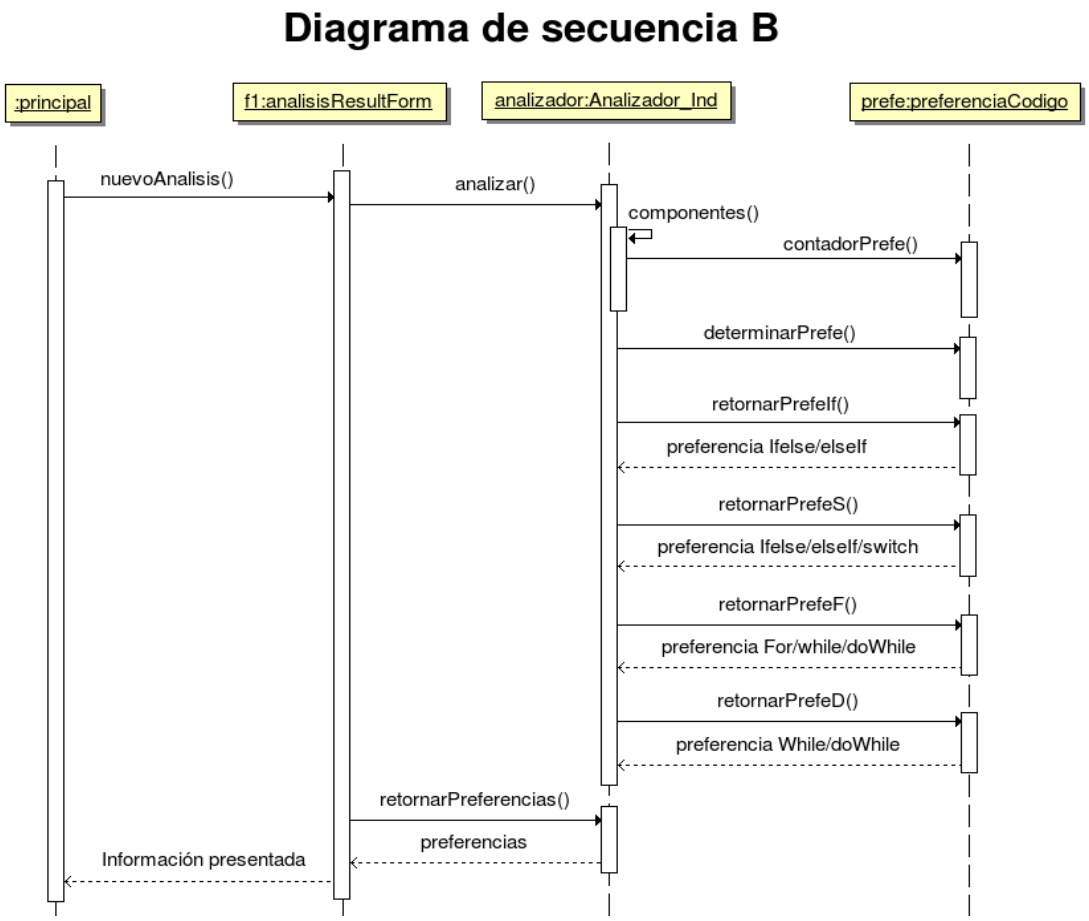


Figura 4.11: Diagrama de secuencia B para la determinación de las preferencias de estructuras.

- Para el caso de *Atributos que no han sido inicializados en el constructor*, explicado en el punto 4.6.3, la herramienta busca aquellos atributos declarados globales y que no han sido inicializados en el constructor, sin importar que se le haya especificado una inicialización en su declaración. En esta buena práctica se toma en cuenta, además de las variables, las declaraciones de vectores, matrices, objetos y todo lo considerado como atributo en la POO.

- De igual forma, al evaluar la buena práctica de inicializar los atributos estáticos en su declaración, se considera todo tipo de atributos sin importar que también sean declaradas final y que se inicialicen posteriormente en el constructor.

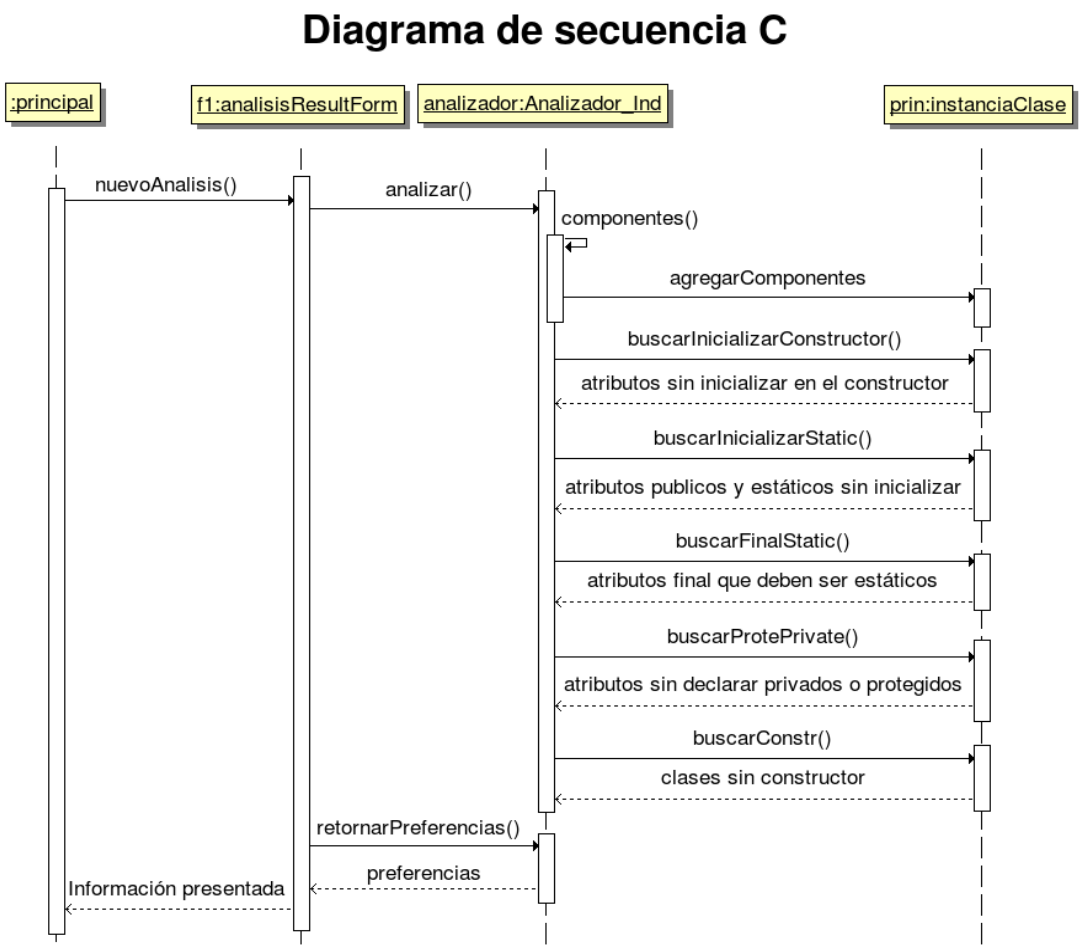


Figura 4.12: Diagrama de secuencia C para la determinación de las buenas prácticas.

Para una mejor representación del caso, se presenta el diagrama de secuencia de la determinación de las buenas prácticas, visualizado en la figura 4.12.

4.8. Formación de grupos de trabajos

Toda la información extraída del código fuente que ha de ser almacenada, será utilizada como referencia, en la formación de los grupos de trabajo. Como ya fue mencionado en el punto 4.4, la clase *generadorGruporForm* es la encargada de realizar dicho proceso, que a su vez utiliza la clase *combinaciones*, esta se utiliza únicamente para el tipo de formación por estilo de programación, en la que requiere evaluar las métricas de las mismas; esto se realiza creando las posibles combinaciones de formación

Tabla 4.1: Estructura de formación de los grupos (Arosemena-Trejos et al., 2010).

Tipos de Grupos	Basados en
Grupo de Par (P)	capacidades similares (CP) capacidades distintas (NCP) estilos de programación (EP) capacidades+estilos de programación (CP+EP) aleatorio (AP)
Grupo de 3 o más	capacidades similares (CP) estilos de programación (EP) capacidades+estilos de programación(CP+EP) aleatorio (AP)
Grupo con tutor (T)	capacidades similares (CP) estilos de programación (EP) capacidades+estilos de programación (CP+EP) aleatorio (AP)

de grupos y se toma la mejor evaluación, todo esto considerando las características y opciones de selección del tipo de grupo que el profesor puede seleccionar.

4.8.1. Tipos de formaciones de grupos

La estructura de formación de estos grupos utilizada por SOFORG, puede ser visualizada en la Tabla 4.1, aquí se observan las combinaciones de dos tipos de grupos para la formación final; por ejemplo, no se podrá formar un grupo de par sin especificar en que se basa su formación (capacidades iguales, distintas, estilos de programación, otros descritos); obteniendo así, el máximo provecho de las características de los grupos mencionados; sin embargo, no siempre el profesor conocerá cuál es el mejor tipo de grupo a formar, esto se debe a que dependerá del estado actual de los estudiantes de un curso, en cuanto a sus niveles de capacidades y estilos de programación Arosemena-Trejos et al. (2010). De esta manera SOFORG, brinda la opción de poder elegir el mejor grupo a formar, evaluando el estado actual del curso.

La capacidad de los estudiantes es registrada en cada análisis estático, esto se basada en la evaluación proporcionada y en una ponderación que se le dá a las sugerencias de buenas prácticas detectadas en el mismo; las capacidades tendrán una métrica de 0-100, en el punto 4.8.3, se explica mejor el cálculo del mismo. El tipo de

formación de grupos basado en diferentes capacidades (NCP), solamente es aplicado para el tipo de grupo Par (P), como es observado en la Tabla 4.1; esto se debe a que es en este tipo de grupo, es que se le puede sacar el provecho deseado como fue explicado en el punto 3.2.4. Por otro lado, el tipo de formación de *grupos de 3 o más*, es considerado a partir de 3, debido a que los grupos de 2 serán considerados como Par (P), para evitar confusiones.

4.8.2. Determinación de la mejor opción de formación

La herramienta SOFORG permite la selección de mejor opción en la formación de grupo, esto deja que la herramienta haga un análisis del estado actual del curso (evaluando capacidades y estilos de programación), y sugerir cuál es la mejor opción de formación de grupo; esto se debe a que muchas veces no se le saca el provecho máximo a los tipos de formaciones de grupos, debido a que no se comparten las características adecuadas, por ejemplo, si deseamos formar grupos basados en estilos de programación y el curso que se quiere evaluar cuenta con estudiantes con diferencias notorias en sus estilos de programación, el resultado será que los grupos formados sean poco eficiente; sin embargo, dicho curso posee características apropiadas para que sean formados basado en capacidades, esto resultaría mucho más provechoso que la anterior opción; como no se sabe cuál sería la mejor opción de formación, SOFORG proporciona una evaluación y sugiere dicha opción. En la figura 4.13, se presenta el algoritmo para la selección de la mejor opción; esta representada por estructuras de decisión en la que se basa para sugerir dicha formación de grupo; dónde: *porcenEstilo* - es el porcentaje de semejanza de los estilos de programación en los estudiantes del curso; *porcenCapaDife* - es el porcentaje de estudiantes que tienen capacidades mayores al promedio de capacidades del curso; *porcenCapaIgual* - porcentaje de estudiantes que tienen capacidades iguales; *numEstuCapaTutor* - es el número de estudiantes que tienen capacidades mayores al rango de promedio; *numeroDeGrupos* - es el número de grupos dado la cantidad de estudiantes por grupo y la cantidad total en el curso; *numEstuEstiTutor* - es el número de estudiantes que tienen capacidades mayores a 91. Cabe destacar que este algoritmo, se parte por evaluar primero la formación de grupo par y luego la de grupos de 5 ya sea con tutor o no, de igual forma los grupos basados en estilo de programación son evaluados con la formación de par de primero; esto se debe a que en cualquier momento específico, puede

```
1. if (porcenEstilo >= 75)
       //FORMACIÓN DE GRUPOS PAR BASADO EN
        ESTILOS DE PROGRAMACIÓN
2. else if (porcenCapaDife > 45 and porcenCapaDife < 55)
       //FORMACIÓN DE GRUPOS PAR BASADO EN
        DIFERENTES CAPACIDADES
3. else if (porcenCapaIgual > 75)
4.      if (numEstuCapaTutor >= numeroDeGrupos)
                //FORMACIÓN DE GRUPOS DE 5 BASADO
                EN CAPACIDADES IGUALES CON TUTOR
        else
                //FORMACIÓN DE GRUPOS DE 5 BASADO
                EN CAPACIDADES IGUALES SIN TUTOR
5. else if (numEstuEstiTutor >= numeroDeGrupos)
       //FORMACIÓN DE GRUPOS DE 5  BASADO EN
       ESTILOS DE PROGRAMACIÓN CON TUTOR
   else
       //FORMACIÓN DE GRUPOS DE 5 BASADO EN
       ESTILOS DE PROGRAMACIÓN SIN TUTOR
```

Figura 4.13: Algoritmo de sugerencia para la mejor opción de formación de grupos.

obtenerse mucho más beneficio en el proceso de enseñanza-aprendizaje, aplicando dicha formación, que con el resto de las formaciones, debido a las bondades que presenta la programación en grupo par y los grupos basado en estilo de programación, como fue explicado en el punto 3.2.4.

4.8.3. Algoritmos de formación de grupos

Una vez seleccionado el tipo de grupo, ya sea por selección manual o utilizando la opción de mejor opción, como se explico en el punto anterior, SOFORG procede a crear los grupos con los parámetros indicados. A continuación se detalla los algoritmos y consideraciones más importante en la formación de grupos.

Capacidades similares

La herramienta SOFORG permite la creación de grupos basados en capacidades similares (CP) como fue detallado en el punto 4.8.1. Es preciso aclarar que las capacidades de los estudiantes representa una métrica que no es directamente extraída del análisis estático del código de los estudiantes; sin embargo, se propone un método de cálculo en la que a través de la evaluación del programa y las buenas prácticas que se presentan como resultado del análisis del código del estudiante, se proporciona una métrica de 0-100 para el mismo. Dicho cálculo consiste en tomar la evaluación del análisis (de 0 a 100) y restarle 3 por cada sugerencia de buenas prácticas detectadas, es decir, si tenemos una evaluación de 80 y, 4 sugerencias de buenas prácticas, la métrica para la capacidad quedara en 78 en dicho análisis estático que será almacenado con la demás información del estudiante.

El algoritmo propuesto para la formación de grupos consiste en tomar el valor de las capacidades de los estudiantes para un curso deseado y ordenarlas, ya sea ascendente o descendente; luego, se procede a formar los grupos con los estudiantes ordenados ya ordenados, en la que se toman contiguamente dependiendo del número de estudiantes y así formarlos de tal manera que los grupos tengan integrantes con las capacidades próximas. En la Tabla 4.2, se presenta un ejemplo de como quedaría una formación de grupos de 2 basado en capacidades similares, en la que se puede apreciar los estudiantes ordenados por el valor de sus capacidades y la pertenencia a sus respectivos grupos.

Capacidades diferentes

Este tipo de formación (NCP) consiste en ordenar, de igual forma, los estudiantes basados en sus capacidades, de manera ascendente o descendente; una vez ordenado los estudiantes se dividen en 2 grupos iguales, de tal manera que en un listado esté un extremo mayor de capacidades y en el otro el menor, como se muestra en la Tabla 4.3. Cuando se tiene la división del curso en partes iguales, como ya fue indicado, los grupos se formaran tomando el primero de la parte 1 con el primero de la parte 2, el segundo de la parte 1 con el segundo de la parte 2 y así sucesivamente, de tal manera que queden agrupados como se observa en la Tabla 4.4. Los grupos resultantes están formados con la mayor diferencia posible de capacidades de los estudiantes.

Tabla 4.2: Formación de grupos de 2 basado en capacidades similares

Grupos	Estudiantes	Capacidades
Grupo A	Estudiante 3	98
	Estudiante 4	90
Grupo B	Estudiante 2	89
	Estudiante 5	87
Grupo C	Estudiante 6	85
	Estudiante 1	82

Tabla 4.3: División de los estudiantes en 2 partes iguales

Partes	Estudiantes	Capacidades
Parte 1	Estudiante 3	98
	Estudiante 4	90
	Estudiante 2	89
Parte 2	Estudiante 5	87
	Estudiante 6	85
	Estudiante 1	82

Tabla 4.4: Formación de grupos de 2 basado en capacidades similares

Grupos	Estudiantes	Capacidades
Grupo A	Estudiante 3	98
	Estudiante 5	87
Grupo B	Estudiante 4	90
	Estudiante 6	85
Grupo C	Estudiante 2	89
	Estudiante 1	82

Tabla 4.5: Posibles formaciones de grupos de 2 para 6 estudiantes

No.	Grupo A	Grupo B	Grupo C
1	12	34	56
2	32	14	56
3	32	54	16
4	32	64	51
5	42	31	56
6	42	51	36
7	42	61	53
8	52	34	16
9	52	14	36
10	52	64	13
11	62	34	51
12	62	54	31
13	62	14	53

Como fue mencionado en secciones anteriores, la formación de grupos basado en capacidades diferentes (NCP) será aplicado por SOFORG, solamente para la formación de grupos par (P).

Estilos de programación

Para este caso, es necesario la utilización de la clase *combinaciones* como ya fue mencionado, en la que se crea las posibles formaciones de grupos para un curso determinado y así evaluarlas uno por uno. Cabe destacar que el algoritmo empleado es de complejidad polinomial O(n^a), debido a que el sistema debe comparar uno por uno toda las posibles formaciones de grupos dentro del curso. Las posibles formaciones de grupos se dá de la siguiente manera: se supondrá que se tiene 6 estudiantes en un curso identificados con su respectivo número del 1 al 6, se formará grupos de 2; las posibles formaciones de grupos en el curso se visualizan en la Tabla 4.5. Se observa así, 13 posibles formaciones de grupos existen para los parámetros especificado, en donde cada estudiante es comparado con las métricas de los demás miembros del grupo, sacando una valoración por grupo que luego será evaluado con los demás grupo y así obtener una valoración global. Esta valoración representa la distancia o diferencia de las métricas almacenadas de cada estudiante con la del otro, en la Tabla 4.6 se presenta un ejemplo del cálculo de la distancia o diferencia entre dos estudian-

Tabla 4.6: Calculo de la diferencia entre estudiante 1 y 2

Métricas	Estudiante 1	Estudiante 2	Diferencia
Longitud de identificadores	5.3	2.4	2.9
Indentación	1.34	1.6	0.26
Llave en la estructura	sí	no	1.0
Líneas en blancos	0.34	0.74	0.4
Código por líneas	0.88	1.02	0.14
Documentación	2.34	1.45	0.89
Inicialización	60	90	30
Prefe1. Ifelse_elseif	null	null	0.5
Prefe2. Ifelse_elseif_switch	elseif	switch	1.0
Prefe3. for_while_dowhile	for	null	0.5
Prefe4. while_dowhile	while	while	0.0

tes. Para el caso de los parámetros de llave en la estructura y las preferencias que no son representados con métricas numéricas, la herramienta tendrá la siguiente ponderación: valores iguales - diferencia 0.0, valores diferentes - diferencia 1.0, uno o los dos valores nulos - diferencia 0.5; este último se debe a que se asume que el estudiante tiene probabilidad de 0.5 de tener la misma preferencia del otro estudiante, al poseer un valor nulo; por lo que se valora con la mitad de 1, que es el valor propuesto como distancia promedio.

Cuando se obtiene el valor de todas las posibles combinaciones, incluyendo el cálculo de la diferencia entre grupos de una posible formación de grupos, entonces se obtiene una valoración global de cada posible formación de grupos en el curso; en la Tabla 4.7, se observa dicha valoración global por posible formación. En esta tabla solamente se incluye las 3 primeras posibles formaciones de grupo de la Tabla 4.5, en la que se observa un valor total que representa la valoración global para la posible formación de grupo; para las 13 posibles formaciones de la Tabla 4.5, existirá un valor total que representa dicha valoración. Este valor total es obtenido a través del promedio de todos los valores de las métricas contenidas. De esta manera, cada posible formación de grupo tendrá su valor global y se elegirá el de menor valor, ya que éste representará aquella formación que posee menor distancia o diferencia de métricas entre sus estudiantes que el resto de las formaciones, en otras palabras, aquella formación cuyo grupos tienen estudiantes con la mayor similitud en sus estilos de programación.

En la Tabla 4.8, se presenta los valores globales para las 13 posibles formaciones

Tabla 4.7: Valoración global de las métricas de las posibles formaciones

Métricas	**1**	**2**	**3**
Longitud de identificadores	4.3	3.4	6.9
Indentación	2.34	3.6	0.46
Llave en la estructura	1.0	0.5	1.0
Líneas en blancos	1.34	0.94	0.56
Código por líneas	1.88	2.02	0.24
Documentación	2.34	1.45	5.89
Inicialización	70	80	70
Prefe1. Ifelse_elseif	0.5	0.5	0.5
Prefe2. Ifelse_elseif_switch	1.0	0.5	1.0
Prefe3. for_while_dowhile	0.5	1.0	0.0
Prefe4. while_dowhile	0.0	0.5	0.0
Valor global	**7.75**	**8.58**	**7.86**

Tabla 4.8: Posibles formaciones ordenadas ascendéntemente por su valor global.

No.	**Valor global**
10	**6.75**
12	7.65
3	7.54
1	7.75
5	7.79
7	7.81
13	7.86
9	7.87
6	7.91
4	7.96
11	7.99
8	8.34
2	8.58

Tabla 4.9: Grupos formados basado en estilos de programación similares

Grupos	**Estudiantes**
Grupo A	Estudiante 5
	Estudiante 2
Grupo B	Estudiante 6
	Estudiante 4
Grupo C	Estudiante 1
	Estudiante 3

de grupos de la Tabla 4.5, ordenadas de manera ascendente, en la que se visualiza que la formación número 10 es la de menor distancia o diferencia de métricas; esta formación se observa en la fila 10 de la Tabla 4.5, cuyo orden son los estudiante 5 y 2 para el Grupo A, 6 y 4 para el Grupo B y para el grupo C los estudiante 1 y 3; estos pueden ser visualizados mejor en la Tabla 4.9. De esta manera, la formación No. 10 es la más optima de todas debido a que presenta la menor distancia entre sus métricas y por consiguiente la de mayor semejanza de estilos de programación entre sus estudiantes.

Sin embargo, la formación de los grupos basado en estilos de programación está limitada al número de estudiantes en el curso, al número de estudiantes por grupo y a la capacidad de procesamiento del computador. En una computadora con especificaciones comunes (e.g. procesador Intel i7 de 2.5 4 núcleos, 4GB de memoria RAM), la herramienta SOFORG puede evaluar aproximadamente 1.8 millones de opciones de formación de grupos, esto equivale a 15 estudiantes en un curso aproximadamente, formándolos en grupo de 2 o 3 estudiantes; cabe destacar que entre más estudiantes por grupo se seleccione, menos número de posibles grupos por curso existirán. En este escenario y con las especificaciones de la computadora presentada, la herramienta puede tardar de 10 a 15 segundos en emitir el resultado; sin embargo, cuando se comprueba que se excederá el tiempo y la capacidad de procesamiento, la herramienta acota el número de opciones de formación de grupos; todo esto de manera progresiva, por lo que se reduce así, la probabilidad de obtener la mejor opción de formación, es decir, si el número de posibles opciones de formación de grupo basado en estilo de programación es de 2 millones para un caso específico y el sistema está acotado en 1.8 millones, el sistema desechará 200 mil formaciones de grupos de manera aleatoria, reduciendo la probabilidad de obtener el resultado más optimo a un 90 %; esto significa que se obtendrán resultados óptimos pero no el más optimo.

Capacidades + estilos de programación

Este tipo de formación de grupo, involucra dos tipos de grupos la formación basada en capacidades similares (CP) y la formación basada en estilo de programación similares (EP); considerando ambos tipos de formación se genera los grupos requeridos. Para este caso, el algoritmo empleado es igual al algoritmo propuesto en el punto anterior, donde se consideran las métricas de los estilos de programación; sin embargo,

Tabla 4.10: Extracción de tutores en la formación de grupos de 3

Grupos	Estudiantes	Capacidades
Tutores	Estudiante 3	98
	Estudiante 5	87
	Estudiante 4	90
Resto de estudiantes	Estudiante 6	85
	Estudiante 2	89
	Estudiante 1	82
	Estudiante 9	82
	Estudiante 7	77
	Estudiante 8	75

Tabla 4.11: Formación de grupos de 3 basado en tutores

Grupos	Estudiantes
Grupo A	Tutor: Estudiante 3
	Estudiante 2
	Estudiante 7
Grupo B	Tutor: Estudiante 5
	Estudiante 8
	Estudiante 6
Grupo C	Tutor: Estudiante 4
	Estudiante 9
	Estudiante 1

para éste, se considera la métrica de las capacidades conjuntamente con las de los estilos de programación y así determinar los posibles grupos.

Grupos con tutor

Este tipo de grupo es formado por SOFORG, tomando en cuenta los otros tipos de formaciones, como se observó en la Tabla 4.1. El procedimiento propuesto es extraer primero los estudiantes tutores, luego formar los estudiantes basado en el otro tipo de grupo deseado y finalmente, agregarles a cada grupo resultante, los tutores ya extraídos. La extracción de los tutores consiste en ordenar los estudiantes de forma descendente basado en sus capacidades, luego se extrae los tutores que pertenecen a los primeros de mayor capacidad, el número de tutores dependerá de el número de grupos a formar. En la Tabla 4.10, se observa la ordenación de 9 estudiantes de ma-

nera descendente basado en sus capacidades; se considera la formación de grupos de 3 estudiantes, esto nos da un número de grupos a formar 3, por lo que se observa la extracción de los 3 primeros estudiantes que representan los de mayor capacidad. El resto de los estudiantes se le aplicará el algoritmo del tipo de grupo que se desea formar, con la diferencia de que la formación debe ser con un integrante menos; para este caso, la formación con el resto de los estudiantes será de 2 ya que el tercer integrante será uno de los estudiantes tutores. De esta manera, en la Tabla 4.11, se presenta la posible formación de los grupos incluyendo los tutores. Los tutores son incluidos en los grupos del resto de los estudiantes sin importar su orden.

4.9. Variables internas de la herramienta SOFORG

Estas variables son aquellas utilizadas internamente en la herramienta SOFORG, éstas representan valores, rangos y porcentajes que sirven como referencia a la hora de decidir operaciones en el módulo de formación de grupos; por otro lado, SOFORG tendrá la posibilidad de editar dichos valores con el objetivo de sensibilizar la herramienta a la hora de verificar la semejanzas de los estilos de programación, capacidades, estudiantes tutor. En la Tabla 4.12, se presenta los valores por defectos de dichas variables que a continuación se describen:

- *porcentajeEstiloPar:* variable de tipo entero que representa un porcentaje de semejanza de los estilos de programación en los estudiantes del curso; es utilizada en la declaración lógica número 1, de la estructura de decisión que se visualiza en el algoritmo de selección de la mejor opción (figura 4.13).
- *rangoCapacidades:* variable de tipo entero que representa un rango para la comprobación de que en el curso existe un porcentaje semejante de capacidades altas de los estudiantes y capacidades bajas; en la declaración lógica número 2 del algoritmo de la figura 4.13, se observa la evaluación de porcentajes mayores del promedio entre 45 y 55, estos números son extraído del valor por defecto de dicho rango 10.
- *rangoCapacidIgual:* es utilizada para determinar el rango de referencia en la comprobación del porcentaje de estudiantes que tienen capacidades semejantes, este rango es a partir del promedio de capacidades en el curso; esta variable

Tabla 4.12: Valores por defecto de las variables internas

Variables	Valores
porcentajeEstiloPar	75
rangoCapacidades	10
rangoCapacidIgual	10
porcentaCapacidIgual	75
porcentaEstilosIguales	75
porcentaPrefeIguales	50
numIntGrupTutor	5
numIntGrup	3
porcTutor	91
ranLong	3
ranInde	1
ranDocu	3
ranLine	0.2
ranSent	0.2
ranInic	10
numEvaluacion	3

es de tipo entero y es implementada para el cálculo del valor de *porcenCapaIgual*, utilizado en el algoritmo de la figura 4.13.

- *porcentaCapacidIgual:* es el porcentaje de referencia utilizado para comprobar la sentencia lógica número 3 del algoritmo de selección de la mejor opción (figura 4.13). Es de tipo entero y su valor por defecto es 75.

- *porcentaEstilosIguales:* variable de tipo entero que hace referencia a un porcentaje para la evaluación por métrica, de la semejanza de los estilos de programación.

- *porcentaPrefeIguales:* representa un porcentaje para la evaluación de la semejanza de las preferencias en estructuras; es de tipo entero.

- *numIntGrupTutor:* en la selección de la mejor opción de grupos, cuando se descarta la formación de grupos Par, se utiliza para determinar el valor de integrantes por grupos incluyendo tutor; por defecto será 5 y se encuentra contenida en esta variable de tipo entero.

- *numIntGrup:* de igual forma que la anterior, representa el número de integrantes por grupo, pero sin incluir estudiante tutor; por defecto será de 3 y tipo de variable

entera.

- *porcTutor:* esta variable es de tipo entero y almacena el porcentaje de capacidad que, igual a dicho valor o más, representarían tutores para los grupos a formar.

- *ranLong:* rango de referencia para verificar la semejanza de la métrica Longitud de los identificadores para cada estudiante; dicha variable es de tipo *double*.

- *ranInde:* variable de tipo *double* que representa el rango de referencia para evaluar la semejanza por estudiante de la métrica Indentación.

- *ranDocu:* esta variable hace referencia de igual forma, al rango para evaluar la semejanza de la métrica Documentación, esto a través del tipo de dato *double*.

- *ranLine:* de igual forma, la métrica Líneas en Blancos, podrá ser evaluada en los estudiantes en donde se busca la semejanzas a través del rango de referencia que representa dicha variable de tipo *double*.

- *ranSent:* la semejanza de la métrica Sentencias por Línea es evaluada a través del rango representado por esta variable que es de tipo *double*.

- *ranInic:* variable de tipo entero, que representa el rango para evaluar la semejanza de la métrica Inicialización de Variable.

- *numEvaluacion:* esta variable representa un valor de referencia, utilizada para determinar el número de evaluaciones trascendente que se les hará a los estudiante, es decir, si un estudiante presenta registros de 10 ejercicios, el sistema evaluará la información de los últimos 3 que han sido almacenados.

4.10. Ventajas y desventajas de la arquitectura

Al presentarse la propuesta de la arquitectura completa de la herramienta SOFORG, se plantean ciertas ventajas y desventajas del mismo. Éstas involucran, tanto el marco de desarrollo, como el marco funcional de la herramienta propuesta.

4.10.1. Ventajas de la arquitectura

Entre las ventajas existentes se tiene que:

- Al presentarse la capa Lógica de Negocio, en el lado del servidor, se mejora el rendimiento de los procesos dentro de SOFORG.
- En cuanto al proceso de desarrollo, se mejora el esfuerzo humano en programación, al implementar un analizador e interprete léxico y sintáctico, como es el caso de JavaCC, ya que se trabaja con GIC y Java inmerso en el mismo. Éste le dá el comportamiento deseado, al identificar un componente dentro del código fuente, al mismo tiempo que es expresado en lenguaje Java.
- La utilización de ontologías para la evaluación del código fuente, permite reducir la programación de líneas de códigos, por otro lado, se obtiene una mejor mantenibilidad del software, debido a las bondades, en cuanto a formalidad y estándar del lenguaje OWL. Por otro lado, proporciona una herramienta que facilita la construcción de modelos bases de los ejercicios para la evaluación de los mismos, ya que cada ejercicio representa un modelo diferente que debe ser construido.

4.10.2. Desventajas de la arquitectura

De igual forma, existen desventajas que han sido identificadas al presentar dicha arquitectura. Se detallan a continuación:

- Si bien es cierto que, utilizar ontologías mejora el proceso de desarrollo de la herramienta, el modelo interno, con el que las ontologías se ejecutan al aplicarles un razonador ontológico (como de detallará en el capítulo 5), resulta no ser el más optimo para el caso específico de esta herramienta. Esto provoca que exista limitaciones en el rendimiento, cuando la solución a evaluar y el modelo ontológico base del ejercicio, sean extensos o complejos en la cantidad de posibles soluciones para el mismo.

- El diseño y construcción de las ontologías bases de los modelos de ejercicios, representa una tarea que debe ser desarrollada, preferiblemente, por especialistas en el diseño de ontologías.
- La complejidad de orden polinomial del algoritmo utilizado para la formación de grupos basado en estilos de programación, compromete el tiempo de la ejecución de esta tarea, a medida que el número de estudiantes y grupos aumenta.

Capítulo 5

Evaluación ontológica del código fuente

SOFORG permite la evaluación académica del código fuente del estudiante, donde éste, proporciona una valoración dependiendo de la ponderación dada a dicho ejercicio, como fue explicado en el punto 4.6.4. Este capítulo presenta los detalles en cuanto a la arquitectura a seguir para la determinación del mismo; un conjuntos de reglas lógicas serán detalladas, conjuntamente con los componentes más importantes del código fuente que serán considerados en esta operación; por otro lado, los aspectos de las ontologías construidas como las propiedades, clases y sus relaciones jerárquica; aparte de los criterios de evaluación para el mismo.

5.1. Determinación de la evaluación académica del estudiante

Para la determinación de esta operación, se hace uso de la clase *Analizador_Ind*, ya que es la encargada del análisis estático con GIC, en la que se va detectando los principales componentes del código fuente y realizando una interpretación del mismo; por lo que, a la medida que se va analizando el código e identificando dichos componentes, se irán haciendo los llamados a los diferentes métodos de la clase *evaluadorOWL*, que debe ser instanciada previamente, en la que se irán agregando los

diferentes componente a una estructura de datos que permitirá crear las instancias de la ontología base; cada componente tendrá su propio método en dicha clase, para la creación de su respectiva instancia. De esta manera, el código estará representado en instancias o individuos de una ontología base, dicha ontología base representa los posibles modelos correctos de los ejercicios que el estudiante desarrolló; las instancias harán referencia a dicha ontología base en la que serán comparadas con el mismo y se calculará la evaluación, todo esto utilizando las librerías de JENA (detallada en el punto 4.2.3). Por lo tanto, se retorna el valor de la evaluación, considerando los criterios y ponderaciones ya especificados en el ejercicio modelo que se está analizando. En la figura 5.1, se presenta un diagrama que detalla lo explicado anteriormente; aquí se observa la secuencia que se sigue en el procesamiento y ejecución de las operaciones, en la que se describen las clases involucradas y sus respectivas relaciones, de igual forma, se representan las diferentes instancias y métodos utilizados, que describen las funcionalidades de este módulo.

Para un mejor entendimiento de lo anterior, el diagrama de actividades que presenta la figura 5.2, describe el flujo de trabajo a seguir, para la evaluación académica del código, en la que se tiene como pre-condición, la existencia de un modelo ontológico base para el ejercicio que se desarrollo. Aquí se observa que, una vez cargada el o los modelos ontológicos base, es necesario, generar las instancias ontológicas de los componentes; de esta manera, se debe identificar cuál de los modelos es el empleado por el estudiante, y así generar la evaluación del mismo.

5.2. Reglas lógicas y consideraciones

Para el desarrollo de este módulo, es necesario considerar ciertos criterios y reglas lógicas que permitirán mejorar la eficiencia del mismo. Si bien es cierto que, el desarrollo de programas no sigue un patrón estricto de desarrollo, las herramientas que componen un lenguaje y las propiedades del lenguaje en sí, si pueden seguir sus propias consideraciones lógicas; tomando en cuenta esas consideraciones se puede hacer una mejor evaluación del código de los estudiantes, a través del desarrollo de este módulo, específicamente en el diseño de las ontologías. A continuación se mencionan algunas reglas lógicas y criterios que serán considerados:

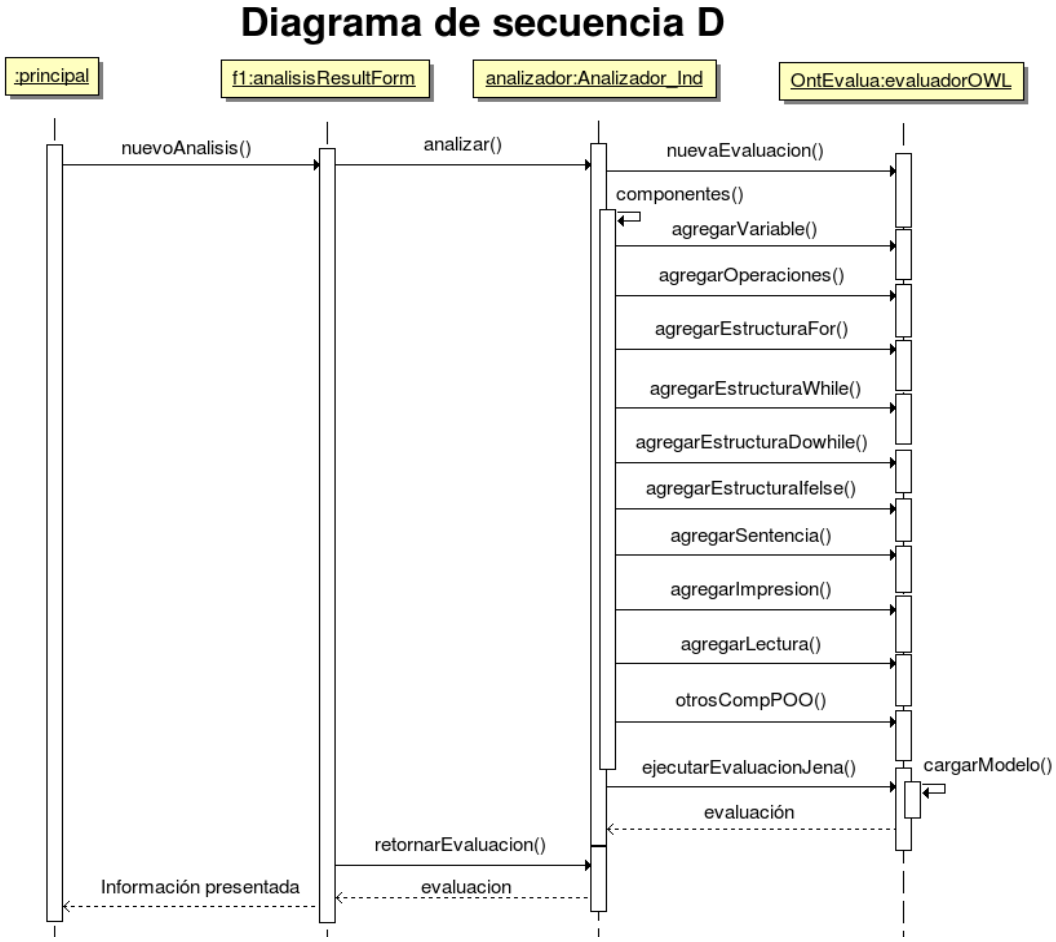


Figura 5.1: Diagrama de secuencia D, para la evaluación ontológica del código fuente.

- Se debe evitar cuartar la creatividad del estudiante ya que el estudiante aprende más al querer implementar otras funciones para mejorar la vista, impresión o colocarlo a su gusto.
- Un enunciado de problema puede presentar muchas soluciones y éstas dependerán de la complejidad y tamaño del mismo.
- Las sentencias lógicas pueden ser expresadas de distintas maneras y trabajarían de la misma forma, ejemplo: (edad $<=$ 10) o (edad $<$ 11) o (10 $>=$ edad) o (11 $>$ edad).
- La sentencia lógica puede ser reemplazada por una llamada a función.
- Las operaciones aritméticas pueden ser expresadas de diferentes maneras y representarán el mismo resultado. Ejemplo: i = i+1; o i++; o i += 1;.

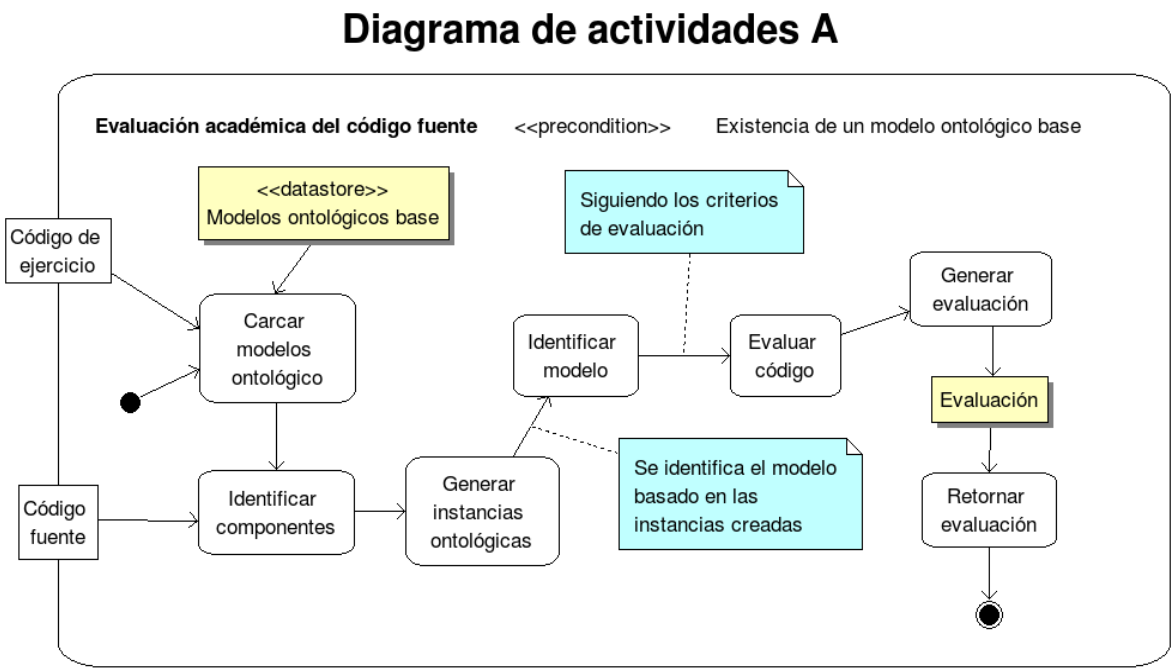


Figura 5.2: Diagrama de actividades A, que describe el proceso de evaluación del código fuente.

- Una sentencia lógica compuestas por varias y separadas por AND en un IF-ELSE, puede considerarse en IF-ELSE separados en forma anidada.
- Al utilizar un IF-ELSE anidado en otro IF-ELSE de la parte ELSE, el primero puede ser considerado como un ELSE-IF, este principio se detalla en el punto 4.6.3.
- Toda estructura FOR puede ser transformada a estructura WHILE y DO-WHILE; sin embargo, no toda estructura WHILE puede ser FOR, pero si DO-WHILE, este principio es detallado en el punto 4.6.3.
- Si existe una solución SWITCH-CASE existe una solución IF-ELSE anidado para el mismo, sin importar el orden, detallado en el punto 4.6.3
- En una misma estructura y nivel jerárquico se tomara como una sola impresión si existe más de una impresión sin involucrar variables.
- El orden secuencial de los componentes solo es relevando, si existen otros componentes en el mismo nivel jerárquico de anidamiento.

- Cada variable tiene un ciclo de vida que debe cumplir a través de operaciones asociadas a la misma.

5.3. Arquitectura de la ontología propuesta

La construcción de una ontología requiere de procesos iterativos (Noy y McGuinness, 2005) que permitan moldear y ajustar el funcionamiento correcto del mismo, en una aplicación determinada. Debido a esto, en la fase de diseño y construcción de las ontologías propuesta, es imprescindible la utilización de una aplicación que permita, además de construir, modelar la ontología que se está creando, brindando la posibilidad de editar la misma, de una manera sencilla y rápida. Por lo tanto, se hará uso de la herramienta Protégé (descrita en el punto 4.2.2), para el diseño y construcción de la ontología, este permitirá realizar las pruebas pertinentes y evaluar el comportamiento del mismo; una vez que se tenga el diseño construido, se generará la ontología base en un archivo con extensión OWL que luego será implementada en SOFORG a través del framework de JENA (descrito en el punto 4.2.3).

La arquitectura de la ontología consiste en un modelo ontológico base que contendrá todo el conocimiento necesario y un conjunto de instancias que harán referencia al mismo. A través de un razonador se determinarán si las propiedades de las instancias le hacen ser miembros o no de una clase, dentro del modelo ontológico base; ésta clase puede representar cualquier componente con propiedades preestablecidas, como por ejemplo, una clase que represente una variable de tipo entero con una asignación inicial de 0; de esta manera, si un individuo o instancia cumple con dichas propiedades, se inferirá que pertenece o es miembro de la clase con las propiedades descrita, sin especificar de antemano, su pertenencia; esto significa que en la solución del estudiante se ha declarado la variable, tal como se encuentra especificada en el modelo base y si de esta manera, se identifican otros componentes, se pudiera identificar entonces, el modelo específico que el estudiante desarrolló. Cabe destacar que un modelo ontológico base, puede representar un conjunto de posibles modelos de soluciones para un determinado ejercicio.

De esta manera, la ontología identificará primero, cuál es el modelo que el estudiante implementó, esto se realiza, a través de la comparación de propiedades en la

que una clase puede representar un modelo específico; una vez se identifique ésto, se procederá a evaluar si los componentes plasmado en el código fuente del estudiante son correctos conforme a las propiedades que esta posean. A simple vista se observa que, la composición del algoritmo para la ejecución de ésta tarea, implica una serie de estructuras de decisiones que se ejecutarán para realizar las comparaciones respectivas; ésto llevará a la obtención de un algoritmo con complejidad de $O(NM)$, dónde N, representa el número de propiedades de las instancias; y M, el número de propiedades de las clases que van a ser comparados uno por uno, dependiendo del dominio del componente. Evidentemente el valor de M y N variaran dependiendo de la estructura de dato que maneje las ontologías, por ejemplo: JENA maneja su propia estructura de datos al cargar en memoria una ontología, Protégé posee otra; es muy probable que existan diferencias entre los mismos al ejecutarse un razonador ontológico. Cabe destacar que JENA posee especificaciones de modelos ontológicos preestablecidos del tipo de razonador a utilizar cuando se crea una ontología, esto se selecciona, dependiendo de el tipo de razonamiento que se desea para una aplicación determinada y el lenguaje empleado; en la figura 5.3, se presenta una vista de algunas opciones que pueden ser seleccionadas, estas representan constantes de la clase *OntModelSpec* de JENA y son empleadas a través de un objeto recipiente con el mismo nombre; éstas, configuran el modelo ontológico para un lenguaje y razonador específico. Las ontologías creadas para SOFORG, utilizarán la configuración *OWL_DL_MEN_RULE_INF*, éste describe que los modelos ontológicos creados, estarán basadas en las características del lenguaje OWL de tipo DL, en la que trabajará bajo un razonador con funciones completas para dicho lenguaje.

Cabe destacar que, las ontologías bases de los modelos de ejercicios, estarán almacenadas en base de datos, es decir, en memoria persistentes y no en archivos RDF u OWL, en la que el sistema debe cargarla previamente para crear un modelo con las instancias que se van creando; en la figura 5.4, se presenta las sentencias de códigos para la creación del mismo, en donde se observa que la línea 42, carga el modelo base del ejercicio, después de haberse creado un modelo local de operación en la línea 41 y haberse establecido la conexión en la 38; la línea 45 y 46, creará un modelo ontológico en la que se especificará la configuración *OWL_DL_MEN_RULE_INF* y el modelo base que fue previamente cargado; esto permitirá que en un mismo modelo, representado con el tipo de dato *OntModel*, pueda irse agregando las instancias de los componentes haciendo referencia al modelo base; al final se tendrá una ontología completa con los conocimientos inferidos de los mismos.

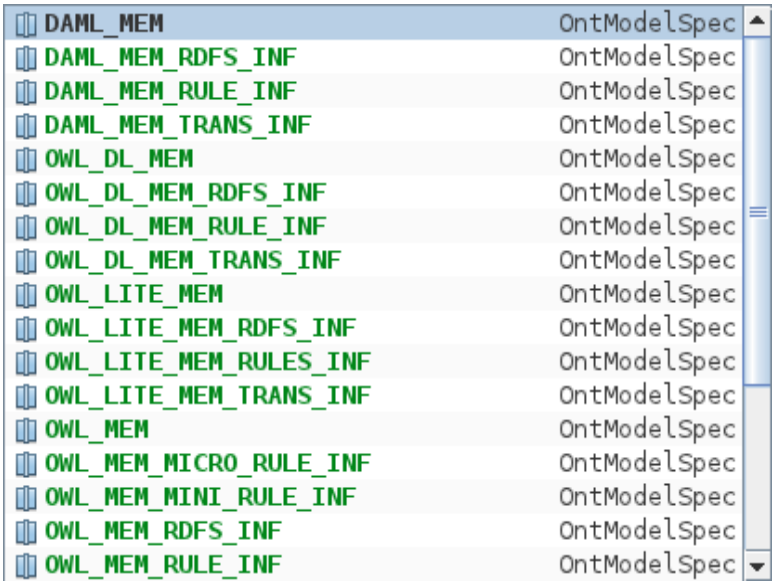


Figura 5.3: Vista de algunas opciones a seleccionar para la creación de modelos ontológicos en JENA.

```
33          String strModel = "jdbc:mysql://localhost/SOFORG";
34          String strModelUser = "root";
35          String strModelPwd = "";
36
37          // conexión con el modelo persistente
38          IDBConnection conModel = new DBConnection(strModel, strModelUser, strModelPwd, "MySQL");
39
40          // modelo desde el almacenamiento
41          ModelMaker maker = ModelFactory.createModelRDBMaker(conModel);
42          ModelRDB model = (ModelRDB) maker.openModel("ModeloEjercicio");
43
44          //modelo utilizando como razonador OWL_MEM_RULE_INF, sobre el modelo abierto
45          OntModel modelOnt = ModelFactory
46                  .createOntologyModel(OntModelSpec.OWL_DL_MEM_RULE_INF, model);
```

Figura 5.4: Modelo de importación y creación de las ontologías en JENA.

Los modelos ontológicos bases, no tienen una construcción estándar a seguir, por lo que en esta investigación no se detallará su proceso de desarrollo, solamente se planteará algunos aspectos relevante y patrones del diseño en sí, esto se debe a las propiedades del diseño de ontologías y a lo complejo que resulta adecuarla a un modelo de ejercicio a evaluar, ya que cada ejercicio asignado puede ser muy diferente en cuando a estructuras y operaciones contenidas. Por lo tanto, se considerará para este proyecto, que los modelos ontológicos base de los modelos de ejercicio, ya se encuentran desarrollados y almacenados en memoria persistente.

La programación utilizando lenguaje Java, presenta muchos componentes que pueden representar estructuras, herramientas y objetos en sí; para este proyecto, se han identificados los más importantes y útiles en el desarrollo del proceso de evaluación, conjuntamente con la información o propiedades que deben tener para ser diferenciados de otros, dentro del código fuente. A continuación se lista los componentes respectivas propiedades:

- *Lecturas*: orden lógico (antes de, después de), variable que recibe la lectura, nivel jerárquico y su ubicación (si se encuentra anidado dentro de una estructura como if-ese, while, etc.; en el caso de if-else si se encuentra en el if o en el else).
- *Impresiones*: orden lógico, variables que son impresas, nivel jerárquico y su ubicación.
- *Operaciones*: orden lógico, variables utilizadas, operadores utilizados, a que variable se le asigno el resultado, nivel jerárquico y su ubicación.
- *Valores numéricos o booleanos*: no contiene propiedades se utiliza solo su nombre.
- *Sentencias lógicas*: variables utilizadas, operadores lógicos, si se relacionan con otro sentencia a través de AND o OR.
- *Variables*: tipo de variable, inicialización, componentes donde fue utilizada.
- *Estructura if-else*: orden lógico, sentencia utilizada, nivel jerárquico y su ubicación.
- *Estructura while*: orden lógico, sentencia utilizada, nivel jerárquico y su ubicación.

- *Estructura do-while*: orden lógico, sentencia utilizada, nivel jerárquico y su ubicación.
- *Estructura switch-case*: orden lógico, variable de comparación, valores de los case, nivel jerárquico y su ubicación.
- *Array*: orden lógico, tipo, longitud declarada, nivel jerárquico y su ubicación.
- *Métodos*: tipos de variables que acepta, si retorno o no un resultado, tipo de resultado que retorna, ubicación.
- *Clases*: tipo de clase (abstract, public, private), si hereda o no, implementa o no.
- *Atributos*: tipo de privacidad, tipo de variable, inicialización, componentes donde fue utilizada.
- *Constructores*: tipos de variables que acepta, clase de origen, tipo de privacidad.
- *Instancias*: ubicación, clase de origen, constructor al que accede.

Basado en lo anterior, se pueden identificar las clases y propiedades ontológicas a utilizar; en la figura 5.5, se presentan las clases ontológicas con sus jerarquías, que describen la composición de las ontologías bases de manera general. En dicha figura, se observa que el proceso de evaluación ontológica se ha dividido en varias clases contenedoras de los componentes que describirán la estructura del código fuente y los modelos de ejercicios en sí; la clase *general*, contiene subclases que describirán el tipo de variable (entero, cadena, etc.) y estructura (ifelse, while, etc.), de igual forma, la clase *operaciones*, contiene la información de los componentes más importante del lenguaje, la clase *ordenEstructura*, poseerá las propiedades que permitirá identificar un modelo específico que el estudiante empleo; el resto de las clases, en esta subdivisión, representan componentes en la que cada una poseerán, una subclase (*secuencia_logica*) que describirá las propiedades de la secuencia y el orden que debiera tener los componentes dentro del código fuente; la clase *estructura*, representa las estructuras que pueden ser de decisión o repetición, ya sea de tipo IF-ELSE, ELSE-IF, WHILE, DO-WHILE o FOR; la clase *expresionesArt*, presentará las clases que describen las expresiones aritméticas, ésta trabajará estrechamente relacionada con la clase *operacionesArt*; por otro lado, la clase *variables*, contendrá la información de las variables declaradas y su utilización dentro del modelo; la clase *sentencias*, contiene

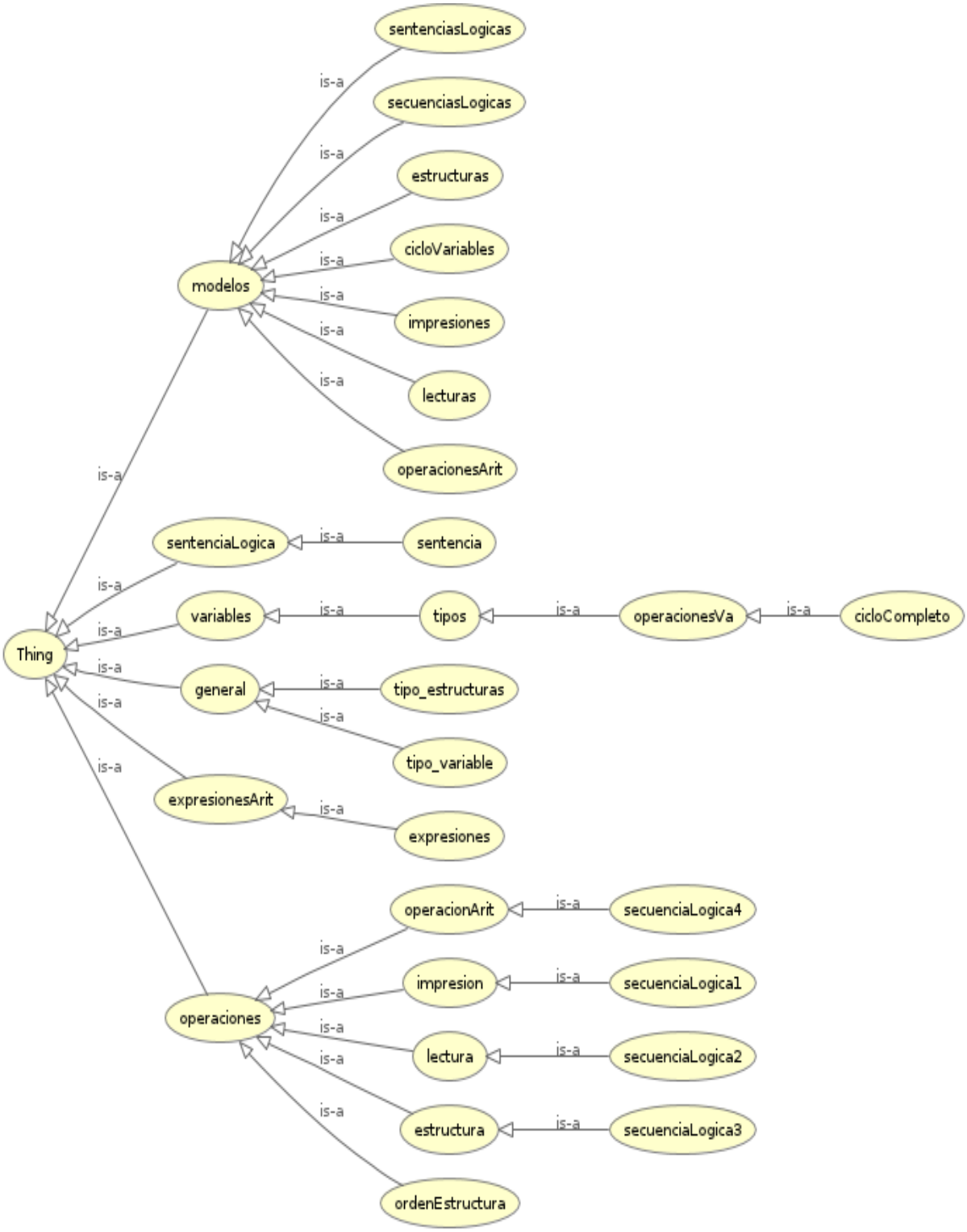


Figura 5.5: Modelo de clases de las ontologías bases para la evaluación del código.

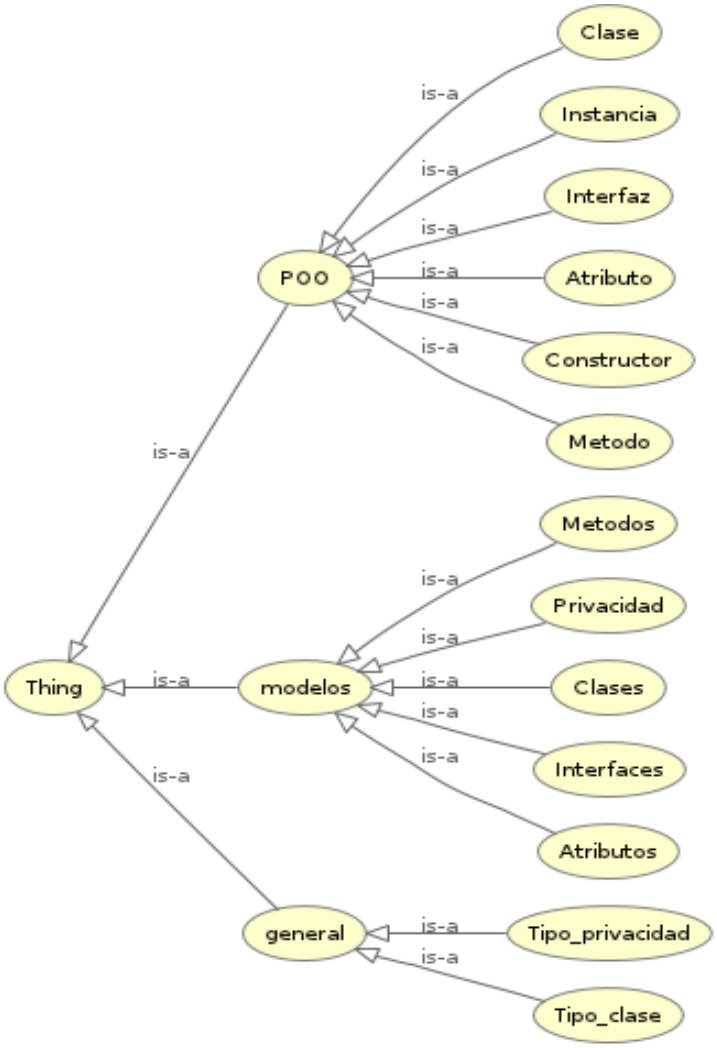


Figura 5.6: Diagrama de clases, que complementa el modelo de la figura 5.5.

las sentencias lógicas utilizadas en dichas estructuras; por otro lado, la clase *modelos*, será la encargada de proporcionar la información de los modelos disponibles para la resolución del ejercicio, las subclases contenidas en la misma, representan criterios que son utilizados para la evaluación del código, estos criterios serán detallados mejor en la sección 5.4. Por otro lado, la figura 5.6, presenta un diagrama que complementa el modelo de clases de la figura 5.5, aquí se observa una subclase *POO*, que describe los componentes de la POO; por otro lado, la clase *modelos* contiene ciertos criterios para la evaluación de POO, esta clase se complementa con la clase del diagrama anterior; y los tipos de privacidad y clases, se describen de igual forma, como subclase en la clase *general*. En la sección 5.3.3, se detallará mejor el funcionamiento de éste

modelo propuesto.

5.3.1. Propiedades ontológicas del modelo propuesto

Dentro de los lenguajes ontológicos, existen dos tipos de propiedades: *Object Property* y *Data Property*, la primera, representan propiedades que describen la relación con otras entidades de la ontología, la segunda, relaciones con valores absolutos (cadenas, enteros, fechas); por otro lado, la declaración de estas propiedades pueden estar delimitadas a un dominio y alcance; éstas a su vez, son propiedades de dichas propiedades declaradas. Los dominios y alcances hacen referencia a clases declaradas en la ontología o a otras propiedades y tipos de datos, estas propiedades son nativas de *RDFS*, cuyas respectivas etiquetas son: *rdfs:domain* y *rdfs:range*. Basado en la información requerida de los componentes listados en el punto anterior, se propone la Tabla 5.1, donde se describen las propiedades de tipo *Object Property*, utilizadas en este proyecto para la construcción de las ontologías, en la que se detalla las clases (descrita en las figura 5.5 y 5.6) a las que hacen referencias sus alcances y dominios. Como se observa, las propiedades *antesDe* y *despuesDe*, describen el orden lógico que cada componente tendrá dentro del código, de esta manera, si se identifica una lectura, éste, tendrá especificado los componentes que deben estar antes y después de dicha lectura; por otro lado, la propiedad *esDeTipo*, describirá el tipo de las variables, las estructuras, privacidad y tipos de métodos y atributos que deben ser declarados; por otro lado, *esUsadaEn*, describe los lugares donde fue empleada una determinada variable, ya sea en sentencias lógicas, operaciones aritméticas, lecturas o impresiones; la propiedad *asignaA*, es empleada en las operaciones aritméticas donde se especifica la variable que recibe el resultado de dicha operación; *imprime* y *lee*, son propiedades que especifican impresiones y lecturas respectivamente dentro del código, conjuntamente con las variables que éstas, involucran; las propiedades *divide*, *multiplica*, *resta* y *suma*, son las que representan las operaciones aritméticas, pero sus relaciones se dá entre los objetos que son del tipo de clase *expresion*; *seLeAsigno*, es una propiedad perteneciente a las variables, en la que relaciona operaciones aritméticas o valores absolutos a los mismos; de igual forma, *tieneExpresion*, es una propiedad que asocia una expresión a una operación aritmética; la propiedad *tieneValor*, relaciona una expresión con una variable o valores absolutos; por otro lado, *distintoQue*, *igualQue*, *mayorOigualQue*, *mayorQue*, *menorOigualQue* y *menorQue*,

Tabla 5.1: Propiedades de tipo *Objetct Property* identificadas en las ontologías de SOFORG.

Object Properties	rdfs:domain	rdfs:range
antesDe	*secuencia_logica*	*operaciones*
despuesDe	*secuencia_logica*	*operaciones*
esDeTipo	*variables* *estructuras* *clases* *atributo* *metodo* *interfaz*	*tipo_variable* *tipo_estructura* *tipo_privacidad* *tipo_atributo* *tipo_metodo*
asignaA	*operacionArit*	*variables*
imprime	*impresion*	*variables*
lee	*lectura*	*variables*
divide	*expresion*	*expresion*
multiplica	*expresion*	*expresion*
resta	*expresion*	*expresion*
suma	*expresion*	*expresion*
seLeAsigno	*variables*	*operacionArit* Valores absolutos
tieneExpresion	*operacionesArit*	*expresion*
tieneValor	*expresion*	*variables* Valores absolutos
distintoQue	*sentencias_logicas*	*variables* Valores absolutos
igualQue	*sentencias_logicas*	*variables* Valores absolutos
mayorOigualQue	*sentencias_logicas*	*variables* Valores absolutos
mayorQue	*sentencias_logicas*	*variables* Valores absolutos
menorOigualQue	*sentencias_logicas*	*variables* Valores absolutos
menorQue	*sentencias_logicas*	*variables* Valores absolutos

continúa...

Object Properties	rdfs:domain	rdfs:range
tieneVariableSentencia	*sentencias_logicas*	*variables* Valores absolutos
tienePadre1	*operaciones*	*estructuras* *metodo*
tienePadre2	*operaciones*	*estructuras* (if-else)
esUsadaEn	*variables*	*sentencias_logicas* *operacionArit* *lectura* *impresion*
tieneInicializacion	*variables*	*operacionArit* Valores absolutos
tieneSentenciaLogica	*estructura*	*sentencias_logicas*
tieneOperacion	*modelos*	*operaciones* *variables*
tieneComp	*modelos*	*POO*
tieneAtributo	*clase*	*atributo*
tieneMetodo	*clase*	*metodo*
tieneInstancia	*metodo*	*clase*
tieneConstructor	*clase*	*constructor*
tieneParametros	*metodo* *constructor*	*variables*
esDeLaClase	*instancia*	*clase*
heredaA	*clase*	*clase*
implementaA	*clase*	*clase*
utilizaA	*clase*	*clase*

representan los operadores de comparación en las declaraciones lógicas; estas propiedades se usan en conjunto con *tieneVariableSentencia*, para describir los mismos; la propiedad *tienePadre1*, es utilizada para especificar la ubicación de un componente, cuando ésta se encuentra anidada en alguna estructura; sin embargo, para el caso de especificar cuando un componente se encuentra del lado del ELSE, cuando se declara una estructura de decisión completa, se utiliza la propiedad *tienePadre2*; de igual forma, *tieneInicializacion*, es utilizada en aquellas variables que se le asigna un valor u operación cuando son declaradas; la propiedad *tieneSentenciaLogica*, representa la relación de una estructura, ya sea de repetición o de decisión, con una sentencia lógica existente; *tieneOperaciones* y *tieneComp*, son las propiedades que serán empleadas para especificar que un individuo de la clase *modelos*, posee los componentes que el estudiante plasmo en su código, de igual forma, incluye la información

de las clases y otras entidades pertenecientes a POO; el resto de las propiedades son descritas para el manejo de la POO, donde se especifica los parámetros de los métodos y constructores, si una clase hereda o no, si se hace una implementación o simplemente, se utiliza una clase dentro de otra; aparte de describir, los atributos, métodos, instancias y constructores que posee un código fuente. Cabe destacar que algunas de las propiedades mencionadas, pueden presentar ciertas características o restricciones que le permitan ser más eficiente a la hora de ejecutarse el razonador ontológico; características como la de *Inverse Property*, para el caso de *seLeAsigno* que es el inverso de *asignaA*, describe que una operación tiene una asignación a una determinada variable, al incluirse ésto, la propiedad *seLeAsigno* queda configurada con la operación y la variable que recibió dicho resultado, esto ocurre de forma viceversa; de igual forma, hay ciertas propiedades que son declaradas con algunas restricciones como *Functional Property*, ésta indicará que no puede ser utilizada dicha propiedad más de una vez, en un mismo objeto, para nuestro caso las propiedades *tienePadre1*, *tienePadre2*, *tieneInicializacion*, *tieneSentenciaLocal* son declaradas de este tipo.

Por otro lado, el número del nivel de anidamiento será definido a través de la propiedad de tipo *Data Property*, *nivelAnidamiento*; esta propiedad describe el nivel de anidamiento que va desde 1 en adelante, es decir, si una estructura u operación se encuentra contenida dentro de otra estructura, la primera tendrá nivel de anidamiento 2 y la segunda, 1. En los siguientes puntos se describirá mejor, la composición de los modelos ontológicos base y las instancias de las mismas, además de detallarse sus funciones y relaciones con el resto del sistema.

5.3.2. Instancias de los componentes

Una instancia ontológica representa un objeto de clases, tendrán las propiedades suficiente para ser identificadas y categorizadas dentro de una ontología. Como fue explicado, las instancias o individuos, en este proyecto, representarán componentes de programación del lenguaje Java y serán creadas con la ayuda del análisis estático llevado a cabo en la clase *Analizador_Ind*, donde ésta hace uso de los métodos de la clase *evaluadorOWL*. En la Tabla 5.2, se presenta las instancias ontológicas conjuntamente con las propiedades que pueden contener una vez sean creadas; todo esto basado en los componentes identificados de la sección 5.3 y el punto 5.3.1, donde se

Tabla 5.2: Instancias con las posibles propiedades ontológicas que las describen.

Instancia	Propiedades
*lectura**	*antesDe*, *despuesDe*, *tienePadre1*, *tienePadre2*, *nivelAnidamiento*, *lee*
*impresion**	*antesDe*, *despuesDe*, *tienePadre1*, *tienePadre2*, *nivelAnidamiento*, *imprime*
*operacion**	*antesDe*, *despuesDe*, *tienePadre1*, *tienePadre2*, *nivelAnidamiento*, *asignaA*, *tieneExpresion*
*expresion**	*divide*, *multiplica*, *resta*, *suma*, *tieneValor*
*sentencia_logica**	*distintoQue*, *igualQue*, *mayorOigualQue*, *meyorQue*, *menorOigualQue*, *menorQue*, *tieneVariableSentencia*
Variable**	*esDeTipo*, *esUsadaEn*, *seLeAsigno*, *tieneInicializacion*
Estructuras***	*antesDe*, *despuesDe*, *tienePadre1*, *tienePadre2*, *nivelAnidamiento*, *esDeTipo*
Métodos**	*tieneParametros*, *esDeTipo*
Clases**	*tieneAtributo*, *tieneMetodo*, *tieneConstructor*, *heredaA*, *implementaA, utilizaA, esDeTipo*
Atributos**	*esDeTipo*
*constructor**	*tieneParametros*
*instancia**	*esDeLaClase*
Valores absolutos**	no tienen propiedades
Modelos	*tieneOperacion*, *tieneComp*

* utilizará el nombre del componente, más una numeración secuencial, ejemplo: lectura1, lectura2...

** utilizará el mismo nombre con que fue declarado en el código fuente.

*** utilizará el nombre de la estructura, más una numeración secuencial, ejemplo: ifelse1, ifelse2...

presentaron las propiedades ontológicas. Cada instancia hace referencia a una clase y a las propiedades que se encuentran declaradas dentro de la ontología base, estas propiedades les hacen relacionarse con las demás instancias creadas; de esta manera, las instancias creadas tendrán propiedades que las asociarán como miembros de una clase, ya sea porque haya sido inferido o declarado explicitamente; también tendrán otras propiedades que lo relacionarán con otros individuos creados en tiempo de ejecución.

Por otro lado, los nombre de las instancias describirán el tipo de componente al que representa conjuntamente con un número de forma secuencial o, simplemente, utilizarán los nombres de los identificadores como fueron declarados en el código fuente. Para el caso de la instancia de los *Modelos*, se creará sólo, una instancia del mismo por cada análisis estático realizado, que puede representará un ejercicio desarrollado, sin importar el número de archivos físicos introducidos; el nombre que recibirá dicha instancia, será el nombre del estudiante más el código del ejercicio, ejemplo: *Maria-Fernandez_E0003*; cabe destacar que, los nombres utilizados para crear las instancias dentro de la ontología, no tienen ningún tipo de utilidad en el funcionamiento del mismo; sin embargo, se recomienda utilizar nombres que puedan ser identificados por el programador, en la que a simple vista se conozca de que tipo de componente se trata; ésto, aunque dicho proceso sea transparente para el usuario final, ya que todo se realizará de manera interna sin presentar detalles de la ontología.

En la figura 5.7, se observa una representación en Protégé, de una instancia declarada y sus propiedades que lo identifican dentro de la ontología, estas propiedades se encuentran declaradas en la parte derecha de la figura; se observa que dicha instancia, es denominada *ifelse1*, se deduce a simple vista, que se trata de una estructura de decisión IF-ELSE, por lo que explicitamente, se declara como miembro de la clase *operaciones*, como se observa en dicha figura; por otro lado, se observa las propiedades *antesDe* y *despuesDe* haciendo referencia a otras instancias ya creada y previamente identificadas, de igual forma, se detalla la sentencia lógica que presenta y el nivel de anidamiento que posee, en este caso es de 1. El código fuente para la creación de esta instancia en JENA, conjuntamente con sus propiedades, puede ser visualizado en la figura 5.8, donde se observa en la línea 54 que, en primera instancia, se crea el individuo *ifelse1* con su respectiva declaración de que es miembro de la clase *operaciones*, esto se realiza, creando una referencia de la clase que será alojada en una variable de tipo *OntClass*, esto se observa en la línea 51, esta clase se encuentra alojada dentro

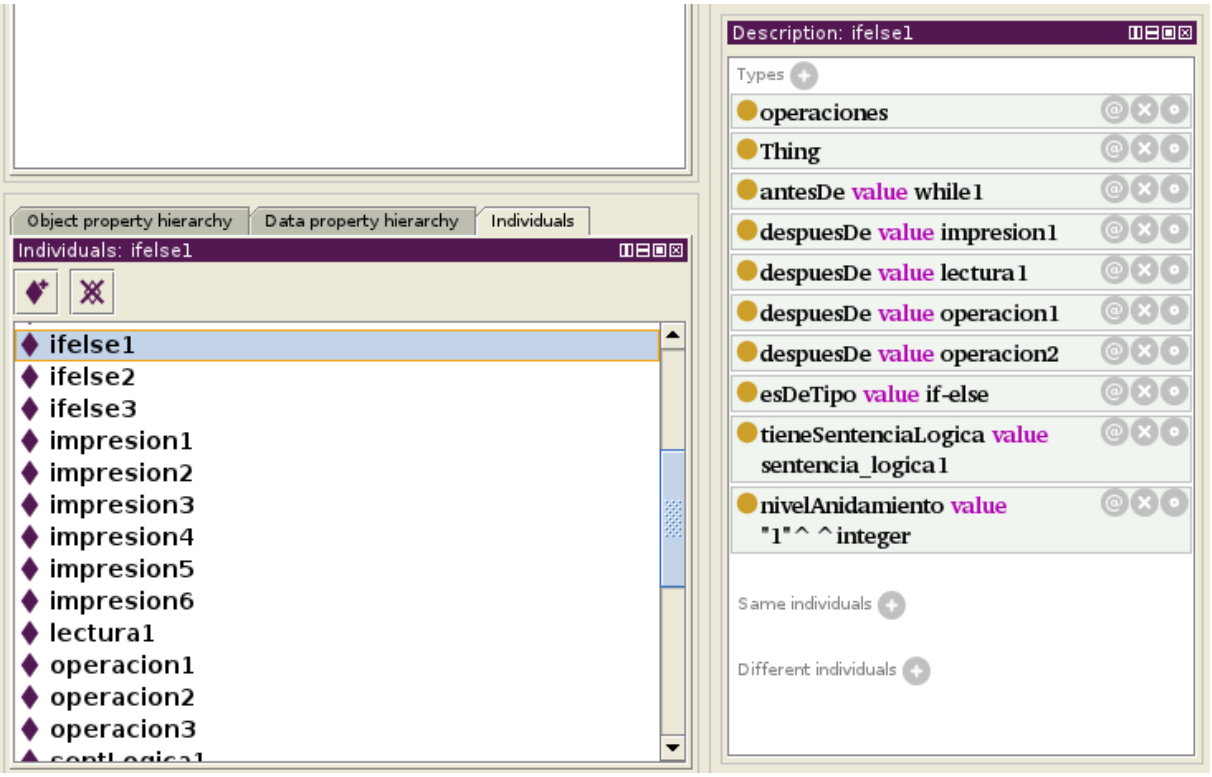


Figura 5.7: Vista de la instancia de una estructura IF-ELSE con sus propiedades en Protégé.

del modelo *modelOnt* que representa la ontología base que fue cargada previamente; de igual forma, se debe hacer referencia a las propiedades para ser creadas, ésto, a través de la variable de tipo *ObjectProperty*, un ejemplo de esto se encuentra en la línea 57, donde se hace referencia a la propiedad *antesDe*; posterior a esto, se crea la propiedad en la que lo relaciona con la instancia *while1*, esto quiere decir que, dentro del código fuente, la estructura de decisión IF-ELSE referenciado con la instancia *ifelse1*, está antes de una estructura de repetición de tipo WHILE y denominada *while1*. Las demás propiedades se asignan de la misma manera; en el caso de la propiedad *nivelAnidamiento* que es de tipo *Data Object*, ésta, debe hacer referencia a través de la variable de tipo *DatatypeProperty* que ha sido declarada en la línea 74; de igual forma, se agrega la propiedad en la que se especifica un valor absoluto de 1. Dichas líneas de códigos se encontrarán en la clase *evaluadorOWL*, para la cuál, existirá un método por cada componente que se crea en el código del estudiante, como ya fue explicado en la sección 5.1.

De esta manera, se observa las relaciones que deben tener las instancias a medida

```
47          //Declaración del namespace
48          String NS = "file:/SOFORG/Ontology#tieneSentenciaLogica#";
49
50          //se localiza la clase operaciones de la onología base
51          OntClass cls = modelOnt.getOntClass(NS+"operaciones");
52
53          //se crea la instancia haciendo referencia a que es miembro de operaciones
54          Individual ind = modelOnt.createIndividual(NS+"ifelse1", cls);
55
56          //se localiza la propiedad antesDe de la ontologia base
57          ObjectProperty prop = modelOnt.getObjectProperty(NS+"antesDe");
58
59          //se crea la propiedad y se le asigna el valor de while1
60          ind.addProperty(prop, NS+"while1");
61
62          prop = modelOnt.getObjectProperty(NS+"despuesDe");
63          ind.addProperty(prop, NS+"impresion1");
64          ind.addProperty(prop, NS+"lectura1");
65          ind.addProperty(prop, NS+"operacion1");
66          ind.addProperty(prop, NS+"operacion2");
67
68          prop = modelOnt.getObjectProperty(NS+"esDeTipo");
69          ind.addProperty(prop, NS+"if-else");
70
71          prop = modelOnt.getObjectProperty("");
72          ind.addProperty(prop, NS+"sentencia_logica1");
73
74          DatatypeProperty datprop = modelOnt.getDatatypeProperty(NS+"nivelAnidamiento");
75          ind.addProperty(datprop, modelOnt.createTypedLiteral(1));
```

Figura 5.8: Creación de individuos con sus propiedades en JENA.

que se van creando, conjuntamente con sus propiedades que las asocian; sin embargo, debe existir una que las asocie todas, en la que a partir de ella, se tenga acceso a la información de las demás; la instancia denominada *Modelos* es la que se encarga de esta tarea, en la que se especifica a través de la propiedad *tieneOperaciones*, los componentes que contiene el código fuente, de igual forma, la propiedad *tieneComp*, detallará las entidades relacionadas con POO. De esta forma, dicha instancia, permitirá iniciar la identificación y comprobación del modelo de ejercicio que el estudiante utilizó, en el modelo ontológico base ya preestablecido.

Para una mejor apreciación de lo que ya se ha explicado, se considera un pequeño enunciado de problema que debe ser desarrollado por un estudiante:

Ejemplo 1: "*Elabore un programa en lenguaje Java que acepte como entrada una edad e indique si es niño, adolescente, adulto o anciano; basado en que los adolescentes son mayores de 10 y menores de 18, los adultos son*

```java
/*
 @ID: 9-736-1510
 @codigo: E0003
*/
import java.util.*;

public class NewClass {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        int edad = 0, i=0;
        System.out.println("INGRESE LA EDAD: ");
        edad = sc.nextInt();

        if (edad <= 10)
        {
            System.out.println("Es un niño");
        } else {
            if (edad <= 18) {
                System.out.println("Es un adolescente");
            } else {
                if (edad < 50) {
                    System.out.println("Es un adulto");
                } else {
                    System.out.println("Es anciano");
                }
            }
        }
      while (i <= edad) {
            System.out.println("Año: "+i);
            i = i+1;
       }
    }
}
```

Figura 5.9: Código fuente de la solución del enunciado del Ejemplo 1.

menores que 50 y los ancianos de 50 años en adelante; además imprima la secuencia desde el año 1 hasta la edad ingresada."

Pueden existir muchas soluciones para dicho enunciado, sin embargo, se considerará un sola; en la figura 5.9, se presenta el código fuente desarrollado para el enunciado descrita, en la que se observa que se utilizó como estructura de decisión, varios IF-ELSE anidados que permitirá hacer las impresiones solicitadas; por otro lado, un WHILE como estructura de repetición y varias operaciones. Si se analiza el código fuente, con el modelo ontológico propuesto, se obtendrá los principales componentes que describen dicha solución. En la figura 5.10, se observa la instancia de tipo *modelos*, que representa la solución proporcionada en la figura 5.9, en donde se presenta

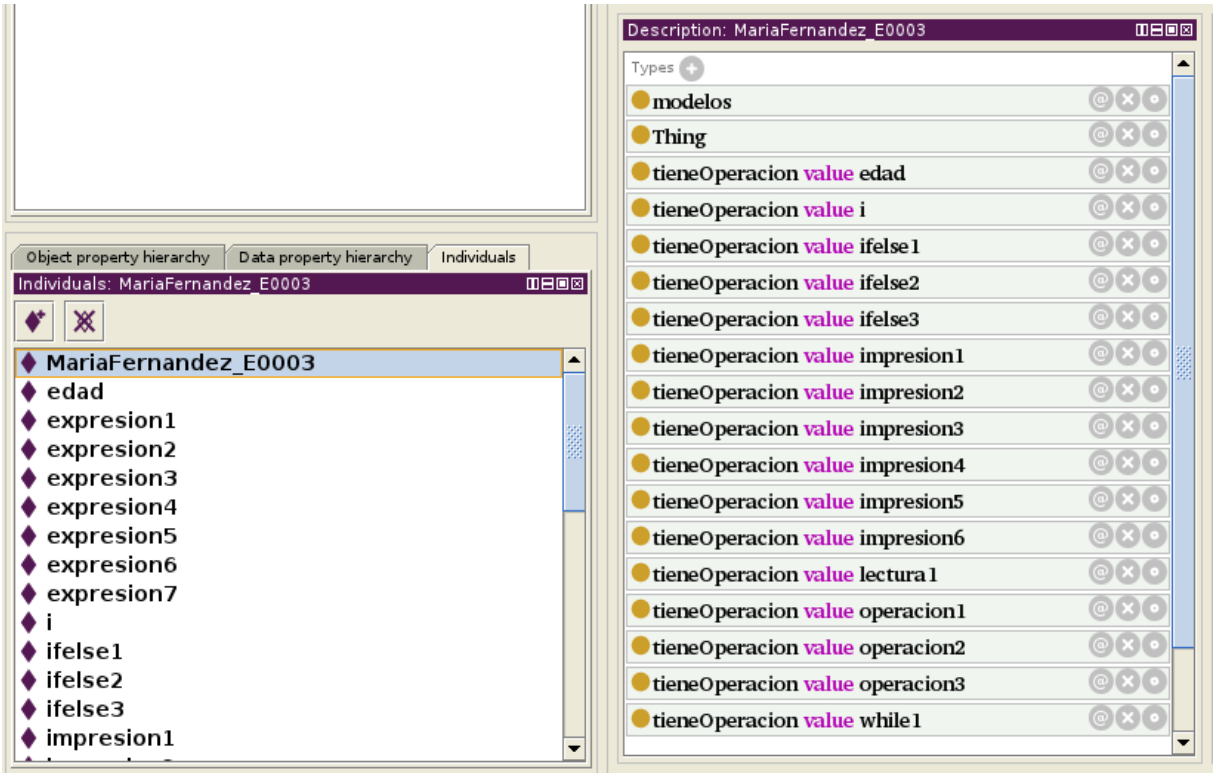


Figura 5.10: Vista de la instancia del modelo que representa el código fuente de la figura 5.9, con sus propiedades.

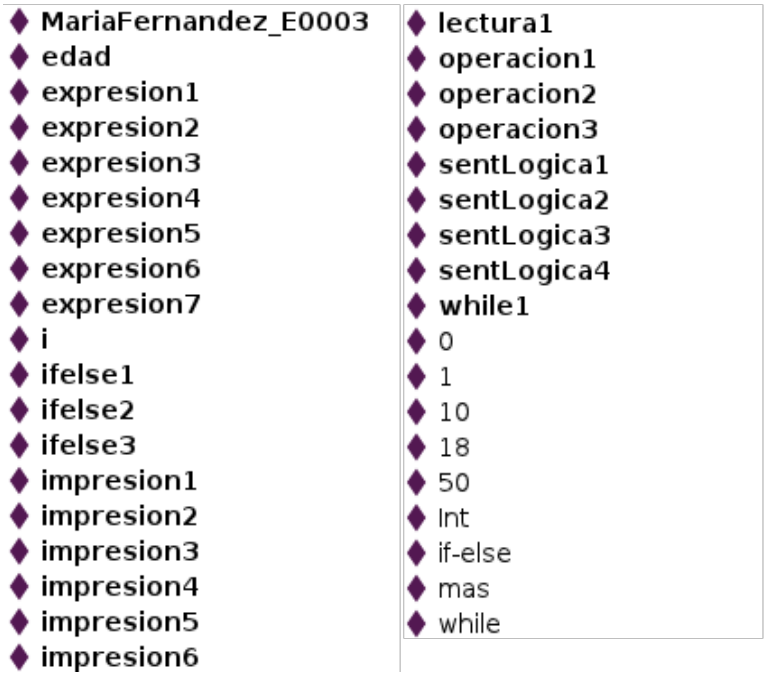


Figura 5.11: Vista de todas las instancias identificadas en el código fuente de la figura 5.9.

las propiedades de dicha instancia; aquí se observa que dicha instancia posee una propiedad que lo identifica como modelo, por otro lado, la propiedad *tieneOperacion* relaciona todos los componentes encontrados en el código fuente; de esta manera, se observa que se identificó 2 variables llamadas *edad* y *i*, respectivamente; 3 estructuras de decisión IF-ELSE, 6 impresiones, 1 lectura, 3 operaciones y 1 estructura de repetición de tipo WHILE. Si se observa el código fuente, a simple vista, se puede constatar de que los componentes fueron identificados; sin embargo, solo se observa 1 operación en el mismo y las instancias creadas fueron de 3 operaciones, esto se debe a que, por las propiedades del diseño de esta ontología, se crea una instancia de operación al hacerse una simple asignación; para este caso, las asignaciones de 0 en las declaraciones de las 2 variables, son consideradas como operaciones. De esta manera, el resto de las instancias que complementan la información del código fuente desarrollado, se visualiza en la figura 5.11, donde se observa instancias de tipo *expresion*, estas instancias representan el contenido de las operaciones aritméticas, que se van complementando para su representación; las expresiones 1 y 2, detallan la operación 1, que es la asignación de 0 a la variable *edad*, la 3 y 4, son utilizadas en la operación 2 de la misma forma que la anterior, con la variable *i*, y las expresiones 5, 6 y 7, describen la operación 3 que representa la suma de $i + 1$, la cuál, es asignada a la variable *i*; por otro lado, la sentencia lógica 4 está asociada con la declaración lógica del WHILE 1 y el resto de las sentencias pertenecen a los IF-ELSE respectivamente, el resto de las instancias representan valores absolutos y los tipos de estructuras y variables identificadas.

5.3.3. Modelo ontológico base

Para el desarrollo de este módulo, es necesario crear una aplicación basada en conocimientos, que permita, a través del conocimiento almacenado, darle un significado a los parámetros de entrada; para este caso, las entradas serán las instancias de los componentes identificados, y el conocimiento almacenado, la ontología base; este último, contendrá la información de las posibles estructuras de desarrollo y las propiedades de los resultados correctos para un ejercicio determinado, en otras palabras, contendrá representado, patrones almacenados que permitirán la comparación con la solución del estudiante plasmado en su código fuente. El producto de esta comparación, será la identificación de un patrón dentro del código fuente, o que solo

exista una parte de las coincidencias o simplemente, ninguna. A la medida que haya mayores coincidencias con el modelo ontológico base, mayor será la valoración del problema resuelto por el estudiante; de esta manera, los modelos ontológicos base juegan un papel importante, en el proceso de evaluación académica. Como ya ha sido mencionado en la sección 4.6.4, no todos los enunciados de problemas pueden tener un modelo ontológico que los contengan, esto dependerá de la extensión y complejidad de los mismos y de la diversidad de formas de resolverlos, ya que se está, bajo las limitaciones de representar dichos conocimientos y de la ejecución de los mismos (explicado en la sección 5.3).

Las clases ontológicas del modelo base, fue presentado en los diagramas de la figura 5.5 y 5.6, en donde la clase *modelos*, presenta subclases que determinarán el modelo de ejercicio que el estudiante utilizó y los criterios de evaluación del mismo. La organización de los modelos dentro de la ontología, siguen un diseño jerárquico, es decir, un modelo puede ser sub-modelo de otro, esto ocurre cuando el sub-modelo tiene estructuras similares que la del modelo padre; de esta manera un modelo denominado con la clase *modelo1-2*, puede tener modelos hijos como *modelo1* y *modelo2* o bien los nombres de las clases que representan los criterios de evaluación con dicha denominación. En la figura 5.12, se observa la composición de la ontología base para el análisis del enunciado presentado en el Ejemplo 1 de la sección 5.3.2; aquí se aprecia que la ontología base presenta 2 modelos diferentes de desarrollo, se sabe que para el enunciado presentado, existen muchas otras maneras de resolverlo, sin embargo para efectos de una mejor apreciación de lo presentado, solo se contemplará 2 modelos. De esta manera, las subclases de la clase *modelo1-2*, representan los criterios que serán evaluados, como ya fue explicado anteriormente, en la que se observa como sufijo uno o dos números, cada número representa el número del modelo a la que hace referencia, como se dijo antes, existirán 2 modelos en este caso, por lo que solo se observan números del 1 al 2, representando dichos modelos; por ejemplo, la clase *cicloVariable1-1*, representa la evaluación del ciclo de vida de la variable de *tipo1* para el modelo 1; en el caso de la clase *cicloVariable1-2-2*, esta contendrá las propiedades que evaluarán el ciclo de vida de la variable *tipo2* para los modelos 1 y 2, es decir, esta misma clase es pertinente para ambos modelos de desarrollo por compartir las mismas características; por otro lado, la clase *impresion_ifelse_1*, evalúa la impresión específica, en la estructura de decisión representada en la clase *if-else1*; la clase *secuencia_logica_1*, contendrá la información de las secuencias lógicas de todos los componentes del modelo 1, en la que se hará una sola evaluación en ge-

Figura 5.12: Modelo ontológico base para el enunciado del Ejemplo 1.

neral de este criterio; por otro lado, la clase *operaciones*, posee la mayoría de los componentes involucrados en los modelos, se observa que en algunas clases que representan los componentes, se encuentran subclases de tipo *secuencia_logica*, esto se debe a que por efectos de las propiedades nativas en el trabajo con ontologías, es conveniente la declaración de este tipo, ésta indica que la clase de la secuencia lógica, contendrá todas las propiedades heredadas a la cuál representa; las clases *ordenestr1* y *ordenestr2* son de tipo *ordenEstructura* y como ya se mencionó, éstas contienen las propiedades que identificarán los patrones del modelo base, de esta manera, la clase *modelo1-2*, poseerá una relación con dichas clases para inferir el tipo de modelo presentado, las relaciones existentes serán del tipo de propiedad *tieneOperacion*; cuando los individuos son identificados como miembro de una clase como *ordenestr1*, la clase *model1-2*, verificará si su instancia es la que se esperaba a través de la propiedad ya mencionada. Por otro lado, la clase *variables*, contiene especificada 2 tipos de variables representadas con las clases *tipo1* y *tipo2*, atendiendo al enunciado, la primera es la que debe leer la edad y la segunda, el que sirve de control en la impresión de las edades; se observa también, que existen subclases que describen

las operaciones que las variables deben realizar, para el caso de *tipo1*, se observa dos subclases *operacionVa1-1*, que representa el comportamiento de dicha variable en el modelo 1 y *operacionVa1-2*, el comportamiento en el modelo 2; de esta manera, se concluye el ciclo de vida de las variables, donde las clases *completo1-1*, *completo1-2* y *completo2-1*, representan el total de las características que las variables deben tener para un modelo determinado; en estas clases, se involucran propiedades como *esUsadaEn*, *seLeAsigno*, *tieneInicializacion*, *esDeTipo*. Cabe destacar que, las representaciones de las estructuras de repetición y de decisión en el modelo ontológico, serán las de las propiedades del IF-ELSE y WHILE únicamente, ya que con cualquiera de las dos, se puede representar las demás estructuras (DO-WHILE, FOR, SWITCH), como fue explicado en el punto 5.2), sin embargo, las instancias creadas de las mismas, mantendrán el mismo nombre de su estructura original.

La creación de las instancias haciendo referencia al modelo ontológico base, permitirá manejar una gran ontología en la que se aplicará un razonamiento ontológico. En la figura 5.13, se presenta el análisis final en protégé, del código fuente de la figura 5.9, para el enunciado presentado del Ejemplo 1. La instancia *MariaFernandez_E0003*, como fue explicado en el punto anterior, representa la instancia que inicia las comparaciones internas de la ontolgoía, ya que será la que determine la membresía en las clases de los criterios; en el cuadro resaltado negro, de la figura presentada, se observa las clases a las que pertenece dicha instancia, estas han sido inferidas después de haberse ejecutado un razonador ontológico e indicará el nivel de coincidencia que tienen, con el patrón correcto del modelo; aquí se observa que la base ontológica ubicó a la instancia de modelo, en el modelo 1 de dicha estructura, esto se debe aque las clases a la que dicha instancia es miembro, pertenecen al modelo 1. De esta manera, la clase *cicloVariable1-1* indica que el ciclo de vida de la variable *tipo1* es correcto, este representa la variable edad en el código fuente; de esta manera, se observa que la solución proporcionada, cumple con todos los criterios evaluados, en el caso de, *impresion_while_1-2*, no se encuentra dentro del listado del recuadro resaltado, debido a que se sobre entiende que ya se es miembro del mismo, ésto, debido a que la clase *while_1-2*, si se encuentra dentro de la lista y es subclase de *impresion_while_1-2*. lo mismo ocurre para los demás casos en la que existen subclases dentro de estos criterios; ésto ocurre debido a que muchas de las propiedades que manejan dichas clases, están compartidas con otras clases dentro del modelo, por lo que la ontología infiere que son subclases de las mismas, sin embargo, mantiene la funcionalidad del diseño original.

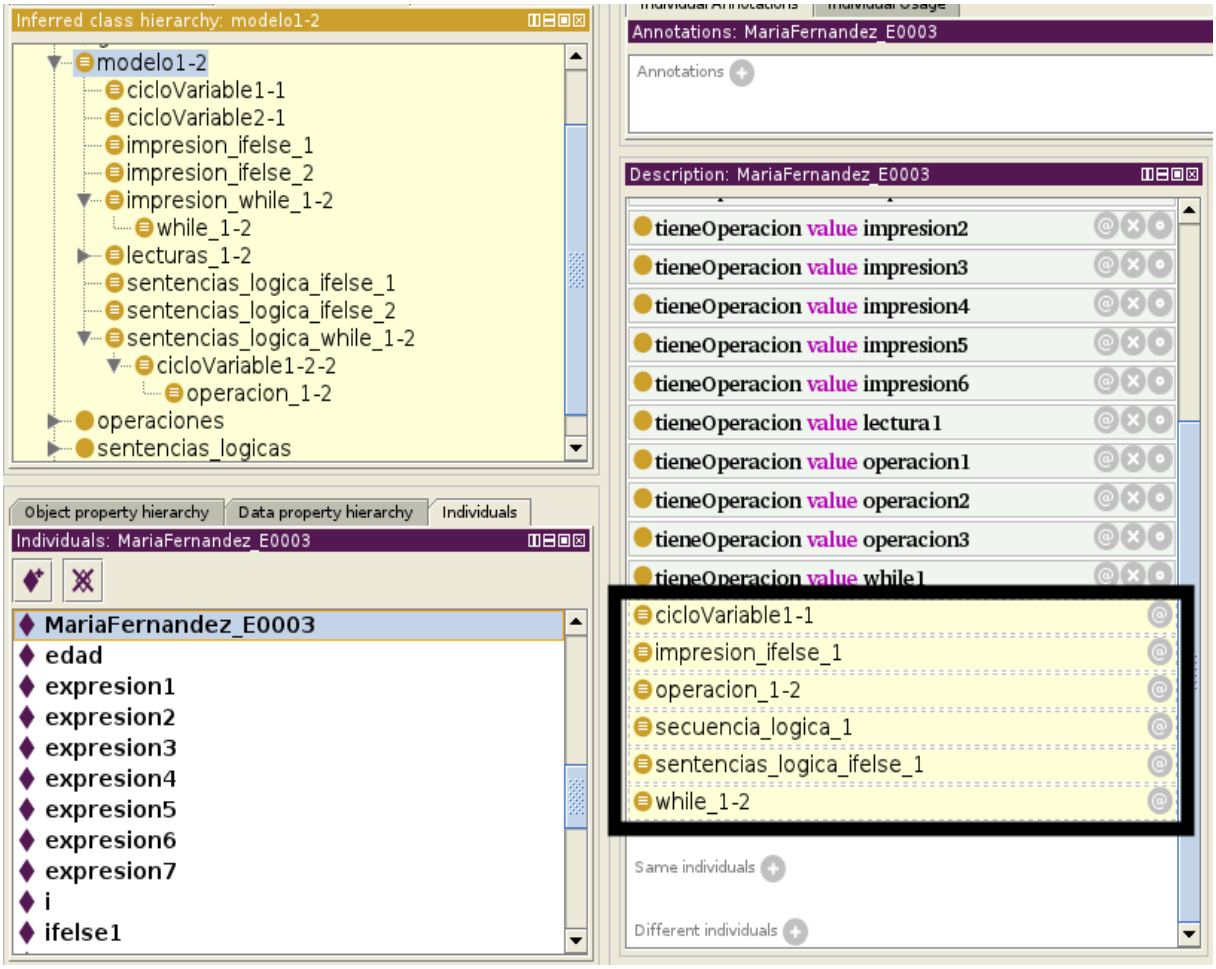


Figura 5.13: Vista en protégé, de la evaluación ontológica del código fuente.

5.4. Criterios de evaluación

El proceso de evaluación del código fuente debe contener criterios y ponderaciones que permita emitir un resultado cuantificado del ejercicio desarrollado por el estudiante. Dentro de la arquitectura ontológica propuesta, en la que se tiene una ontología base y el conjunto de instancias, la ontología base tendrá dichos criterios específicamente en la clase de tipo *modelos*, como ya fue dicho anteriormente. A continuación se describen los criterios de evaluación propuestos para la arquitectura ontológica;

- **Secuencia lógica:** este criterio evalúa la ubicación lógica de los componentes dentro del código fuente, por ejemplo: utilizar una variable en alguna estructura

lógica, después de haber leído un valor. Este criterio puede contener varias sub-clases que profundicen dicha evaluación.

- **Estructuras:** evalúan las estructuras especificadas en el código fuente, en la que se comprueba la existencia de las mismas, estas pueden ser: IF-ELSE, WHILE, FOR, DO-WHILE, SWITCH; y existirá un criterio por cada estructura declarada.

- **Sentencias lógicas:** criterio que toma en cuenta las sentencias lógicas, ya sea para las estructuras de decisión o de repetición, estas sentencias deben contener las propiedades definidas para considerarse correctas.

- **Ciclo de las variables:** esta implica la idea de que una variable tenga un ciclo de vida, es decir que cumpla con las operaciones para la cuál fue creada, este criterio puede tener sub-clases que permita detallar mejor la evaluación.

- **Impresiones:** evalúa las impresiones realizadas en el código fuente, este criterio toma en cuenta las variables que han sido impresas y el lugar donde han sido declaradas; puede contener sub-clases para una mejor profundización del mismo.

- **Lecturas:** de igual forma, evalúa las lecturas en la que se hace referencia a una variable de un tipo específico, puede obtenerse otras sub-clases del mismo.

- **Operaciones aritméticas:** serán evaluadas de igual forma, en la que se tomará en cuenta la ubicación operaciones en sí, y las variables y valores absolutos involucrados en el mismo. Para cada operación declarada pudiera existir un criterio a evaluar.

- **Declaración de clases:** este criterio evalúa las clases que han sido declarada tomando en cuenta el tipo de declaración, si hereda o no, o si implementa una interfaz; estos aspectos pudieran ser divididos en varios criterios, que serán representado con sus respectivas clases en la ontología.

- **Declaración de métodos:** considera los métodos declarados que debieran tener su relación con las clases a las que pertenecen, éstas, debieran especificar el tipo de método, tipo de valor que devuelve y los parámetros de entrada correctos para que se cumpla con dicho criterio.

Tabla 5.3: Posible ponderación de los criterios para el modelo 1, descrito en la figura 5.12.

Criterios	Valoración
cicloVariable1-1	5
cicloVariable1-2-2	5
impresion_ifelse_1	5
impresion_while_1-2	5
lecturas_1-2	5
operacion_1-2	15
secuencia_logica_1	15
sentencias_logica_ifelse_1	20
sentencias_logica_while_1-2	15
while_1-2	10
Total	100

- **Declaración de atributos:** los atributos serán evaluados como criterios ya que se tienen que especificar como tal, en vez de ser utilizadas como variables internas; también evalúa el tipo de declaración y su utilización.
- **Interfaces:** este criterio busca comprobar de que se haya implementado correctamente una interfaz definida. Cada interfaz presentará su propio criterio de evaluación.
- **Privacidad:** se evaluará la privacidad, como criterio, ya que puede existir declaraciones erróneas de los componentes y obtenerse un mal funcionamiento.

Los criterios de evaluación deben contener ponderaciones que le permita emitir un resultado; ésto consiste en darle un valor o puntuación a cada criterio de evaluación presentado en un modelo específico, esta valoración quedará en consideración del docente y dependerá de la complejidad del ejercicio y de los componentes evaluados. En la Tabla 5.3, se presenta un ejemplo de una posible ponderación basada en los criterios de la clase *modelo1-2*, que fue presentada en la figura 5.12; para la tabla presentada, solo se toma en consideración los criterios del modelo 1, en la que dichos criterios son denominados con el nombre de la clase a la que hace referencia; de igual forma, se considera 100 como puntuación total del mismo.

Capítulo 6

Ambiente de trabajo de SOFORG

Una vez presentado los detalles del diseño y la arquitectura de la herramienta, se describe el ambiente de trabajo en el presente capítulo; donde se detallará las ventanas principales y su utilización, además las funcionalidades de la herramientas y sus características más importantes. El desarrollo de este capítulo ha sido dividido en tres secciones, la primera sección representa la descripción de la ventana principal conjuntamente con los menús y barra de herramientas; la segunda, representa la sección del módulo de análisis estático, se describe de igual forma las funcionalidades y opciones permitidas para el mismo; la tercera sección, el módulo de generador de grupos.

Cabe destacar que la construcción de SOFORG contempla un prototipo funcional, como fue explicado en la sección 1.6; por lo que se limitará a la descripción de las funciones principales, que cumplan con los objetivos previsto del proyecto.

6.1. Ventana principal

Como ya fue mencionado, SOFORG es una herramienta de escritorio donde los profesores tendrán acceso a través de su propia sesión de usuario. La figura 6.1, presenta un vistazo del inicio de sesión por parte del profesor que incluye la petición de los datos comunes; sin embargo, se contemplará la utilización de diferentes tipos de sesiones en la que limitará el acceso a algunas funciones, como por ejemplo: la

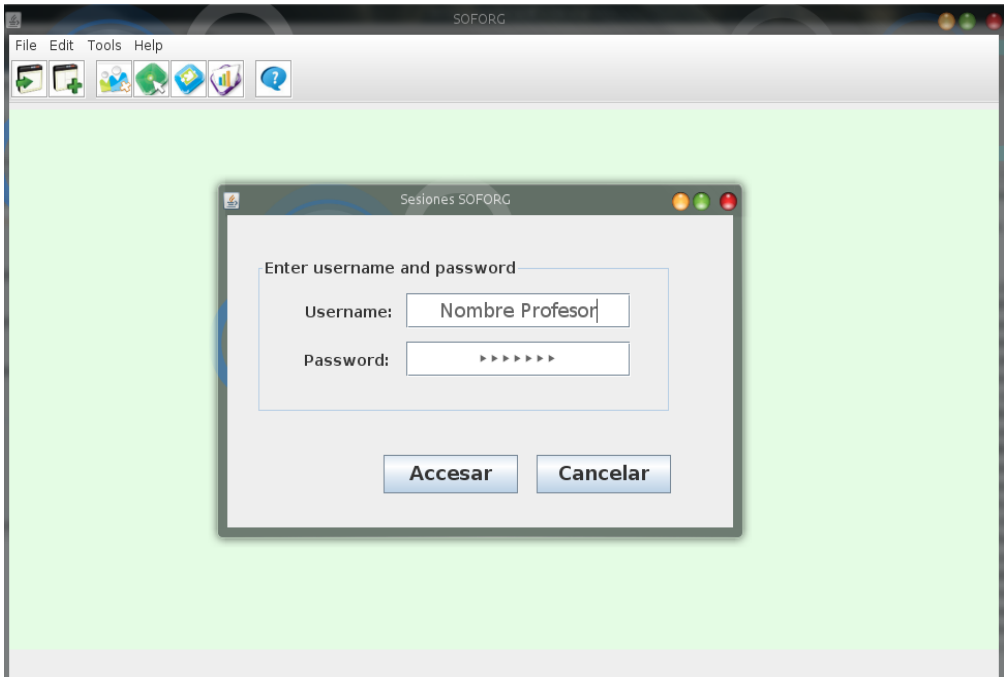


Figura 6.1: Ventana principal de inicio de sesión.

sesión de profesores tendrá únicamente acceso a las funciones principales y no a la evaluación de los reportes e informes, como es el caso de la sesión de administrador. Aparte de los beneficios clásicos de utilizar sesiones, SOFORG implementa sesiones, específicamente en el caso de los profesores, para informar al sistema de la identidad de los operadores y de las actividades realizadas.

SOFORG presenta un entorno amigable con fondo verde en la que se despliega las ventanas, ya sea en forma de escritorio o ventanas flotantes como es el caso de la ventana de inicio de sesión; por otro lado, se observa la presencia de una barra de menús y una barra de herramienta. En la barra de herramientas se contempla iconos para cada botón de las opciones, de igual forma en los menús desplegables de la barra de menús; todo esto es observado en la figura 6.2. Ésta figura presenta 4 vistas en la que se visualiza las opciones contenidas en la barra de menús, se observa que cada icono representado en la barra de herramienta, invocará la misma función que el de los menús desplegables; por otro lado, cada opción presenta combinaciones de teclas para su acceso en la herramienta. En la vista 2 de la figura 6.2, se observa el menú *File* desplegado, en este se visualiza las opciones de *New static analysis*, *New exercise model* y *Exit*; la primera opción inicia un nuevo análisis estático en el que el profesor ingresará el código fuente de los estudiantes, el segundo, presenta la he-

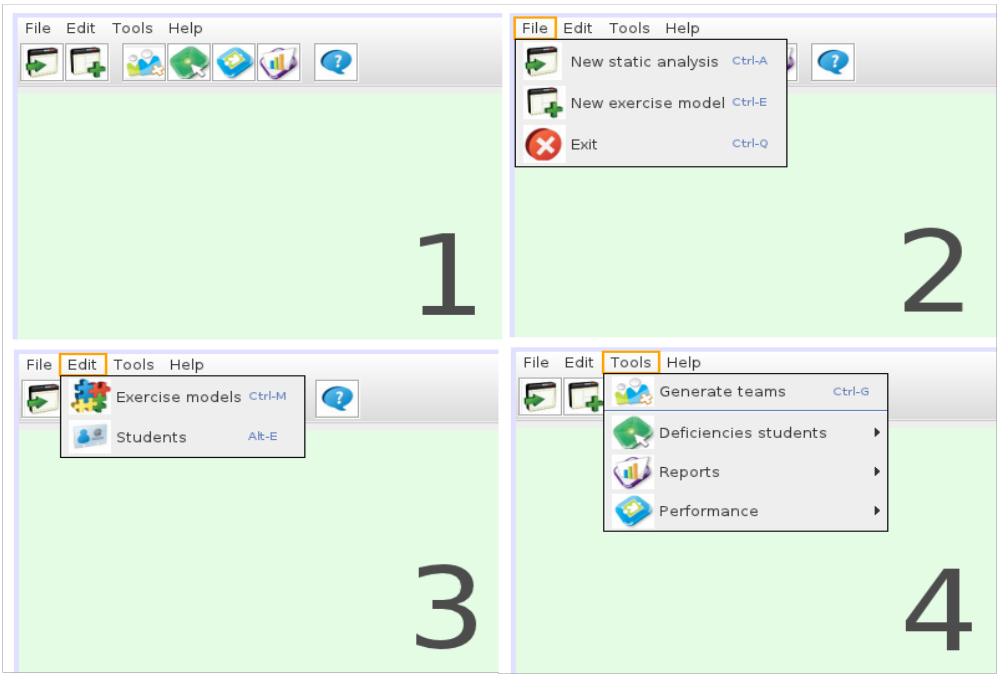


Figura 6.2: Vistas del menú y barra de herramientas.

rramienta para ingresar un nuevo modelo de ejercicio al sistema, para el ingreso de un nuevo modelo, es necesario especificar cierta información que permita hacer uso efectivo del mismo, esta información puede ser: nombre, definición del problema, temas académicos que debe cubrir y en caso de que exista, la referencia a una ontología base que permitirá la evaluación del mismo, de igual forma, se especificará su ponderación a cada componente del problema desarrollado. La tercera opción, permite salir del sistema. En la vista 3, las opciones *Exercise models* y *Students* representan el menú desplegable *Edit*, donde el primero presenta una ventana, en la que se puede editar los modelos de los ejercicios ya ingresados, esta opción es útil debido a que la creación de modelos de ejercicios implica un proceso iterativo, ya que muchas de las características especificadas al inicio, como la ponderación, son evaluadas a medida que se vayan utilizando con los análisis estáticos de los estudiantes; de esta manera, se logra configuraciones más optimas. De igual forma, la segunda opción *Students*, presenta muchas funcionalidades en la gestión de la información de los estudiantes, desde agregar nuevos, hasta modificar su información personal. Por último, en la vista 4, el menú desplegado *Tools* representa las herramientas que complementan a SOFORG; se observa la opción *Generate teams*, esta opción es la encargada de generar los grupos formados basados en opciones que deben ser especificadas; *Deficiencies*

students, *Reports* y *Performance*, son menús para la presentación de informes. En el primero, se puede observar informes de deficiencias por estudiantes, ya sea por año, curso, profesor o semestre; de igual forma, estas deficiencias son extraídas de las buenas prácticas que son sugeridas por el sistema. La segunda opción, puede presentar evaluaciones y métricas de los estudiantes, número de ejercicios aplicados en el año, cuál de los ejercicios ha tenido mayor obtención de buenas calificaciones, y de igual forma, cuáles han sido menos efectivos; esto permitirá obtener una retroalimentación de lo presentado y así mejorar la configuración de las opciones (proceso iterativo como se explico anteriormente). La tercera *Performance*, despliega los informes relacionados con el rendimiento del estudiante y la efectividad de los profesores. Éstos son presentados por año, curso, carrera, profesor y semestre; útil para una mejor visualización retrospectiva de la información que el sistema va capturando a lo largo que se realiza las operaciones en el mismo.

En las siguientes secciones, se detalla dos de las opciones presentadas anteriormente, que representan las principales funcionalidades del prototipo funcional presentado; el analizador estático de código fuente y el generador de grupos, serán descritos.

6.2. Analizador estático de código fuente

El analizador estático es uno de los módulos desarrollado para el sistema SOFORG, como fue ya explicado, éste, analiza el código fuente del estudiante y extrae información del mismo, en forma de métricas que serán almacenadas. En la figura 6.3, se observa la vista de la ventana que es abierta al seleccionar la opción *New static analysis*; tiene la función de ejecutar el análisis estático al seleccionar un archivo con extensión *java*, esta conformada por un campo de texto en la que se debe especificar la dirección del código del estudiante. Existe de igual forma, un botón con etiqueta, *"..."* que funciona como apoyo en la búsqueda y selección del código del estudiante, el botón *"Analyze"*, ejecuta una vez seleccionado el código fuente, el análisis del mismo y el botón *"New"*, hace una limpieza de algún análisis en curso y presenta un nuevo análisis para ser ejecutado; en la misma ventana, se observa en el panel *Information*, un área de texto en el que presentará, en caso de encontrar algún error, la información pertinente. Estos errores, en su mayoría, tienen que ver con la parte sintáctica del análisis. En la figura 6.4, se puede visualizar la ventana de abrir desplegada. Cabe

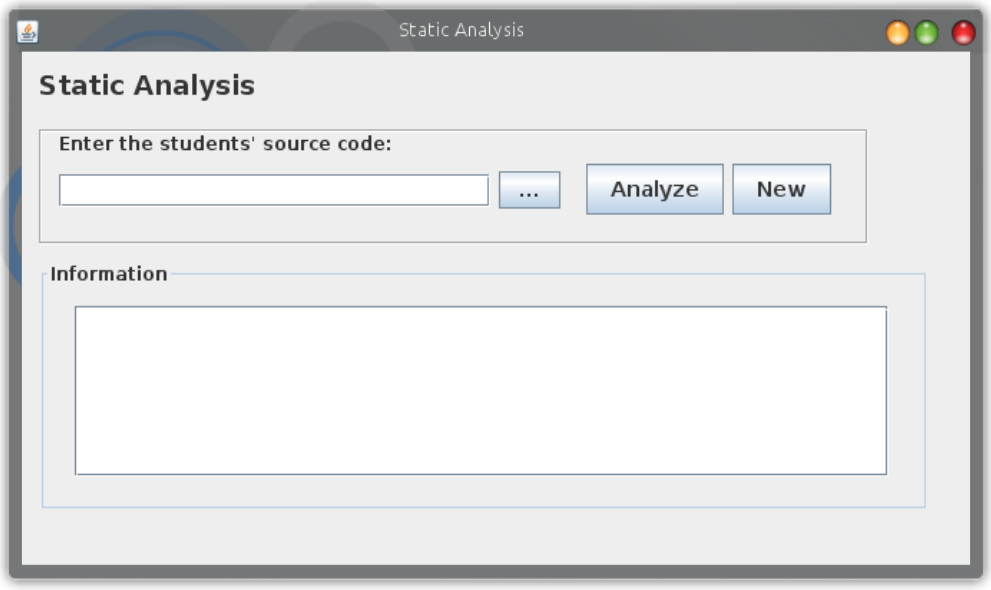


Figura 6.3: Vista de la ventana de nuevo análisis estático.

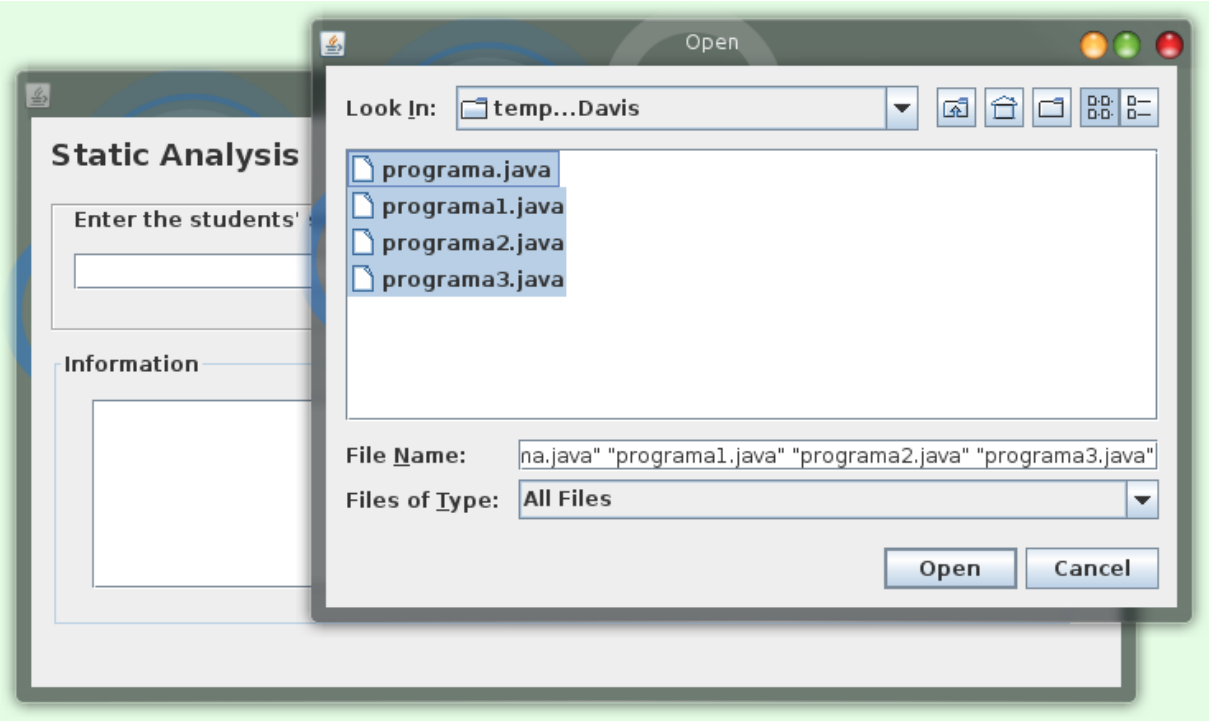


Figura 6.4: Vista de la ventana abrir archivos con extensión java.

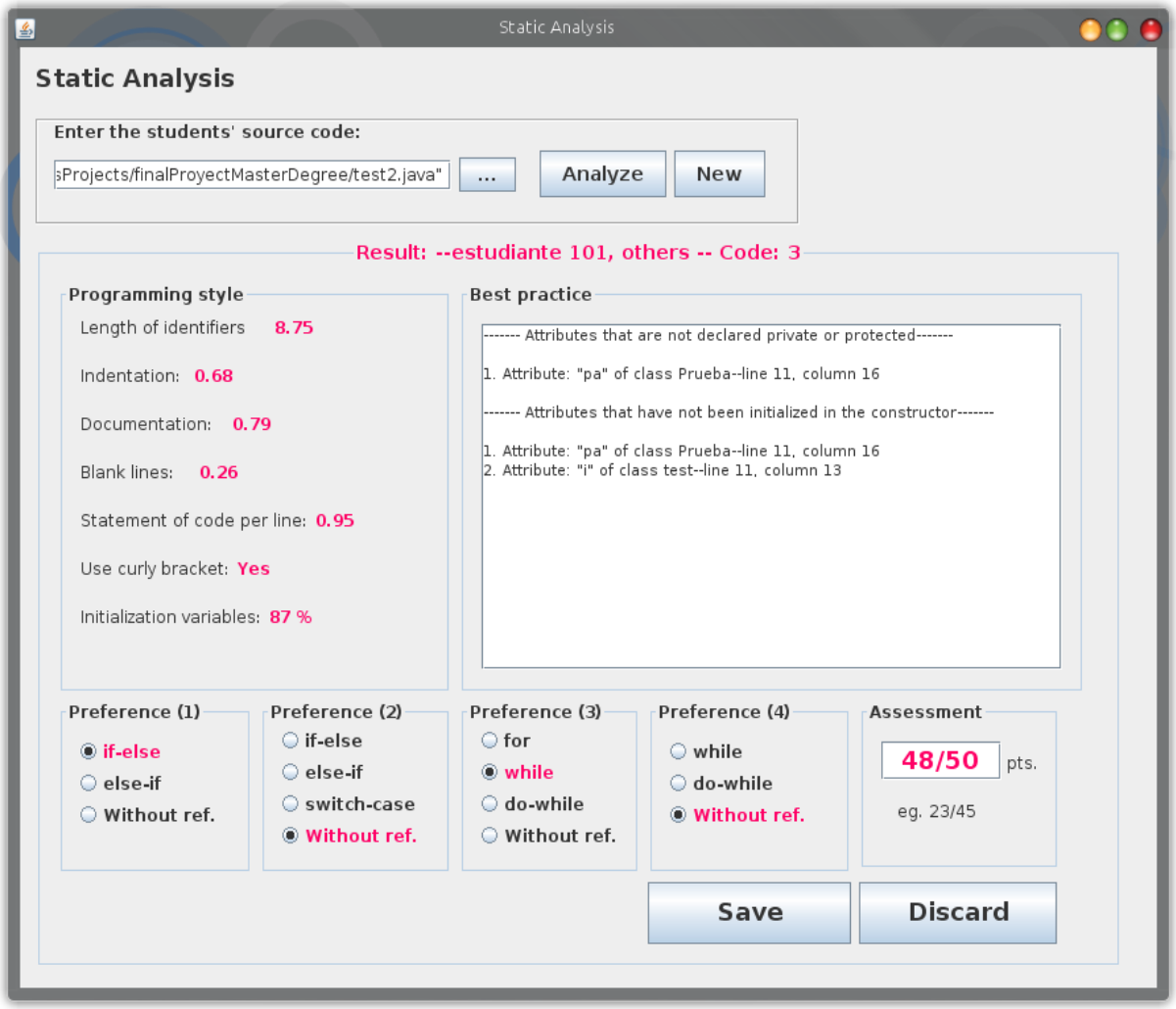


Figura 6.5: Vista del resultado de un análisis estático.

destacar que la herramienta SOFORG, podrá analizar más de un archivo con extensión *java*, como se visualiza en dicha figura; esto permitirá analizar aquellos programas en la que se declaran clases en archivos separados, manteniendo la misma integridad del análisis como si fuera un mismo archivo; sin embargo, SOFORG no analizará más de un programa a la vez, ya que cada programa deberá tener su propia evaluación.

De esta manera, cuando el profesor selecciona la opción de nuevo análisis estático y después de indicar el programa del estudiante a evaluar, éste debe dar clic en el botón *Analyze* para ejecutar el análisis que dará como resultado una ventana con toda la información extraída; la vista de dicha ventana se observa en le figura 6.5, donde se aprecia las métricas, la evaluación, preferencias de estructuras y sugeren-

cias de buenas prácticas en la misma. El resultado del análisis está organizado en sus respectivas categorías (detallado en secciones anteriores), conjuntamente con el nombre del estudiante y el código del ejercicio. En el panel de estilos de programación (*Programming style*), se presenta el resultado basado en las métricas expuestas; las preferencias se encuentran separadas en cada panel así como se explicó, y utilizan botones de radio con cada opción, en caso de que no exista una determinada preferencia, SOFORG marcara el botón de radio *"Without ref."*; por otro lado, el panel de buenas prácticas denominado *Best practice*, se observa un cuadro de texto en la que se imprimen las buenas prácticas que los estudiantes no han aplicado, y dependiendo del caso, se especifica el objeto que al que se hace referencia, clase a la que pertenece, línea y columna en el código; y por último, el panel de evaluación (*Assessement*), en la que presenta un campo de texto y tendrá el siguiente formato *puntosObtenidos/puntosTotales*; en caso de que no apareciera la evaluación, significaría que no existe un modelo ontológico del ejercicio, por lo que el profesor debe especificar la evaluación; por lo que no se guardará un registro del análisis si no se especifica una evaluación para el problema analizado. Tanto las preferencias como la evaluación pueden ser editados antes de ser guardados, ésto se debe a que SOFORG sugiere dichos valores y queda en consideración del profesor su aceptación.

6.3. Generador de grupos

El generador de grupos es uno de los módulos más importante en SOFORG, tiene la funcionalidad de evaluar la información almacenada y emitir los respectivos grupos basados en las opciones seleccionadas. Una vez se tenga una base de los datos ya capturados en el análisis estáticos, el profesor puede seleccionar la opción *Generate teams*, visualizado en la figura 6.2, y acceder a dicha herramienta; en primera instancia, se presentará una ventana como se visualiza en el cuadrante 1 de la figura 6.6, en esta misma figura, se observa que el conjunto de opciones disponibles para la configuración de los tipos de grupos a generar, tiene una arquitectura de asistente de 4 vistas dirigida hasta el final, donde se observa las opciones seleccionadas y el botón de generar grupos. En la primera vista, el profesor debe especificar el código del curso para el que desea formar los grupos, en donde presenta la opción de poder seleccionarlo de una lista, como se observa en la figura 6.7; una vez se selecciona el

Figura 6.6: Vistas de las opciones disponibles para la generación de grupos.

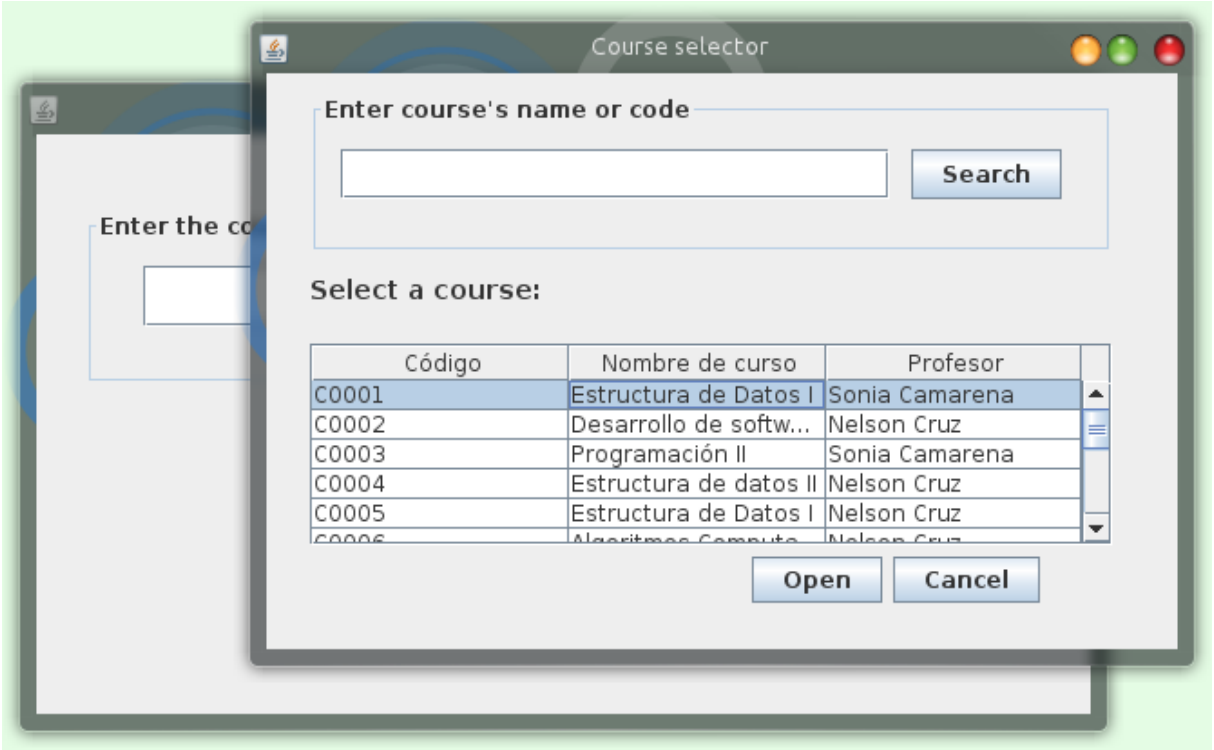


Figura 6.7: Ventana de selección de cursos.

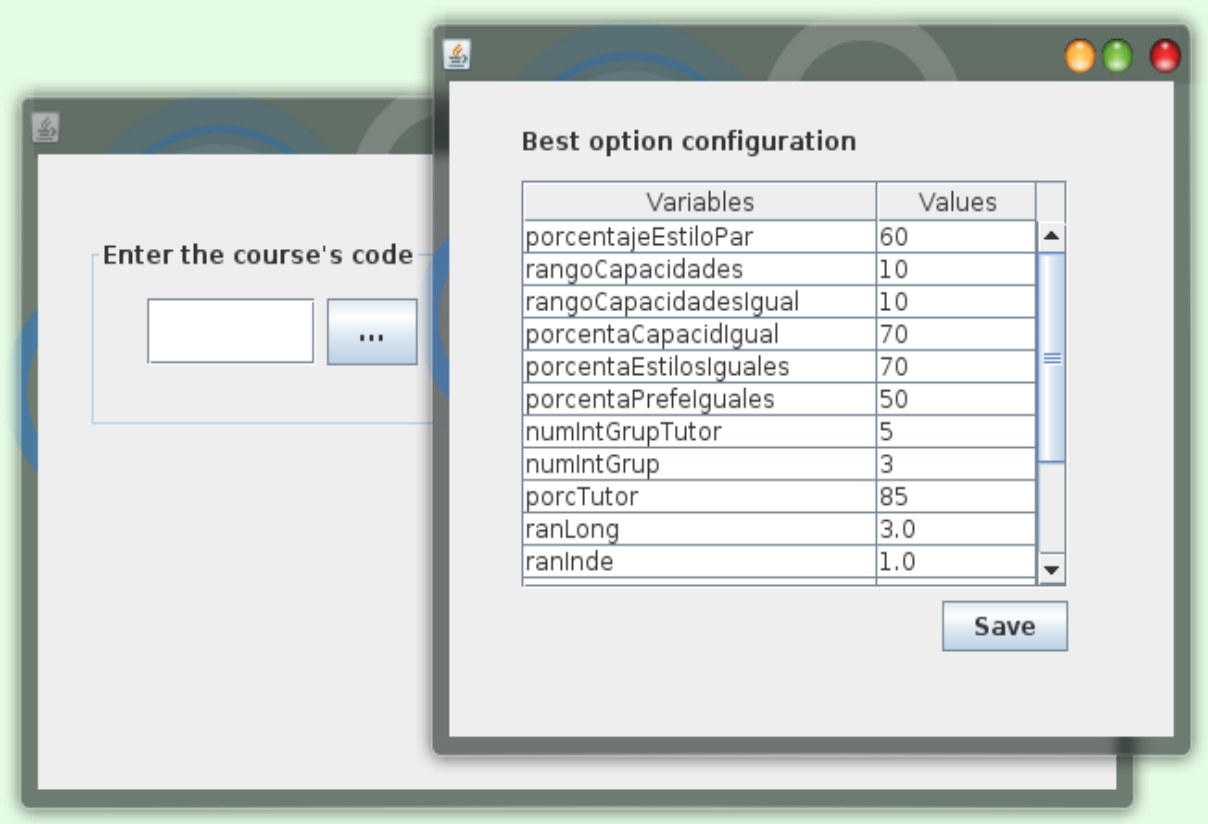


Figura 6.8: Ventana de configuración de las variables internas.

curso de la lista, se procede a dar clic en el botón *Open* y éste, copiara el código del curso en el campo de texto con letras rojas, como se observa en la vista 1 de la figura anterior; en la misma se observa un panel llamado *Select the type of team*, donde se presenta las tres opciones principales para la selección de grupos: *Pair*, que representa la selección de grupo Par, *Group*, la selección para formar grupos de 3 o más, y *Best option*, permitirá al sistema seleccionar la mejor opción de grupo, explicado en la sección 4.8.2; esta misma vista, se observa un botón con etiqueta *Config.*, dicho botón despliega una ventana que permite editar los valores de las variables internas del sistema, como fue explicado en la sección 4.9. La figura 6.8, presenta una vista de la ventana de configuración, en la que el profesor podrá editar los valores de dichas variables; esto será útil para configurar el comportamiento deseado por parte del sistema en la determinación de la mejor opción. Continuando con la descripción de la vista 1 de la figura 6.6, si el profesor selecciona la opción *Pair*, el asistente lo llevará a la vista 3; si selecciona *Group*, cambiará a la vista 2 y la opción *Best option*, llevará al asistente hasta la vista 4. En caso de seleccionar la opción *Group*, la vista 2 se pre-

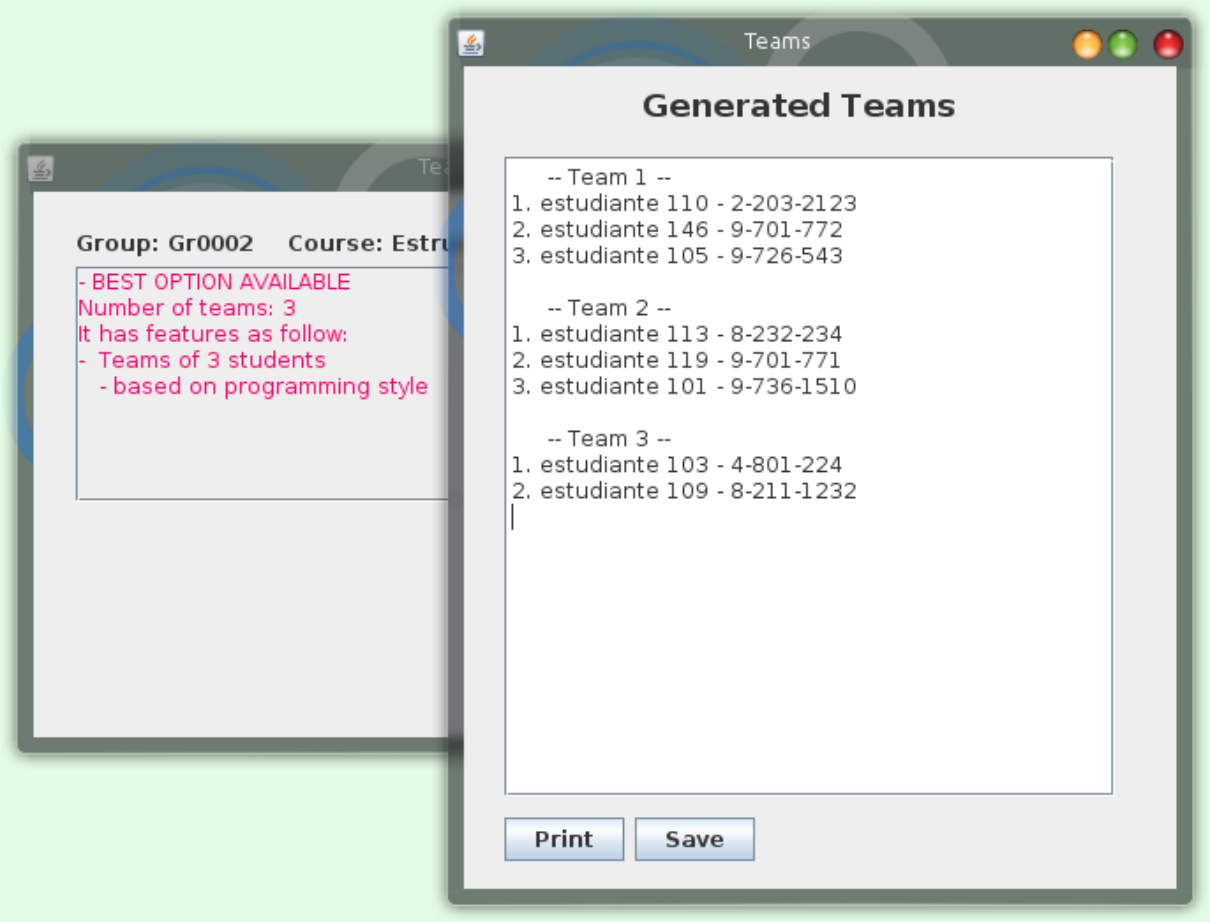


Figura 6.9: Vista de la ventana con los grupos generados.

sentará y el profesor debe especificar el número de integrantes por grupo; por defecto se encuentra en 3, de igual forma, se observa el número total de estudiantes en el curso para que se tenga referencias del mismo; luego, cuando se fija dicho número el asistente llevará al profesor a la vista 3 donde se observa los tipos de grupos con las características a formar, estas opciones se basan en lo expuesto en la sección 4.8.1, para seleccionar la formación de grupos con tutor, se aprecia un *checkbox* donde se puede habilitar o deshabilitar dicha opción. Al final, en la vista 4, se observa todas las opciones seleccionadas para la formación de grupos, conjuntamente con el código de grupo y el nombre de la materia; de igual forma, puede editarse opciones ya seleccionadas hasta ese punto, simplemente dándole clic en *Back*, hasta la opción deseada; en el botón *Generate teams*, se procede a dar clic para la generación de los grupos y se desplegará una ventana aparte, como se observa en la figura 6.9. En esta figura se observa los grupos generados en la que se indica el nombre del estudiante y al lado

su respectivo ID, cada grupo está enumerado secuencialmente y en caso de haber seleccionado la opción con tutor, el sistema presentará el estudiante tutor con dicha palabra al inicio de su nombre, indicando que es el estudiante tutor.

El profesor una vez generado el grupo, podrá imprimir o guardar en archivo de texto dichos grupos con sus respectivos nombres; por otro lado, SOFORG permite cambiar las configuraciones y seguir generando tipos de grupos al gusto del profesor, sin necesidad de cerrar la ventana de configuración, ni la de los grupos generados.

Capítulo 7

Validación y evaluación del sistema

Hasta este punto, la construcción de la herramienta ha sido llevada a cabo, tomando en cuenta, los criterios y requerimientos planteados en la arquitectura de software del capítulo 4. Siguiendo con la metodología planteada en la sección 2.1, este capítulo desarrollara fase 6 del mismo, en la que se plantea un caso de estudio que permitirá evaluar la herramienta, con el propósito de obtener resultados que nos permita validar una de las hipótesis planteadas en la sección 2.4. Se describirá de forma global, el escenario utilizado para dicha evaluación y los detalles del mismo; además de los resultados obtenidos.

7.1. Descripción de la evaluación y escenario

La evaluación de la herramienta fue realizada en base a la metodología presentada en la sección 2.3. Esta evaluación se desarrolló en la Universidad Tecnológica de Panamá, en la que se tomó 2 grupos de estudiantes del curso de Desarrollo de Software II, de la Licenciatura en Desarrollo de Software, esta carrera pertenece a la Facultad de Ingeniería en Sistemas Computacionales (FISC); el primer grupo cuenta con 13 estudiante, al que se le asignó el código Gr0007 y el segundo, se le asigna código Gr0008 cuyo número de estudiantes es de 14; estos cursos abordan la programación básica en lenguaje Java. El periodo en que se llevó a cabo dicha evaluación, fue el segundo semestre del año lectivo 2011 y se aplicaron 2 problemas de progra-

mación. Los estudiantes tenían que desarrollar los problemas de manera individual, preferiblemente dentro del laboratorio de clases, sin embargo podían ser llevados a casa, para su desarrollo de igual forma. Los problemas debían ser resuelto en lenguaje Java y agregar, en forma de comentario, datos como: nombre del estudiante, cédula, código de ejercicio; el enunciado de estos problemas se observan en el Anexo 2 y 3 respectivamente, en la que se presenta las indicaciones de como debe ser resuelto el problema y el método de envío. Los estudiantes deben enviar sus programas en archivos con extensión *java*, a un e-mail proporcionado en dichos enunciados; por otro lado, un problema es de aplicación básica de Java y el otro se aplica POO para su resolución.

Una vez que se obtuvieron los archivos con la solución de los estudiantes, se procedió a ingresar dichos estudiantes al sistema, haciendo referencia a un código de grupo; de igual forma, se ingresó el curso para el que se esta desarrollando la prueba. Cada solución de los estudiantes fue analizado por SOFORG a través del analizador estático, éste emitió un resultado con las métricas de los estilos de programación y preferencias, además de la evaluación; toda esta información fue almacenada en el sistema. Cuando se finalizó el análisis estático de todos los códigos fuentes, se procedió a formar los grupos de trabajo para cada grupo. De esta manera, cada grupo presentó su propio código de curso y para cada curso se evaluará la opción de formación de *Best Option*, como se encuentra descrito en la figura 6.6, así, el análisis empezó por los datos de los estudiantes dentro del curso, en la que se recomendó la mejor opción para la formación de los grupos. Una vez que se obtuvieron los resultados de dicha opción, se procedió a la generación de los grupos. Los resultados obtenidos de esta formación, serán comparados con las métricas obtenidas del análisis estático; esto nos permitirá visualizar los valores y realizar una discusión de los mismos. En las siguientes secciones, se describen los resultados que se obtuvieron y una discusión de dichos resultados.

7.2. Resultados

Después de ser evaluados los códigos fuentes de los estudiantes, se describe el resultado de los mismos. En primera instancia, las Tablas 7.1, 7.2 y 7.3, describen las métricas de los estilos de programación, las preferencias y valores de las capacidades,

Tabla 7.1: Estudiantes del grupo Gr0007, con la evaluación de los estilos de programación

No.	Nombre de estudiante	LI	I	CB	BL	SCL	D	IV
1	Luis Obaldia	9.86	0.0	1	0.30	0.99	0.86	45
2	Edgar Espinosa	4.43	0.25	1	0.15	1.05	0.58	25
3	Julio Archibold	3.71	0.27	0	0.06	1.26	0.54	50
4	Mario Mejia	6.10	0.05	0	0.24	1.35	0.53	36
5	Miguel Cedeño	5.80	1.29	0	0.06	1.36	0.76	47
6	Rigel Rebolledo	7.33	0.0	0	0.01	1.17	0.59	43
7	Abdiel Williams	4.42	0.0	0	0.39	1.08	0.31	23
8	Arney Baxter	4.43	0.05	0	0.18	1.36	0.45	47
9	Gabriel Garrido	3.19	0.07	1	0.12	1.00	0.73	41
10	Jonathan De Bello	5.25	0.63	1	0.28	1.05	0.69	60
11	Kevin Carcache	4.48	0.01	0	0.05	1.22	7.22	50
12	Leandro Garces	4.12	0.27	0	0.03	1.28	0.52	50
13	Sherly Acosta	6.62	0.06	0	0.14	1.30	0.41	47

Tabla 7.2: Estudiantes del grupo Gr0007, con evaluación de las preferencias

No.	Nombre de estudiante	IE	IES	FWD	WD
1	Luis Obaldia	elseif	null	dowhile	dowhile
2	Edgar Espinosa	ifelse	switch	while	null
3	Julio Archibold	ifelse	elseif	dowhile	null
4	Mario Mejia	ifelse	elseif	dowhile	null
5	Miguel Cedeño	elseif	elseif	dowhile	dowhile
6	Rigel Rebolledo	elseif	switch	dowhile	null
7	Abdiel Williams	ifelse	switch	while	null
8	Arney Baxter	elseif	elseif	dowhile	dowhile
9	Gabriel Garrido	ifelse	ifelse	for	dowhile
10	Jonathan De Bello	elseif	switch	null	dowhile
11	Kevin Carcache	ifelse	switch	while	null
12	Leandro Garces	ifelse	elseif	dowhile	null
13	Sherly Acosta	ifelse	elseif	dowhile	dowhile

Tabla 7.3: Estudiantes del grupo Gr0007, con la evaluación de las capacidades

No.	Nombre de estudiante	Métrica capacidades
1	Luis Obaldia	80
2	Edgar Espinosa	88
3	Julio Archibold	92
4	Mario Mejia	85
5	Miguel Cedeño	95
6	Rigel Rebolledo	95
7	Abdiel Williams	90
8	Arney Baxter	95
9	Gabriel Garrido	98
10	Jonathan De Bello	100
11	Kevin Carcache	95
12	Leandro Garces	90
13	Sherly Acosta	95

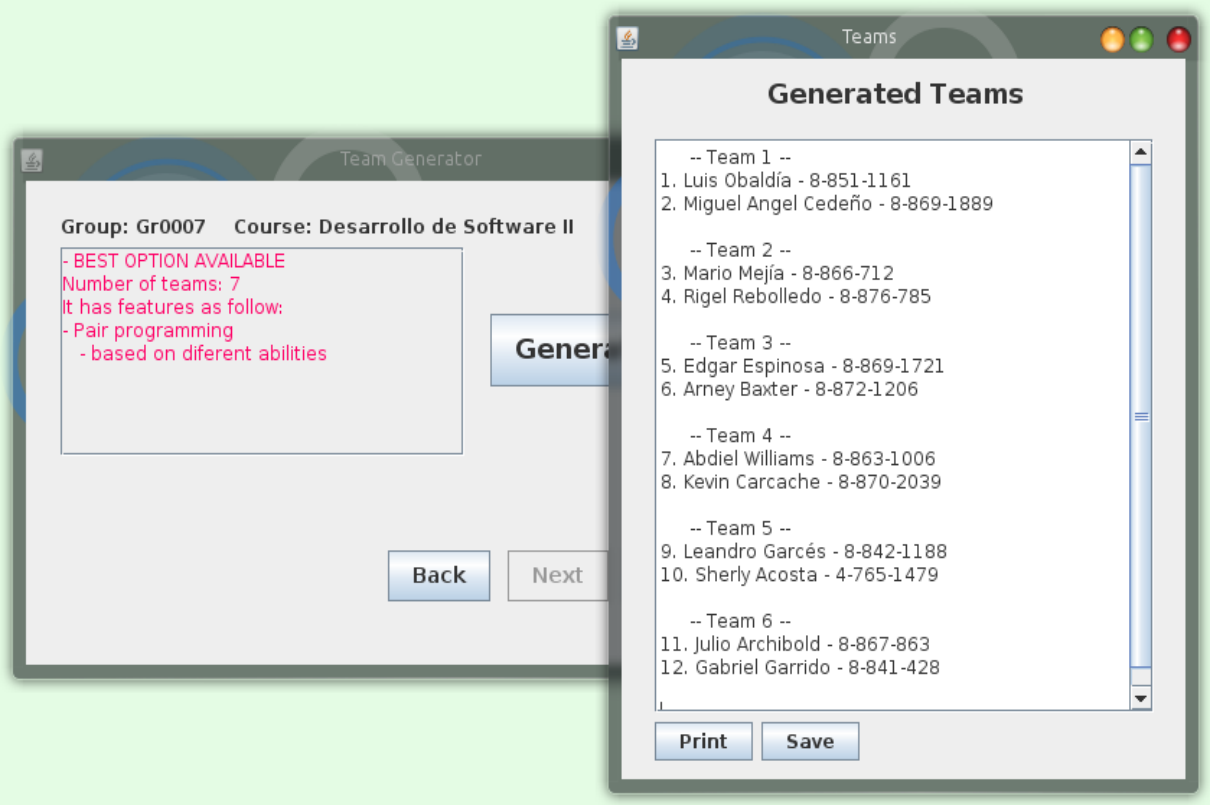


Figura 7.1: Vista de la ventana con los grupos de trabajos generados (Grupo Gr0007).

respectivamente; todo esto por estudiante, como resultado del análisis estático del grupo Gr0007. Al aplicar la opción *Best Option*, se obtiene que la mejor opción de formación de grupo es la de Grupo Par (P) basado en capacidades diferentes (NCP); y, al generarse los mismos, se obtienen 7 grupos de 2 integrantes que son visualizados en la figura 7.1, en la que se observa dichas opciones seleccionadas, todo esto con letras rojas en el área de texto ubicado en la parte izquierda. Cabe destacar que, como el grupo Gr0007 presenta 13 estudiantes, el último grupo (*Team 8*) no se encuentra generado ya que sólo contaría con un integrante, este estudiante, como excepción, pudiera ser ubicado en cualquier otro grupo o, simplemente, que el profesor seleccione otra opción manual de formación que le permita agruparlos de manera más equitativa, como por ejemplo: formarlos en grupo de 3 y agregar el último integrante, al grupo anterior formado.

Por otro lado, el grupo Gr0008 presentó los siguientes resultados expuestos en las Tablas 7.4, 7.5 y 7.6. Estos resultados describen de igual forma, las métricas de estilos de programación, selección de preferencias y valores de capacidades basados en los códigos analizados. Al aplicarse la opción que describe la selección de la mejor opción en la formación de grupos, se obtuvo que la formación de grupos de 3 integrantes basado en estilos de programación similares (EP), es la más adecuada. Con este tipo de información se obtiene una formación de 5 grupos en total, en la que el último grupo (Team 5), esta conformado por 2 integrantes que pertenece a los estudiantes sobrantes. En este caso, el profesor puede optar por mantener dicho grupo o agregarlos a cualquier otro de los grupos formados. La figura 7.2, presenta la vista de la ventana con los grupos generados de la opción seleccionada, aquí se presenta los nombres de los estudiantes y su número de cédula o ID.

De igual forma, SOFORG analizó en busca de sugerencias de buenas prácticas que los estudiantes podían implementar. Las buenas prácticas que tuvieron mayor incidencias, ordenadas de mayor a menor, fueron:

1. Atributos que no han sido inicializados en el constructor
2. Atributos que no han sido declarados *private* o *protected*.
3. Clases que no tienen constructor.

Tabla 7.4: Estudiantes del grupo Gr0008, con la evaluación de los estilos de programación

No.	Nombre de estudiante	LI	I	CB	BL	SCL	D	IV
1	Davis Nelson	5.82	0.03	0	0.29	1.20	0.51	66
2	Francisco Lau	6.41	0.01	0	0.11	1.26	1.60	46
3	Jairo Ramos	5.76	0.02	0	0.05	1.17	0.53	50
4	Jesus Samaniego	4.13	0.02	0	0.07	1.10	1.57	76
5	Nidia Salceda	4.33	0.21	0	0.40	1.28	0.46	33
6	Steven Caballero	5.88	0.06	0	0.01	1.18	0.95	75
7	Eduardo Griffith	5.28	0.20	1	0.13	1.10	0.45	73
8	Francisco Him	5.17	0.01	0	0.04	1.26	1.77	30
9	Fidelino Camarena	6.36	0.01	0	0.25	1.30	1.68	43
10	Franklin Rodriguez	5.75	0.97	0	0.28	0.97	3.14	36
11	Frederick Sanson	4.62	0.0	1	0.17	1.30	1.44	57
12	Israel Gamas	5.51	0.23	0	0.08	1.18	0.34	75
13	Juan Munoz	4.45	0.05	0	0.21	1.36	0.43	47
14	Yeiluant Flores	8.64	0.14	0	0.44	1.28	8.91	34

Tabla 7.5: Estudiantes del grupo Gr0008, con evaluación de las preferencias

No.	Nombre de estudiante	IE	IES	FWD	WD
1	Davis Nelson	ifelse	elseif	for	dowhile
2	Francisco Lau	elseif	switch	while	null
3	Jairo Ramos	ifelse	elseif	for	dowhile
4	Jesus Samaniego	ifelse	elseif	while	null
5	Nidia Salceda	ifelse	switch	dowhile	null
6	Steven Caballero	elseif	null	for	dowhile
7	Eduardo Griffith	elseif	elseif	while	null
8	Francisco Him	ifelse	null	while	while
9	Fidelino Camarena	elseif	elseif	dowhile	dowhile
10	Franklin Rodriguez	null	null	null	null
11	Frederick Sanson	elseif	elseif	dowhile	dowhile
12	Israel Gamas	elseif	elseif	while	while
13	Juan Munoz	elseif	elseif	dowhile	dowhile
14	Yeiluant Flores	elseif	elseif	dowhile	null

Tabla 7.6: Estudiantes del grupo Gr0008, con la evaluación de las capacidades

No.	Nombre de estudiante	Métrica capacidades
1	Davis Nelson	85
2	Francisco Lau	85
3	Jairo Ramos	80
4	Jesus Samaniego	68
5	Nidia Salceda	85
6	Steven Caballero	85
7	Eduardo Griffith	90
8	Francisco Him	85
9	Fidelino Camarena	80
10	Franklin Rodriguez	10
11	Frederick Sanson	90
12	Israel Gamas	90
13	Juan Munoz	80
14	Yeiluant Flores	85

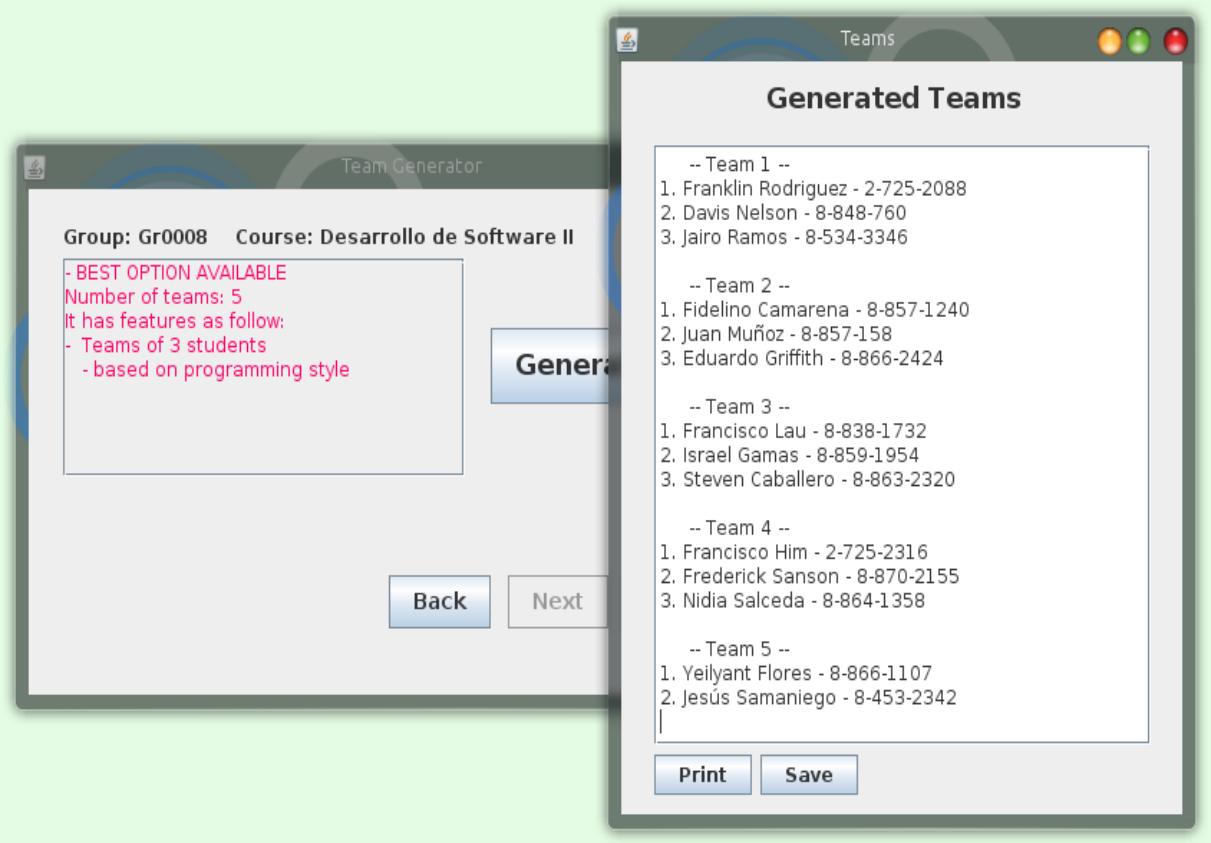


Figura 7.2: Vista de la ventana con los grupos de trabajos generados (Grupo Gr0008).

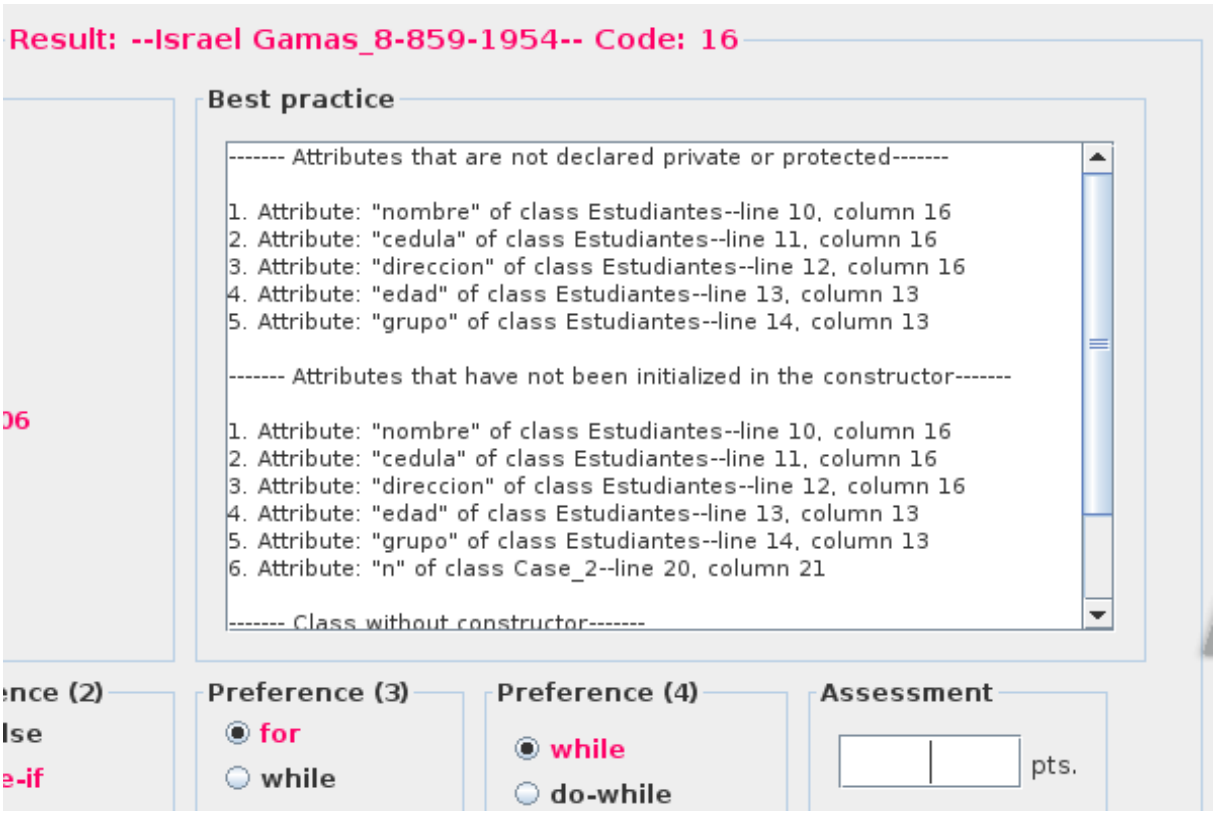


Figura 7.3: Vista de las sugerencias de buenas prácticas de uno de los estudiantes analizados.

De esta manera, la primera sugerencia descrita en la lista anterior, representa la de mayor incidencia en el análisis, ya que fue la de mayor frecuente en la mayoría de los estudiantes; sin embargo, aunque la sugerencia de buenas prácticas No. 3 de la lista anterior, no fue la de mayor incidencia, ésta fue la única encontrada en el resultado del análisis estático de todos los estudiantes que participaron. Por otro lado, las demás sugerencias que se obtuvieron fuera de la lista, fueron de menor frecuencia entre los estudiantes y algunas, simplemente, no se evaluaron debido a la naturaleza del problema presentado. Buenas prácticas como: métodos que pueden ser declarados *static* y clases anidadas que pueden ser declaradas *static*, representan aquellas que no han sido evaluadas dentro de los problemas presentados, por lo que no se obtuvo incidencia de los mismos. En la figura 7.3, se observa una vista del resultado de las buenas prácticas sugerida a uno de los estudiantes al que se le analizó su ejercicio; este análisis pertenece al enunciado del Anexo 3, que proponía un desarrollo utilizando POO.

7.3. Análisis de los resultados

En la sección anterior se observo los resultados del análisis y formación de grupos utilizando la herramienta SOFORG, se puede decir que los estudiantes mantienen un patrón parecido en cuando a los estilos que utilizan para programar; estos estudiantes, en su mayoría, ponen las llaves en la siguiente línea de código después de declarar la estructura (CB), utilizan poco o ningún espacio entre líneas (BL), en cuanto al porcentaje de inicialización de variables (IV), se mantiene un promedio aproximado del 50 % en su mayoría, de igual forma, hablar de las preferencias y el resto de las métricas, guardan relación entre los valores obtenidos, sin embargo no son suficientes para ser considerados como un grupo que admite una formación de estilos de programación de grupo Par(P), como fue descrito en la sección 4.8.2. Fuera de las métricas evaluadas, se puede observar otros tipos de patrones que particularizan los estilos de programación adoptados por los estudiantes, es el caso del código fuente observado en el Anexo 4, en la que se presenta como un estudiante declara, en la misma sentencia lógica de un IF-ELSE, una operación aritmética que es evaluada en dicha sentencia; normalmente no se acostumbra a hacer todo en la misma declaración, ya que muchas veces se puede incurrir en desorden, por lo que muchos prefieren llevar las operaciones a una variable y de allí ser comparada en la sentencia lógica planteada.

El resultado de este análisis sugiere, como mejor opción de formación para el grupo Gr0007, grupos Par(P) basado en diferentes capacidades(NCP), como ya fue mencionado en la sección anterior. Esto se debe a que si se analiza el algoritmo de la figura 4.13, que describe la selección de la mejor opción, se puede observar que en primera instancia, se analiza si el grupo tiene estilos de programación semejantes en un 75 % (valor por defecto de la variable interna *porcentajeEstiloPar*, detallado en la sección 4.9); para el caso presentado del grupo Gr0007, las métricas de los estilos de programación y preferencias del mismo son detallados en las Tablas 7.1 y 7.2, en la que se observa a simple vista, que no existe dicha semejanza; esto, tomando en cuenta los valores por defectos de los rangos de semejanzas por estilo de programación, pertenecientes a las variables internas (*ranLong*, *ranInde*, *ranDocu*, *ranLine*, *ranSent*, *ranInic*, descrito en la sección 4.9). De esta manera, el algoritmo descarta esta posibilidad y evalúa la segunda opción que es, verificar si tienen distantes capacidades; para esto se saca el porcentaje de las capacidades mayores o menores del promedio de capacidades y si está dentro del rango de 45 y 55 % (basado en variable interna *ran-*

Tabla 7.7: Estudiantes del grupo Gr0007, ordenados por el valor de sus capacidades de menor a mayor

No.	Nombre de estudiante	Métrica capacidades
1	Luis Obaldia	80
2	Mario Mejia	85
3	Edgar Espinosa	88
4	Abdiel Williams	90
5	Leandro Garces	90
6	Julio Archibold	92
7	Miguel Cedeño	95
8	Arney Baxter	95
9	Rigel Rebolledo	95
10	Sherly Acosta	95
11	Kevin Carcache	95
12	Gabriel Garrido	98
13	Jonathan De Bello	100

goCapacidades, cuyo valor por defecto es 10), entonces se acepta dicha opción para la selección de la mejor opción de formación de grupos; en la Tabla 7.3, se describe los valores de las capacidades obtenidas; utilizando dichos valores, el promedio calculado es de 92 %, a simple vista se observa que 7 estudiantes tienen valores mayores que el promedio, esto representa un 53 % y se acepta dentro del rango presentado y por consiguiente, la formación de grupo perteneciente a esta opción. En la Tabla 7.7, se presenta las capacidades ordenadas de los estudiantes y si se compara con el resultado obtenido de la formación de grupos de la figura 7.1, se observará que cada grupo se formo siguiendo el algoritmo de la sección 4.8.3, que describe dicho proceso. Para este caso, los grupos formados no presentan diferencias notorias en cuanto a las capacidades, sin embargo, es funcional ya que entre menos es diferencia de capacidades entre los dos estudiante del grupo, menor es el esfuerzo requerido para nivelar los conocimientos de dichos estudiantes.

Por otro lado, si analizamos el resultado del grupo Gr0008, observamos que se obtuvo una formación de 3 basada en estilos de programación similares (EP). En el algoritmo descrito de la figura 4.13, la formación obtenida representa, como última instancia, la opción a seleccionar, después de evaluar todas las existentes. Las Tablas 7.4 y 7.5, describen las métricas de los estilos de programación y las preferencias del grupo Gr0008, en la que se observa a simple vista, que no existen similitudes notables en los estilos de programación de dichos estudiantes, todo esto, tomando en cuenta

el algoritmo mencionado y los valores por defecto de las variables internas; por otro lado, la evaluación de las diferencias en las capacidades, muestra que el promedio de capacidades (presentado en la Tabla 7.6) es de 78 % y los valores mayores de dicho promedio, representan el 85 %, por lo que no se acepta en el rango de diferentes capacidades. Siguiendo con la próxima evaluación del algoritmo mencionado, se procede a comprobar entonces, la similitud de las capacidades; aquí se evalúa un rango entre 73 y 83, esto basado en el promedio de capacidades y la variable interna *rangoCapacidIgual*, que tiene valor 10 por defecto; el promedio de los valores que se encuentran dentro del rango es de 21 %, muy por debajo del promedio de referencia que es de 75 % cuyo valor es obtenido de la variable interna *porcentaCapacidIgual*. Si bien es cierto que, a simple vista, se observa similitudes en los valores de las capacidades, la métrica del estudiante No. 10, hace no cumplir con dicha opción; esto se debe a que si se formarán los grupos basados en capacidades similares, al incluir dicho estudiante en cualquier grupo, afectaría el rendimiento de ese grupo, debido a la baja capacidad que posee el estudiante; por lo que ya no sería un solo estudiante afectado, sino un grupo completo de los que se formarían, ya que no generarían los conocimientos suficiente con relación a los demás grupos del curso. Como última opción, se tiene la formación basado en estilos de programación similares, pero antes debe ser evaluada la posibilidad de ser formados con o sin tutor; esto se realiza verificando si existe un número de valores de las capacidades mayores de 91 (valor por defecto de la variable interna *porcTutor*), igual al número de posibles grupos a formar; para este caso se formarán 5 grupos, sin embargo, ningún valor es mayor que 91, por lo que simplemente se formaran sin tutor. De esta manera, la formación de grupos de 3 basado en estilo de programación (EP), se observó en la figura 7.2; dicha formación sigue el algoritmo descrito en la sección 4.8.3, en la que se busca la mejor combinación de los posibles grupos a formar, evaluando la de menor distancia entre los estudiantes. En la Tabla 7.8, se observa las primeras 10 posibles formaciones de grupos, ordenadas de menor a mayor basado en la distancia calculada de los estilos de programación, entre los estudiantes del grupo Gr0008. En dicha Tabla, se observa que la primera fila pertenece a la formación obtenida de la figura 7.2, basado en el listado de estudiante de las Tablas 7.1, 7.2 y 7.3; dónde la columna *No.*, presenta el número de referencia de las 298582 posibles formaciones de grupos; las columnas de grupos muestran las posibles formaciones de grupo basado en los 5 grupos a formar; y la columna *Distancia*, muestra la distancia calculada para dichas formaciones. De esta manera, se obtuvo la de menor distancia, para la formación de grupos de 3 basado en estilo de programación.

Tabla 7.8: Las primeras 10 formaciones de grupos, ordenadas ascendentemente basado en la distancia calculada del grupo Gr0008.

No.	Grupo 1	Grupo 2	Grupo 3	Grupo 4	Grupo 5	Distancia
86596	10 1 3	9 13 7	2 12 6	8 11 5	14 4	15.63
262978	10 13 1	9 7 3	2 12 6	8 11 5	14 4	15.85
33698	6 4 3	9 10 1	2 7 13	8 11 5	14 12	16.45
53315	7 4 3	9 10 1	2 13 6	8 11 5	14 12	17.0
270547	10 2 5	9 13 8	4 11 12	1 3 7	14 6	17.20
86573	10 1 3	9 13 7	2 4 6	8 11 5	14 12	17.22
262955	10 13 1	9 7 3	2 4 6	8 11 5	14 12	17.25
83026	10 1 3	6 13 4	2 7 9	8 11 5	14 12	17.28
257003	10 9 13	4 12 1	2 5 8	6 11 7	14 3	17.35
35752	6 4 3	9 13 7	2 1 10	8 11 5	14 12	17.36

De esta manera, se evaluó la funcionalidad de la herramienta, en la que se analizó dos escenarios distintos de formación de grupos y se pudo corroborar su correcto funcionamiento, con los datos reales extraído de los estudiantes participantes. Tomando en cuenta las hipótesis expuestas en la sección 2.4, se resuelve descartar la hipótesis nula (H_0) y adoptar la alternativa (H_1), en la que se dice que SOFORG mejora el proceso de enseñanza de la programación, ya que dá la posibilidad de formar grupos de trabajos tomando en cuenta ciertos criterios y consideraciones; esto se sustenta a través de la teoría que presentan de Faria et al. (2006), en la que se expone una investigación, donde se comprueban los beneficios de formar grupos de trabajos basados en capacidades y estilos de programación, dando como resultados rendimientos mejores por parte de los estudiantes (detallado en el punto 3.2.3); de igual forma, las investigaciones de Scott et. al (1994, 1995), proponen la formación de grupos de trabajos, utilizando ciertos parámetros, conjuntamente con algunos escenarios que fueron evaluados en dichas investigaciones. Estos fundamentos teóricos han sido considerados principalmente, en las secciones 4.6 y 4.8.

Capítulo 8

Conclusiones y trabajos futuros

Una vez que se ha concluido esta investigación, en la que se siguió la metodología propuesta del capítulo 2, que establece ciertas hipótesis y la evaluación de las mismas, conjuntamente con el funcionamiento de la herramienta, se presenta las conclusiones finales de las mismas, que se detallarán en este capítulo; de igual forma, se presentarán las publicaciones realizadas de los trabajos realizados y la propuesta de las posibles investigaciones futuras, ésto permitirá la profundización en los temas y problemáticas abordadas a lo largo de esta trabajo.

8.1. Conclusiones

La didáctica es considerada un arte por muchos expertos, en la que el docente debe ser un artista a la hora de aplicar los métodos y recursos didácticos; sin embargo, con el pasar de los tiempos y el avance en las investigaciones científicas, se han determinados procedimientos concretos que facilitan dicha tarea; los grupos de trabajos como método didáctico, ha sido utilizado por mucho tiempo en el proceso de enseñanza-aprendizaje de la programación; sin embargo, muchas veces no se le ha sacado el provecho como recurso disponible, esto se debe a la deficiente implementación que se le ha dado; los resultados esperados de este recurso, dependerá de las consideraciones que se tengan, en cuanto al estado actual de los estudiantes y así aplicar las variaciones pertinentes. Este trabajo presentó un marco teórico que

permitió el desarrollo del mismo, en la que se fundamentó las arquitecturas propuestas y funcionalidades concretas del proyecto.

La evaluación de criterios y aspectos concretos de los estudiantes, es de vital importancia en el proceso educativo, ya que éste, puede ayudar al docente a tener un mejor panorama de la situación actual; uno de los principales problemas en esta tarea, es la identificación y obtención de ésta información. En este proyecto, se propuso obtener información de los estudiantes basado en sus estilos de programación, preferencias de estructuras, buenas prácticas y capacidades, en la que se detalló los aspectos más importantes para cada categoría; sin embargo, tras la realización del caso de estudio donde se evaluó el código fuente de un grupo de estudiantes, se pudo observar que, existen otros aspectos que pudieran ser considerados y agregados dentro del análisis de SOFORG, enriqueciendo dicho proceso.

Los analizadores estáticos son una alternativa a la hora de evaluar códigos fuentes, de hecho, la programación ha tenido sus fundamentos en este principio; la implementación de esta técnica en SOFORG, ha facilitado el cálculo de las métricas, referente a los estilos de programación de los estudiantes, además de otras informaciones extraída; esta implementación se llevó a cabo, utilizando GIC con la herramienta JavaCC, obteniéndose resultados positivos, ya que hubieron mejoras en el proceso de desarrollo de SOFORG.

Las aplicaciones basadas en conocimientos están siendo muy utilizadas en el desarrollo de software cotidianos, donde le aplican una mayor funcionalidad a los mismos, aparte de ayudar en el proceso de desarrollo y obtener mejores resultados en cuanto a la mantenibilidad del software; estas implementaciones se han facilitado con la formalización de los lenguajes ontológicos; sin embargo, hay que destacar que, las aplicaciones de las ontologías pueden ser reemplazadas, como se hacía tradicionalmente, por aplicaciones informales, es decir, que muchas veces se hacían uso de ontologías (basado en su concepto) o de aplicaciones basadas en conocimientos, sin la utilización de lenguajes ontológicos o principios formales de Inteligencia Artificial, creándose estructuras de datos propias para una solución específica; de esta manera, las ontologías representan una alternativa para estos tipos de aplicaciones, ya que fueron creadas con el propósito de ser fundamento en la Web semántica y aplicaciones de servicios Web. En la herramienta presentada, se hizo uso de ontologías como solución a la necesidad de evaluar académicamente, el código fuente del estudiante,

donde se obtuvieron mejoras en el proceso de desarrollo, ya que con la formalidad de los lenguajes ontológicos, se tiene un patrón de diseño y una estructura formal que facilita principalmente, los procesos de mantenibilidad; y, para el caso concreto de la necesidad de tener ontologías bases con los modelos de ejercicios y sus instancias evaluadas de los estudiantes, se le sacó el provecho máximo a dicha tecnología, ya que para cada ejercicio involucra la estructuración de dicho conocimiento; sin embargo, existen desventajas en cuanto al rendimiento de ejecución, ya que se ve afectado por la complejidad y extensión de los problemas resueltos.

La estructuración del modelo para la formación de grupos de trabajos, principalmente la de mejor opción para la formación grupos, se basó en los fundamentos teóricos que sustentan su fiabilidad (de Faria et al., 2006; Scott y Cross, 1995; Scott et al., 1994); el funcionamiento de SOFORG pudo comprobarse a través del desarrollo de una evaluación que consideró un caso de estudio con estudiantes, el cuál, desarrollaron programas y se les extrajo la información necesaria, para que pudieran ser formados en grupos; obteniéndose como resultado, formaciones de grupos elocuentes a la información real que presentaban los estudiantes; además, se pudo observar que los estudiantes mantienen patrones similares de programación, en estilos de programación específicos. Se puede considerar ciertas mejoras a la herramienta que serán detallados en la sección de trabajos futuros.

El algoritmo de mejor opción para la formación de grupo descrito en la figura 4.13, aparte de buscar la mejor opción de formación, busca minimizar las diferencias entre las capacidades de los estudiantes en un determinado curso, esto se debe a que, entre mayor sea la similitud de los mismos, mayor será los conocimientos generados.

Aparte de lo ya mencionado, SOFORG es útil en la identificación de deficiencias basadas en buenas prácticas de los estudiantes (e.g. inicialización de variables, variables que no han sido declaradas *private* o *protected*, otros), que en la mayoría de los casos se vuelve una tarea tediosa para el profesor; también ayuda a mantener una estadística del porcentaje de avance de los estudiantes que permitirá evaluar la efectividad de las técnicas, metodologías, modelos de ejercicios y tipos de grupos utilizados por el profesor, además de otras investigaciones científicas dentro del proceso de enseñanza-aprendizaje en la programación.

8.2. Difusión de los resultados

Durante el desarrollo de este trabajo, se han obtenido las siguientes publicaciones:

Publicación 1

Título	**Ontology and Source Code Semantic-Based Groups Programming Learning**
Autores	Arosemena-Trejos, Davis and Crespo, Sergio and Clunie, Clifton
Datos de la publicación	EATIS '10: Proceedings of the 5th International Conference of the Euro-American Association on Telematics and Information Systems, Panamá, Panamá, 2012. ISBN-978-958-44-7280-9.
Estado	Publicado
Resumen	Muchos estudios han concluido que la utilización de grupos de trabajo para la enseñanza de la programación puede ser efectivo dentro del aula de clase, ya que ayuda a los estudiantes a generar y adquirir nuevos conocimientos en un menor tiempo; sin embargo, estos grupos al ser formados sin tomar en cuenta algunos aspectos, puede provocar un efecto contraproducente en el proceso de enseñanza-aprendizaje. Este trabajo propone un Sistema Ontológico para la Formación de Grupos de trabajo(SOFORG). Este sistema extraerá la semántica del código fuente de cada estudiante a través de un análisis estático, que permita la construcción de una estructura ontológica en lenguaje OWL, útil en la inferencia de información que será de ayuda en la formación de grupos de trabajos. La formación de estos grupos estará basada en aspectos como estilos de programación, deficiencias, niveles de capacidad y diferentes situaciones dentro del salón de clase.

Publicación 2

Título	**Criterios y consideraciones para la formación de grupos de trabajo en la enseñanza de la programacion**
Autores	Arosemena-Trejos, Davis and Crespo, Sergio and Clunie, Clifton
Datos de la publicación	Edutec 2010: Proceedings of the Edutec, Bilbao, España, 2010. ISBN-978-84-9860-447-4. http://2edutec2010.wordpress.com/
Estado	Publicado
Resumen	El proceso de enseñanza-aprendizaje de la programación no es una tarea sencilla, por lo que se han empleado esfuerzos para mejorar dicho proceso a través de métodos y técnicas propuestas; tal es el caso de utilizar los grupos de trabajo como método didáctico, en donde los estudiantes pueden desarrollar la obtención de nuevos conocimientos en un tiempo menor. Sin embargo, no siempre se obtienen estos beneficios debido a que existen factores directos e indirectos que afectan el éxito de los grupos; por lo que se hace necesario tomar en cuenta ciertas consideraciones a la hora de utilizar dicho método didáctico. Este artículo presenta un conjunto de criterios y consideraciones para la formación de grupos de trabajos; basado en ésto se propone un esquema de decisión en donde se considera la situación actual de los estudiantes; esta situación puede estar representada por las capacidades, estilos de programación y deficiencias dentro del aula de clase. Este esquema servirá como soporte al profesor a la hora de utilizar este método didáctico para la enseñanza de la programación.

Publicación 3

Título	**Forming Teams for Teaching Programming based on Static Code Analysis**
Autores	Arosemena-Trejos, Davis and Crespo, Sergio and Clunie, Clifton
Datos de la publicación	IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 3, No.1, May 2012. ISSN (online): 1694-0814
Estado	Aceptado, por publicar
Resumen	The use of team for teaching programming can be effective in the classroom because it helps students to generate and acquire new knowledge in less time, but these groups to be formed without taking into account some respects, may cause an adverse effect on the teaching-learning process. This paper proposes a tool for the formation of team based on the semantics of source code (SOFORG). This semantics is based on metrics extracted from the preferences, styles and good programming practices. All this is achieved through a static analysis of code that each student develops. In this way, you will have a record of students with the information extracted, it evaluates the best formation of teams in a given course. The teams formations are based on programming styles, skills, pair programming or with leader.

8.3. Trabajos futuros

Como trabajos futuros, se propone la realización de otras evaluaciones que permitirán mejorar el sistema; para luego ser implementado en la FISC de la Universidad Tecnológica de Panamá, para que sea uso de los profesores que laboran allí, de esta manera, se evaluará el impacto en la enseñanza y su rendimiento a larga escala. Por otro lado, se propone agregar otras reglas de decisión en la detección de otras preferencias, estilos de programación y buenas prácticas; también, evaluar otros estilos de programación tales como, el nivel de condensación del código fuente y las preferencias en cuando a la utilización de vectores o variables auxiliares, entre otros; ésto, producto de la evaluación realizada al mismo. Además, se propone incluir otros tipos de grupos a formar y complementar el módulo de generación de informes, donde se incluya otros tipos de informes relevantes, gráficas y estados de los estudiantes; además de presentarse el rendimiento de los docentes y las deficiencia de los estudiantes. Por otro lado, se propone mejorar el rendimiento en el proceso de evaluación académica del código fuente, donde se creen estructuras de datos personalizada para el manejo de dicha ontología y así se mejora el rendimiento del mismo; por otro lado, mejorar el algoritmo de formación de grupos basado en estilo de programación.

Apéndice A

Grupo vs. Equipo

En la siguiente tabla, se presenta las diferencias que existen entre un grupo de trabajo y un equipo, en la que se detallan características claves de ambos.

	Grupo de trabajo	**Equipo**
Liderazgo	Líder claramente definido: el jefe	Liderazgo compartido
Objetivo	Los miembros persiguen finalidades propias. El objetivo fijado por la organización o por el líder, es secundario	Específico y generado por él mismo, normalmente en concordancia con los objetivos de la organización
Tareas	Individuales	Interdependientes
Resultados	Fruto del trabajo individual	Resultado del trabajo colectivo
Efectividad y/o Eficacia	Medida de forma individual	Medida directamente por los resultados del trabajo colectivo
Sinergia	Neutral (a veces negativa)	Positiva
Responsabilidad	Individual	Individual y mutua
Habilidades	Aleatorias y variadas	Complementarias
Reuniones	Se sostiene por largos periodos de tiempo, se maneja eficientemente	Se fomenta las discusiones abiertas y la resolución activa de problemas

	Grupo de trabajo	**Equipo**
Decisiones	Normalmente el líder o la organización son los que dicen qué se debe hacer. El resto del grupo discute sobre ésto, deciden cómo hacerlo	Un problema es responsabilidad del equipo y se espera que todos participen. Discuten, deciden y hacen el trabajo real juntos
Conflictos entre miembros	”Se barre bajo la alfombra” o se hace caso omiso de él. Por lo tanto, persiste hasta que se hace ingobernable	Se hace un esfuerzo consciente para emendar el conflicto antes de que se vuelva destructivo. Se admite que existen diferencias individuales y de perspectiva por lo que se prestan atención a las mismas

Fuente: U. Barcelona, 2006 / Capital Humano n^{o}. 206, pág. 100, Enero de 2007

Apéndice B

Criterios en la Formación de Grupos

El siguiente apéndice presenta con conjunto de criterios y consideraciones para la formación de grupos de trabajo en la enseñanza de la programación; ésto se plantea a través de la identificación de cuatro áreas de influencia, para luego generar un esquema de decisión que ayudará a formar dichos grupos. La información presentada en este apéndice, es un fragmento extraído de la publicación de Arosemena-Trejos et al. (2010) que complementa la información de la investigación desarrollada.

B.1. Áreas de influencia en el éxito de los grupos

Un conjunto de criterios y consideraciones son presentados a continuación en donde se ha divido en cuatro áreas. Éstas áreas son las de mayor influencia en el proceso de aprendizaje en ambientes colaborativos dentro del área de la programación, por lo que su consideración puede ser determinante. Éstas áreas de influencia se dan a través de un conjunto de técnicas didácticas en la que su dinámica o conjunto de combinaciones y variaciones tendrán un efecto en el éxito de los grupos de trabajo. El modelado de estos áreas de influencia puede aceptar otras y ésta a su vez cualquier otro tipo de técnica que puede ser aplicada. Estas áreas son: asignaciones de tareas, tamaño y complejidad de las tareas, tamaño del grupo y formación de grupos; son detalladas a continuación.

B.1.1. Asignaciones de tareas

Las asignaciones de las tareas representan la manera de cómo el profesor asigna las tareas a los diferentes grupos del salón de clase. El tipo de asignación tendrá influencia en el proceso de aprendizaje de la programación; ya que entre más común es una tarea en todo el salón, mayor será la comprensión y dominio del tema al momento de discutir las diferentes soluciones de los grupos. Basado en Scott et al. (1994), se presentan tres técnicas de asignación de tareas a los grupos que son descrito a continuación.

Asignación de la misma tarea (A1)

este tipo de asignación es aquella en donde una misma tarea es asignada a todos los grupos formados en el salón de clases. Sus beneficios son reflejados al momento de realizar la discusión de las diferentes soluciones de los grupos, en donde será de competencia para todos en el salón de clases, ya que se posee mayor comprensión y dominio del problema a tratar. Ésto permitirá afianzar aun más el conocimiento del estudiante al evaluar dichas soluciones de un problema que ha tratado con su propia solución. La principal desventaja es que limita la obtención de nuevos conocimientos ya que solo se discute un mismo problema y no varios. La utilización de esta asignación puede depender de las capacidades en el salón de clases; ya que si se tiene un salón de clases donde los estudiantes poseen un dominio pobre de un tema de programación, se puede considerar la utilización de esta técnica de asignación debido a que reforzaría el dominio de dicho tema al tratarse el mismo en el momento de la discusión general.

Asignación de subtareas (A2)

esta técnica de asignación representa la manera en que el profesor asigna diferentes tareas a los grupos; dichas tareas conforman una parte de un proyecto general. De esta manera, los grupos trabajan en tareas o módulos que al final deben ser unidas en un solo programa desarrollado en el salón. Este tipo de asignación ayuda a generar nuevos conocimientos ya que los estudiantes estarán consciente del trabajo

de los demás grupos, además de ver como sus módulos interaccionan con los otros; en donde se discute y evalúa, de manera general, un proyecto de mayor envergadura. La utilización de ésta técnica puede traer grandes ventajas en aquellas clases donde el nivel de capacidad sean parecidas; ya que la falta de dominio de los temas tratados por parte de los estudiantes entorpece la comprensión general del proyecto desarrollado. Por lo que aquellos estudiantes con deficiencias menores serán los que aprovechen ésta técnica en la generación de conocimientos y los de mayor deficiencia podría causar su frustración e incrementar la distancia de capacidades.

Asignación de diferentes tareas (A3)

la utilización de esta técnica implica que el profesor asigna a cada grupo formado en el aula de clase, tareas diferentes que cada uno debe desarrollar. Éste tipo de asignación puede generar mayor conocimiento dentro de los estudiantes ya que evaluarán y discutirán soluciones de diferentes problemas al momento de la exposición de dichos proyectos. La desventaja de la utilización de este tipo de asignación radica en el hecho de que si los estudiantes no poseen los conocimientos o capacidades deseadas solo el grupo que proporcionó dicha solución será la que aproveche los conocimientos al momento de discutirse; por lo que su efectividad dependerá de la similitud en los niveles de capacidades de los estudiantes; es decir, que en la medida que todos compartan la mismas capacidades estarán aptos para asimilar las soluciones de las diferentes tareas expuestas.

B.1.2. Tamaño del grupo de trabajo

Al momento de formar un grupo de estudiantes es de vital importancia tomar en cuenta el número de los mismos, ya que ésto agrega un grado de dificultad en el trabajo que se debe desarrollar de manera colaborativa; la conformación de grupos de 2 puede facilitar las relaciones colaborativas ya que no es necesario definir un líder ni llevar una estructura compleja de operación a diferencia de los grupos de mayor cantidad; de esta manera, el éxito de los grupos que sean mayores de 8 ó 10 dependerá estrictamente de las asignaciones de tareas, estructura organizacional, asignaciones de tareas, conformación de subgrupos, entro otros Scott y Cross (1995). Sin embargo,

en aquellos grupos pequeños de 3 a 5 estudiantes son los más utilizados dentro del salón de clases ya que son más manejables y se le puede extraer mayor provecho de aprendizaje. Los grupos de 6 a 9 personas tienen iguales características que los grupos pequeños pero presentan mayor dificultad para su estructura operacional y en las relaciones interpersonales Scott y Cross (1995).

B.1.3. Tamaño y complejidad de las tareas

El tamaño y la complejidad de una tarea tiene influencia directa en el éxito de los grupos de trabajo dentro del proceso de enseñanza-aprendizaje de la programación Scott et al. (1994); ya que si a un grupo de estudiantes se le asigna una tarea que está por arriba de las capacidades del grupo, es muy probable que pueda causar un sentimiento de frustración y no se obtendrá los resultados deseados. Tanto el tamaño como la complejidad de una tarea llevan la connotación de tiempo requerido para ser completado. A continuación se describirá el tamaño y la complejidad de una tarea en donde se indicará posibles niveles de implementación.

a- Tamaño de las tareas: el tamaño de las tareas se refiere a qué tan extenso es el trabajo que debe realizarse al asignársele una tarea determinada a un grupo de estudiantes, sin importar la complejidad del mismo; es decir, que se evaluará el esfuerzo físico requerido; ya sea por el número de líneas de código u otras actividades. De esta manera, el tamaño de una tarea dependerá del número de estudiantes que conformen el grupo y de las capacidades que se posean. La siguiente categorización del tamaño de las tareas está basado en el número de estudiantes que pueden realizarla con éxito siempre y cuando compartan las mismas capacidades.

- *Tamaño de tarea individual (T0):* esta tarea tiene tamaño suficiente como para que un estudiante, de manera individual, pueda llevarla a cabo con éxito.
- *Tamaño de tarea en grupo de par (T1):* el tamaño de ésta tarea es suficiente como para ser realizada por un grupo conformado por dos estudiantes.
- *Tamaño de tarea para grupo (T2):* este tipo de tarea describe su tamaño o extensión de trabajo para un grupo conformado por 4 ó 5 estudiantes.

b- Complejidad de las tareas: la complejidad de las tareas describe que tanto tiempo de análisis es requerido para que uno o varios estudiantes realicen una tarea asignada; por lo que se deja claro que el esfuerzo mental será evaluado para la categorización de los mismos. Por consiguiente, la complejidad de una tarea dependerá del número de estudiantes que conformen el grupo y de las capacidades que se posean; entre mayor sea la el mismo nivel de capacidad mayor será la efectividad en el proceso de análisis. La categorización para la complejidad de las tareas se basará en el número de estudiantes con iguales capacidades que puedan trabajar de manera colaborativa para el análisis y solución de una tarea asignada.

- *Complejidad de tarea individual (C0):* la complejidad de esta tarea está dirigida para ser desarrollada por un solo estudiante.
- *Complejidad de tarea para grupo (C1):* dichas tareas presentan suficiente complejidad como para ser realizadas por un grupo de 2 ó 3 estudiantes.
- *Complejidad de tarea para grupo mayor (C2):* estas tareas presentan complejidad suficiente como para ser realizadas en un grupo conformado por 4 ó 5 estudiantes.

La categorización del tamaño y complejidad de las tareas es llevada a cabo a través de la experiencia de los expertos.

B.2. Capacidades de los estudiantes

En el aula de clases las capacidades (habilidades cognitivas) de los estudiantes pueden variar considerablemente, debido a factores externos e internos que influyen en ellos como lo es: formación académica previa, conocimientos generales, problemas personales, familiares y sociales, entre otros Cox y Fisher (2009). Por lo que se hace imprescindible tomar en cuenta éstas variaciones a la hora de utilizar los grupos de trabajo.

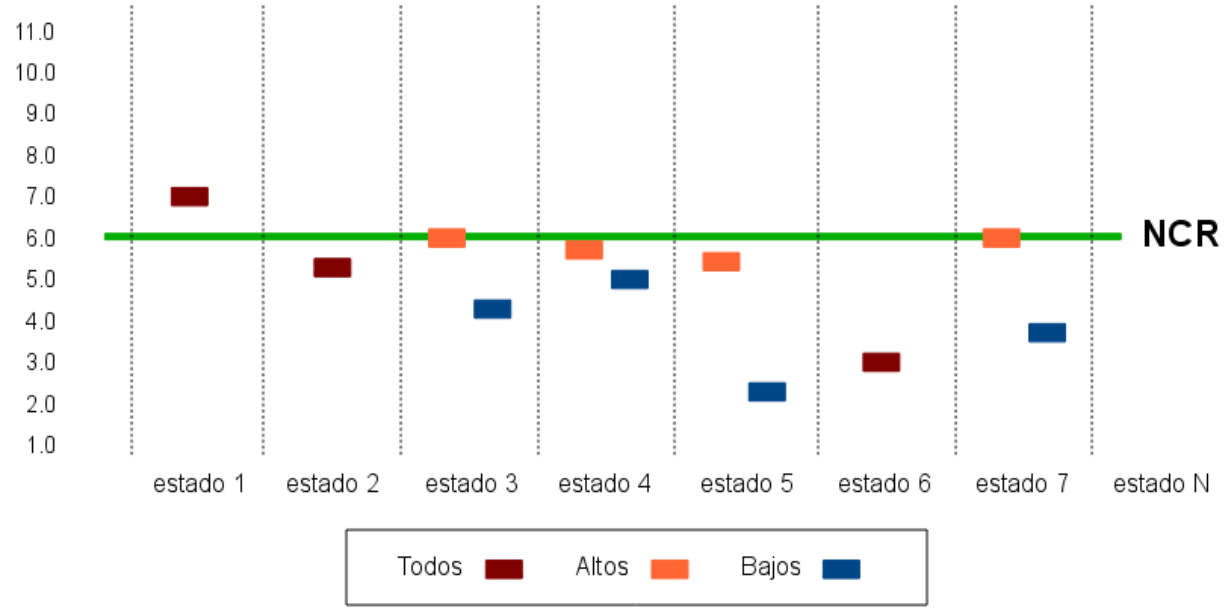
Al tomar en cuenta los niveles de capacidades dentro del salón de clases se evitará que los estudiantes que posean los niveles bajos se frustren al no poder estar a

la altura de los demás en las asignaciones grupales; de esta manera, se ayudará a reforzar y mejorar los conocimientos de los estudiantes de bajo rendimiento al recibir la asistencia de los de mayor capacidad.

En la figura B.1, se presenta un esquema de algunos de los posibles estados dentro del aula de clase en donde se distribuyen las capacidades de los estudiantes con respecto al nivel de capacidad de referencia (NCR) que es el deseado para una asignación o materia. Los diferentes estados de las capacidades están planteadas en *todos, altos* y *bajos*. *Todos* describe que todos los estudiantes poseen iguales o parecidas capacidades, es decir, que se encuentran en un mismo rango de capacidades sin importar el nivel del mismo; *altos* indica el conjunto de estudiantes que poseen conocimientos mayores al resto y *bajos*, los estudiantes de menor capacidades; en donde la suma de los *altos* y *bajos* son los *todos*. Cabe destacar que la representación de este esquema no esta basado en criterios formales, solo se considera dividir a los estudiantes en los conjuntos de altas y bajas capacidades utilizando alguna ponderación para ello; de ésta manera, si dentro de un rango de ponderación existe un número de estudiantes por arriba de un porcentaje estipulado (ejemplo: más del 90 % del total de los estudiantes se encuentran en un rango determinado), se considera un *todos* en la distribución. El nivel del *todos* se ubica a través del cálculo de una moda; de igual forma, la moda es utilizada para la ubicación del nivel de *altos* y *bajos*. Por otro lado, los conjunto de *todos* que se encuentren por arriba del NCR pudieron haber sido calculado ya sea utilizando la forma normal ya explicada o tomando el nivel bajo de capacidades y decir que es el nivel todos; esto se debe a que no importa el nivel alto ya que se encuentra por encima del NCR. Los niveles altos que se encuentren por encima del NCR se especifican en el mismo nivel de NCR, siempre y cuando los del nivel bajo se encuentren por debajo del NCR; ya que lo importante es que todos los estudiantes tengan el mismo nivel de capacidades y que se sea el del NCR o más.

En el esquema de la figura a tampoco se considera el porcentaje de estudiantes que se encuentran dentro de cada conjunto; esto se considerará en secciones posteriores. Por otro lado, el NCR que se observa, es a manera de ejemplo ya que puede variar a lo largo de la escala o considerarse otra escala. En cuanto al número de estado, se incluyeron solo algunos ya que se considera que pueden existir muchas combinaciones de capacidades y generar distintos estados.

Figura B.1: Esquema de los posibles estados de distribución de capacidad en el salón de clases.



B.3. Esquema de decisión para la formación de grupos

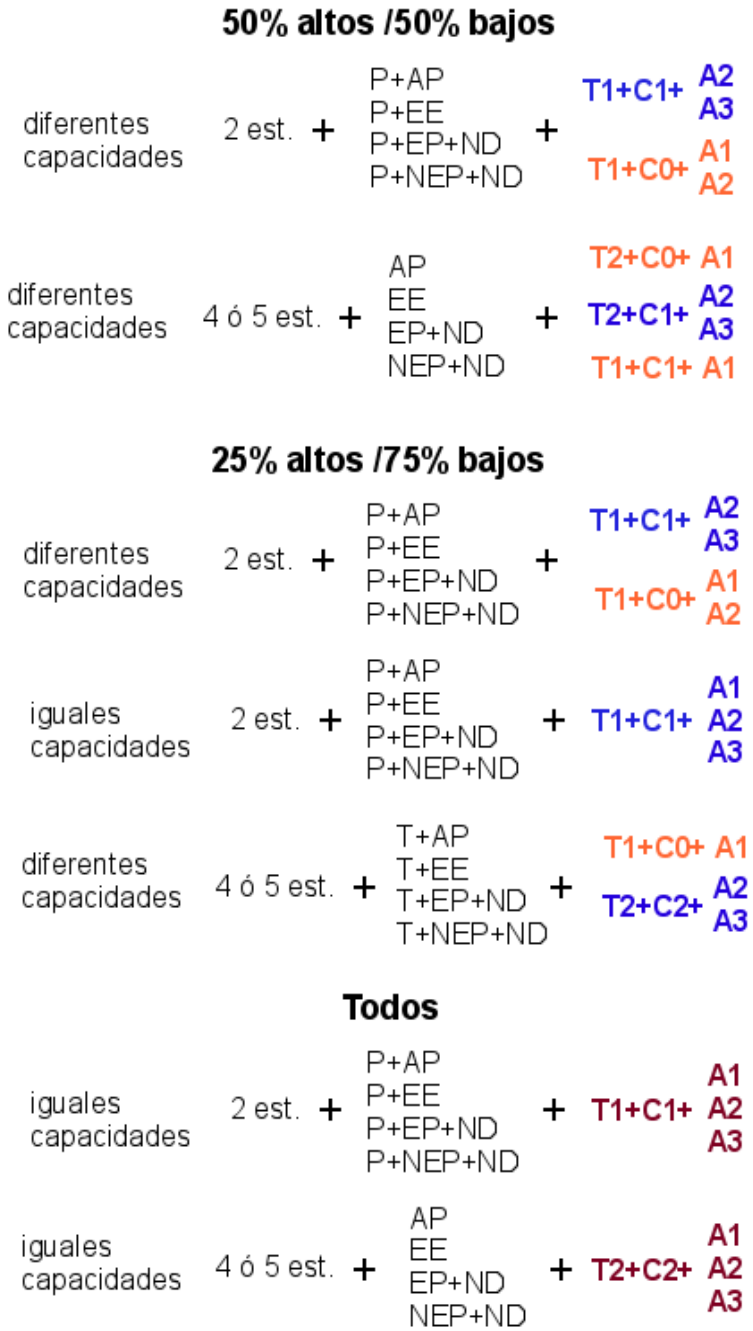
El esquema de decisión para la formación de grupos está basado en el conjunto de consideraciones y criterios que fueron expuesto con anterioridad; en donde se planteó las características, ventaja y desventajas y su posible implementación de ciertas técnicas relacionadas al método de trabajo en grupo. Este esquema de decisión permitirá la utilización optima del método de formación de grupos ya que tomará en cuenta el estado de los estudiantes dentro del aula de clases; ésto en su mayoría tiene que ver con los niveles de capacidades que fueron evaluadas en el punto anterior.

Para poner en práctica el esquema de decisión es necesario tomar en cuenta previamente los criterios y consideraciones de la sección 3.2.5; ya que es imprescindible que los estudiantes posean una cultura y habilidades de trabajo grupal. Partiendo de que los estudiantes posean dichas habilidades y que sean lo suficientemente marcadas como para no entorpecer el aprendizaje dentro de ambientes colaborativos, se propone el siguiente esquema de decisión que puede ser observado en la figura B.2. En este esquema se ha utilizado las abreviaturas y notaciones de las técnicas expuestas en secciones anteriores; además de las abreviaturas de los tipos de grupos

descritos en la sección 3.2.4. Por otro lado, solo se contempla dos clases de grupos basado en el número de miembros, un grupo formado por 2 y otro formado por 4 ó 5 estudiantes como fue explicado en la sección B.1.2. Este esquema posee tres posibles casos de distribución de capacidades basada en la sección B.2, los cuales son: *50 % altos/50 % bajos*, *25 % altos/75 % bajos* y *Todos*. *50 % altos/50 % bajos* representa que el número estudiantes de capacidades altas y de bajas se encuentran en iguales proporciones; para este caso, la formación de grupo permitirá la utilización de *Grupos de par (P)* en la que se posean estudiantes de diferentes capacidades, en donde las asignaciones se basaran en el del nivel bajo para que el estudiante del nivel alto tenga dominio del tema y trasmita conocimientos al necesitar del otro por la extensión del problema; también, la asignación puede estar basada en el mismo nivel del alto, pero extensión mayor; el nivel de altos, bajos o todos que se considera se encuentra representada en éste esquema, a través de los colores que fueron presentados en el esquema de la figura B.1. Otra formación de grupo sugerida será los grupos de mayor extensión que estarán conformados por estudiantes de capacidades altas y bajas en iguales proporciones; los estudiantes en este grupo pueden elegir sus compañeros (EE) siempre y cuando se cumpla con la propiedad del grupo, para ello se les indicará cuáles son los compañeros que pueden ser elegidos; de igual forma, pueden ser elegidos de manera aleatoria por el profesor (AP). La asignación de estos grupos puede tener tres combinaciones de especificación y cada uno busca incrementar la participación de los de nivel bajo en la generación de conocimientos. Esto se logra al asignarles tareas con suficiente extensión para que los del nivel alto no puedan realizar solo y requieran la ayuda de los del nivel bajo y así puedan transmitir conocimiento a los mismos; la transmisión de éstos conocimientos será efectiva siempre y cuando los estudiantes del nivel alto dominen el tema de la asignación presentada, por lo que las combinaciones propuestas contemplan ésta consideración.

Otro caso de distribución es el *25 % altos/75 % bajos*, en donde la mayoría de los estudiantes poseen bajo nivel de capacidad con respecto al resto que pueden representar un 25 %. En este caso de distribución se puede utilizar la formación de *Grupos de par (P)*. La formación de este tipo de grupo contendrá grupos con iguales y diferentes capacidades; los de iguales capacidades solo pueden estar conformado por los de nivel bajo. La asignación de las tareas de ambos grupos puede estar basada en las capacidades del nivel bajo; solo se considerará el nivel alto para las asignaciones en los grupos conformado por diferentes capacidades y tendrá el mismo beneficio y propiedades de los *Grupos de par (P)* formados en el caso de distribución *50 %*

Figura B.2: Esquema de decisión para la formación de grupos de trabajos en la enseñanza de la programación.



altos/50 % bajos. Otro tipo de grupo que puede ser formado con ésta distribución son los basado en tutores (T), dichos tutores van ha ser los estudiantes que representen el 25 % de la distribución. Se propone dos combinaciones de asignación de tareas, uno basado en los niveles del bajo y extensión mayor perteneciente a grupos de 4 ó 5 miembros y el otro, en los niveles del alto en donde la dificultad será la de la individual (*C0*) y su tamaño de *T1* como se puede ver en la figura B.2; ya que la responsabilidad recae sobre el tutor y este debe poseer total dominio del tema, para así transmitir los conocimientos al resto del grupo en donde se asegurará participación a través de las extensiones propuestas. De igual forma, se puede permitir la elección de los miembros por parte de los estudiantes (EE) especificando los tutores por grupo o bien la elección de manera aleatoria por parte del profesor (AP).

Todos como se ha mencionado, representa que todos los estudiantes en el aula de clase poseen iguales o parecidas capacidades. Este caso de distribución los grupos de formación pueden ser de par o grupos completo de 4 ó 5 estudiantes en la que los estudiantes pudieran elegir sus propios compañeros o bien, el profesor elegirlos de manera aleatoria. Las tareas asignadas se presentan en su máxima capacidad de acuerdo a su nivel, ya que se considera que los grupos de iguales capacidades pueden rendir más. De esta manera, entre más los grupos se asemejan en capacidades mejor es el trabajo desarrollado y los conocimientos generados.

Para todos los tres casos presentados, el tipo de asignación de tareas esta basado en las consideraciones expuestas de la sección B.1.1, en donde podemos ver que para unos casos algunos tipos de asignaciones son utilizadas y para otros no, en el caso de *Todos*, los tres tipos de asignaciones pueden ser presentadas a los estudiantes, en donde se les puede sacar el máximo provecho a las asignaciones de diferentes tareas para cada grupo (*A3*). Por otro lado, se considera en la conformación de los grupos, los estilos de programación iguales (EP) o diferentes (NEP) y la diferencia en cuanto a las deficiencias de los estudiantes (ND), en donde sea posible siempre y cuando no se pierda las propiedades antes mencionadas. Para el caso en que se considere una elección aleatoria (AP) o que los estudiantes elijan sus compañeros (EE) no puede ser considerada la elección basado en estilo de programación y deficiencias por lo que puede seguir siendo un opción; dependiendo de los beneficios que se quieran buscar y de la situación actual del salón de clases.

Referencias

AROSEMENA-TREJOS, D., CRESPO, S. y CLUNIE, C. Criterios y consideraciones para la formacion de grupos de trabajo en la ensenanza de la programacion. En *Edutec 2010: Proceedings of the Edutec 2010*. 2010. ISBN 978-84-9860-447-4. Http://2edutec2010.wordpress.com/.

BAGLEY, C. A. y C. CANDACE CHOU. Collaboration and the importance for novices in learning java computer programming. En *ITiCSE'07*, páginas 211–215. ACM, 2007.

BLACK, T. R. Helping novice programming students succeed. *JCSC 22, 2*, páginas 106–114, 2006.

CALZADILLA, M. E. Aprendizaje colaborativo y tecnologias de la informacion y la comunicacion. *Revista Iberoamericana de Educacion*, 2004.

CARTER, J. y JENKINS, T. Gender and programming: What's going on? En *ITiCSE '99 6/99*, páginas 1–4. 1999.

CASTRO, T., FUKS, H. y CASTRO, A. Programming in groups: a progression learning scheme from the individual to the group. En *38th ASEE/IEEE Frontiers in Education Conference*, páginas 15–20. IEEE Computer Society, 2008.

CHONG, J., PLUMMER, R., LEIFER, L., KLEMMER, S. R., ERIS, O. y TOYE, G. Pair programming: When and why it works. 2010. `http://www.ppig.org/papers/17th-chong.pdf`.

COX, A. y FISHER, M. Programming style: Influences, factors, and elements. En *2009 Second International Conferences on Advances in Computer-Human Interactions*, páginas 82–89. IEEE Computer Society, 2009.

CUBRANIC, D. y STOREY, M.-A. D. Collaboration support for novice team programming. En *GROUP'05*, páginas 136–139. ACM, 2005.

DE FARIA, E. S. J., ADÁN-COELLO, J. M. y YAMANAKA, K. Forming groups for collaborative learning in introductory computer programming courses based on students programming styles: An empirical study. En *36th ASEE/IEEE Frontiers in Education Conference*, páginas 6–11. IEEE Computer Society, 2006.

GRAY, S., LEWANDOWSKI, G., EDWARDS, S. y SHENDE, A. Improving student programming skills by developing program comprehension abilities. *JCSC 20, 3*, páginas 235–237, 2005.

GÓMEZ-PEREZ, A. y FERNÁNDEZ, M. Ontological engineering. *Springer*, 2004.

HAAHR, P. A programming style for java. 2010. http://192.220.96.201/essays/java-style/single-page.html.

HEFLIN, J. Owl web ontology language - use cases and requirements. 2010. http://www.w3.org/TR/webont-req/.

HEWITT, D. J. Diagnostico de la industria de tecnologias de informacion y comunicaciones de panama. 2010. http://www.capatec.org.pa/docs/SectorTICsdePanamaA-HEWITT.pdf.

HRISTOVA, M., MISRA, A., RUTTER, M. y MERCURI, R. Identifying and correcting java programming errors for introductory computer science students. En *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education*, páginas 153–156. ACM, New York, NY, USA, 2003. ISBN 1-58113-648-X.

III, T. V. T., LEE, R. y ALHSWEDE, T. Evaluating the usefulness of pair programming in a classroom setting. En *6th IEEE/ACIS Internacional Conference on computer and Information Science (ICIS 2007)*. IEEE Computer Society, 2007.

JACKSON, J., COBB, M. y CARVER, C. Identifying top java errors for novice programmers. En *35th ASEE/IEEE Frontiers in Education Conference*, páginas 24–28. IEEE Computer Society, 2004.

JUEDES, D. W. Web-based grading: Further experiences and student attitudes. En *35th ASEE/IEEE Frontiers in Education Conference*, páginas 18–23. IEEE Computer Society, 2005.

KAILA, E., RAJALA, T., LAAKSO, M.-J. y SALAKOSKI, T. Automatic assessment of program visualization exercises. En *Koli Calling'08*, páginas 101–104. ACM, 2005.

KO, A. J. y MYERS, B. A. Development and evaluation of a model of programming errors. En *the IEEE 2003 Symposia on Human Centric Computing Languages and Environments (HCC'03)*. IEEE Computer Society, 2003.

KOPEC, D., YARMISH, G. y CHEUNG, P. A description and study of intermediate student programmer errors. En *The SIGCSE Bulletin*, páginas 146–157. ACM, 2007.

LAHTINEN, E., ALA-MUTKA, K. y JARVINEN, H.-M. A study of the dificulties of novice programmers. En *ITiCSE'05*, páginas 14–18. ACM, 2005.

LOURIDAS, P. Static code analysis. *Software, IEEE*, vol. 23(4), páginas 58 –61, 2006. ISSN 0740-7459.

LUCERO, M. M. Entre el trabajo colaborativo y el aprendizaje colaborativo. *Revista Iberoamericana de Educacion*, 2004.

MCGUINNESS, D. L. y VAN HARMELEN, F. Owl web ontology language - overview. 2010. `http://www.w3.org/TR/owl-features/`.

MENA, A. A., MIRANDA, F. y SANGUINO, R. Desarrollo por etapas: Una nueva generacion de procesos de desarrollo. 1999.

MENGEL, S. A. y YERRAMILLI, V. A case study of the static analysis of the quality of novice student programs. En *Proceedings of SIGCSE 99*, páginas 78–82. ACM, New Orleans, LA, USA, 1999.

MILNE, I. y ROWE, G. Difficulties in learning and teaching programming - views of students and tutors. *2002 Kluwer Academic Publishers*, 2002.

MORIMOTO, Y., KURASAWA, K. y YOKOYAMA, S. A support system for teaching computer programming based on the analysis of compilation errors. En *the Sixth International Conference on Advanced Learning Technologies (ICALT'06)*. IEEE Computer Society, 2006.

NAMATAME, Y. Evaluation of the group study using peer review in the lesson of programming. En *C5 '04: Proceedings of the Second International Conference on Creating, Connecting and Collaborating through Computing*, página 190. IEEE Computer Society, Washington, DC, USA, 2004. ISBN 0-7695-2166-5.

NOVAK, J., KRAJNC, A. y ZONTAR, R. Taxonomy of static code analysis tools. En *MIPRO 2010*, páginas 418–422. the IEEE Computer Society, 2010.

NOY, N. F. y MCGUINNESS, D. L. Ontology development 101: A guide to creating your first ontology. 2005.

PILLAY, N. A study of object-oriented design errors made by novice programmers. En *SACLA '09*, páginas 101–104. ACM, 2009.

RAE. *Diccionario de la Lengua Española*. Vigésima segunda edición edición, 2001.

RAMALINGAM, V. y WIEDENBECK, S. An empirical study of novice program comprehension in the imperative and object-oriented styles. 1996.

SCOTT, T. J. y CROSS, J. H., II. Team selection methods for student programming projects. En *Proceedings of the 8th SEI CSEE Conference on Software Engineering Education*, páginas 295–303. Springer-Verlag, London, UK, 1995. ISBN 3-540-58951-1.

SCOTT, T. J., TICHENOR, L. H., RALPH B. BISLAND, J. y II, J. H. C. Team dynamics in student programming projects. En *SIGSCE 94-3/94*. Springer-Verlag, 1994.

TAHERKHANI, A., MALMI, L. y KORHONEN, A. Algorithm recognition by static analysis and its application in students submissions assessment. En *Koli Calling '08*, páginas 88–92. ACM, 2008.

TRUONG, N., ROE, P. y BANCROFT, P. Static analysis of students java programs. En *fixth Australian computing Education Conference (ACE2004)*, páginas 317–325. ACM, 2009.

W3C. Java programming style guide. 2010. `http://www.w3c.es/divulgacion/-guiasbreves/tecnologiasxml`.

WALLE, T. y HANNAY, J. E. Personality and the nature of collaboration in pair programming. En *Thrid International Symposium on Empirical Software Engineering and Measurement*, páginas 203–213. ACM, 2009.

WHEATON, P. Java programming style guide. 2010. `http://www.javaranch.com/styleLong.jsp`.

YARMISH, G. y KOPEC, D. Revisiting novice programmer errors. En *The SIGCSE Bulletin*, páginas 131–137. ACM, 2007.

Anexo 1

UNIVERSIDAD TECNOLÓGICA DE PANAMÁ

Tabla de Calificaciones de Desarrollo II

	A	B	C	D	F	I	N	R	Total
Total	**121**	**177**	**180**	**90**	**38**	**58**	**115**	**20**	**799**
AZUERO	**2**	**1**	**15**	**3**	**8**	**0**	**6**	**0**	**35**
2009	**2**	**1**	**15**	**3**	**8**	**0**	**2**	**0**	**31**
2010	**0**	**0**	**0**	**0**	**0**	**0**	**4**	**0**	**4**
BOCAS DEL TORO	**2**	**1**	**7**	**1**	**0**	**0**	**1**	**0**	**12**
2009	**2**	**1**	**7**	**1**	**0**	**0**	**1**	**0**	**12**
CHIRIQUI	**22**	**18**	**17**	**24**	**5**	**6**	**14**	**1**	**107**
2009	**9**	**6**	**5**	**15**	**5**	**5**	**3**	**0**	**48**
2010	**13**	**12**	**12**	**9**	**0**	**1**	**11**	**1**	**59**
COCLE	**1**	**3**	**15**	**4**	**9**	**2**	**5**	**0**	**39**
2009	**1**	**3**	**4**	**2**	**6**	**0**	**4**	**0**	**20**
2010	**0**	**0**	**11**	**2**	**3**	**2**	**1**	**0**	**19**
COLON	**2**	**2**	**10**	**0**	**0**	**4**	**3**	**1**	**22**
2009	**0**	**1**	**7**	**0**	**0**	**0**	**3**	**0**	**11**
2010	**2**	**1**	**3**	**0**	**0**	**4**	**0**	**1**	**11**
PANAMA	**76**	**127**	**101**	**44**	**13**	**41**	**75**	**18**	**495**
2009	**44**	**63**	**43**	**13**	**3**	**21**	**40**	**14**	**241**
2010	**32**	**64**	**58**	**31**	**10**	**20**	**35**	**4**	**254**
PANAMA OESTE	**7**	**12**	**8**	**2**	**0**	**2**	**5**	**0**	**36**
2009	**5**	**7**	**4**	**2**	**0**	**1**	**3**	**0**	**22**
2010	**2**	**5**	**4**	**0**	**0**	**1**	**2**	**0**	**14**
VERAGUAS	**9**	**13**	**7**	**12**	**3**	**3**	**6**	**0**	**53**
2009	**6**	**5**	**1**	**3**	**0**	**0**	**4**	**0**	**19**
2010	**3**	**8**	**6**	**9**	**3**	**3**	**2**	**0**	**34**

Fuente: Información proporcionada por la Ing. Yadira Vargas de la Dirección de Tecnología de Información y Comunicación (DITIC). Universidad Tecnológica de Panamá.

Anexo 2

UNIVERSIDAD TECNOLÓGICA DE PANAMÁ
MAESTRÍA EN CIENCIAS DE TECNOLOGÍA DE INFORMACIÓN Y COMUNICACIÓN
CASO DE ESTUDIO
PROBLEMA DE PROGRAMACIÓN No. 1

Resolver el siguiente problema:

1. Cree un programa en Java que acepte como parámetros de entrada un nombre y la edad de la persona; el programa debe imprimir seguidamente, si es niño, adolescente, adulto o de tercera edad. Esto basado en los siguientes criterios:

   - infante - menor de 10.
   - adolescente - menor de 18.
   - adulto - menor de 60.
   - De tercera edad - mayor de 60.

   El programa aceptara nombres y edades hasta que se logre obtener, después de 3 repeticiones, el porcentaje o más de personas mayores de edad; este porcentaje dependerá de un parámetro de entrada que el usuario ingresará y consiste en un número entero del 1 al 3 que tendrá el siguiente significado: 1=50%, 2=60%, 3=70%. De esta manera, si el usuario ingresa el número 3, se debe obtener un 70% o más, de entradas mayores de edad; es decir, si después de 3 repeticiones, las entradas de mayores de edad no representa el 70% o más, el programa debe seguir repitiendo el proceso de entrada hasta que el número de mayores de edad cumpla con la condición, tomando en cuanta todas las entradas anteriores sin utilizar vectores u otras estructuras de almacenamiento.

   Una vez terminada las entradas, imprimir el promedio de edades, la edad mayor y menor.

   Este programa debe ser resuelto en Java y en un solo archivo .java en donde se debe incluir, en modo de comentario, el nombre del estudiante, cédula, código de ejercicio=15, materia. Todo esto con el siguiente formato.

```
/*
  @estudiante: Juan Perez
  @ID: 8-434-2345
  @codigo: 15
  @materia:  Desarrollo de Software II
*/
```

   Para la entrega del programa, se solicita enviar solamente el archivo con extensión .java a la siguiente dirección: davis.arosemena@utp.ac.pa.

Anexo 3

UNIVERSIDAD TECNOLÓGICA DE PANAMÁ
MAESTRÍA EN CIENCIAS DE TECNOLOGÍA DE INFORMACIÓN Y COMUNICACIÓN
CASO DE ESTUDIO
PROBLEMA DE PROGRAMACIÓN No. 2
(ORIENTADO A OBJETOS)

Resolver el siguiente problema:

1. Crear un programa utilizando Programación Orientada a Objetos para la administración de grupos de estudiantes en un salón de clases. Cada grupo se le debe especificar un número único de grupo que aumentara secuencialmente. De igual forma, contendrá los estudiantes asociados al mismo especificando un líder de grupo. Las clases deben permitir operaciones tales como: impresión de los nombres de los estudiantes con sus datos generales (cédula, dirección, edad) por grupo, adición o eliminación de estudiantes en los grupos, editar información de los estudiantes y de los grupos, imprimir el número de estudiantes por grupo, promedio de edad de los estudiantes y otras operaciones básicas.

Este programa debe ser resuelto en Java en uno o varios archivos .java; debe incluir, en modo de comentario, el nombre del estudiante, cédula, código de ejercicio=16, materia. Todo esto con el siguiente formato.

```
/*
  @estudiante: Juan Perez
  @ID: 8-434-2345
  @codigo: 16
  @materia:  Desarrollo de Software II
*/
```

Para la entrega del programa, se solicita enviar solamente los archivos con extensión .java a la siguiente dirección: davis.arosemena@utp.ac.pa.

Anexo 4

Código fuente de estudiantes para enunciado del Anexo 5

```
/*
 @estudiante: Jairo Ramos
 @ID: 8-534-3346
 @materia: Desarrollo de Software
 @codigo: 15
*/


import javax.swing.JOptionPane;
class ProblemaUno
{
private String nombre;
private int edad;
private int vec;
void asignar(String n,int edad1,int veces1)
{
nombre=n;
edad=edad1;
vec=veces1;
}
String determinar()
{ String resp="";
  if(edad<10)
  resp+=nombre+"          "+edad+"           "+"infante";
  else if((edad>=10)&(edad<18))
  resp+=nombre+"          "+edad+"           "+"adolescente";
  else if((edad>=18)&(edad<60))
  resp+=nombre+"          "+edad+"           "+"adulto";
  else if(edad>59)
  resp+=nombre+"          "+edad+"           "+"tercera edad";
  return resp;
}
int calcular(int total1)
    { int tot=total1;
      int resp=0;
  if(vec/tot*100==0.5)
  resp=1;
  else if(vec/tot*100==0.6)
  resp=2;
  else if(vec/tot*100==0.7)
  resp=3;
```

```
 return resp;
}
double promediar(int cantidad1,int tedad)
{   double can=cantidad1;
    double t=tedad;
double resp=0;
resp=t/can;
return resp;
}
}
class Princi
{
    public static void main(String arg[])
{ ProblemaUno obj=new ProblemaUno();
  String n="",result="";
  String titulo="Nombre          Edad          Etapa\n";
  int Por=0,veces=0,result2=0,cantidad=0,edad1=0,edad2=0,edad3=0,edadt=0;
  double result3=0;
  Por=Integer.parseInt(JOptionPane.showInputDialog(null,"1 . 50% de Personas Mayores
  \n 2. para el 60% \n 3 .para el 70%","Problema #1",JOptionPane.QUESTION_MESSAGE));
  for(int c=1;c<=3;c++)
   {
  n=JOptionPane.showInputDialog(null,"Introduzca el nombre de la persona","Problema
  #1",JOptionPane.QUESTION_MESSAGE);
  edad1=Integer.parseInt(JOptionPane.showInputDialog(null,"Introduzca su
  edad","Problema #1",JOptionPane.QUESTION_MESSAGE));
  if(edad2<edad1)
  edad2=edad1;
  else
  edad3=edad1;
  edadt+=edad1;
      if(edad1>=18)
      {
        veces+=1;
      }
  obj.asignar(n,edad1,veces);
  result+=obj.determinar();
  result2=obj.calcular(c);
  JOptionPane.showMessageDialog(null,"                    Lista de
  Personas\n"+titulo+result,"Problema #1",JOptionPane.INFORMATION_MESSAGE);
   }
  cantidad=3;
  if(result2!=Por)
  {
  do
   {
    cantidad+=1;
    n=JOptionPane.showInputDialog(null,"Introduzca el nombre de la persona","Problema
```

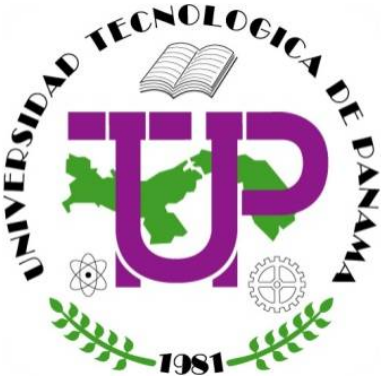
```
    #1",JOptionPane.QUESTION_MESSAGE);
    edad1=Integer.parseInt(JOptionPane.showInputDialog(null,"Introduzca su
    edad","Problema #1",JOptionPane.QUESTION_MESSAGE));
     if(edad2<edad1)
         edad2=edad1;
     else
         edad3=edad1;
    edadt+=edad1;

        veces+=1;

    obj.asignar(n,edad1,veces);
    result+=obj.determinar();
    result2=obj.calcular(cantidad);
    JOptionPane.showMessageDialog(null,"               Lista de
    Personas\n"+titulo+result,"Problema #1",JOptionPane.INFORMATION_MESSAGE);
   }while(result2==Por);
  }


  result3=obj.promediar(cantidad,edadt);
  JOptionPane.showMessageDialog(null,"               Lista de
  Personas\n"+titulo+result+"\n"+"Promedio de Edades: "+result3+"\n"+"La mayor edad
  fue de: "+edad2+"\n"+"La menor edad fue de: "+edad3,"Problema
  #1",JOptionPane.INFORMATION_MESSAGE);
  System.exit(0);
}
}
```

DAVIS ELIAS AROSEMENA TREJOS



GRUPOS DE TRABAJOS PARA LA ENSEÑANZA DE LA PROGRAMACIÓN BASADOS EN ONTOLOGÍA Y EN LA SEMÁNTICA DEL CÓDIGO FUENTE

Universidad Tecnológica de Panamá

Facultad de Ingeniería en Sistemas Computacionales

Maestría en Ciencias de Tecnología de la Información y Comunicación

Panamá

2012