

Universidad de las Ciencias Informáticas

Facultad 2



Solución Informática para la representación del corpus textual de archivos digitales

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias
Informáticas

Autores: Andis Eloy Yero Guevara

Pável Reyes Estévez

Tutor: Ing. Dasiel Cordero Morales

La Habana, junio de 2015

“Año 57 de la Revolución”

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores del trabajo de diploma “Solución Informática para la representación del corpus textual de archivos digitales” y autorizamos a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los ____ días del mes de junio del año 2015.

Andis Eloy Yero Guevara

Firma Autor

Pável Reyes Estévez

Firma Autor

Ing. Dasiel Cordero Morales

Firma Tutor

RESUMEN

La recuperación de información es el área del conocimiento mediante la cual se localiza y accede a los recursos de información que son pertinentes para la resolución de un problema determinado. Incluye la representación, el almacenamiento, la organización y el acceso a elementos de información. Uno de los problemas actuales en la recuperación de información es la representación del contenido de archivos digitales, debido a la necesidad de realizar actividades de clasificación de archivos digitales para la identificación y el agrupamiento de documentos semejantes con características comunes. En esta investigación se tiene como objetivo el desarrollo de una solución informática para la representación del corpus textual de archivos digitales textuales en contribución a la recuperación de información. Para el logro del objetivo se realiza un estudio de los antecedentes de los sistemas informáticos de recuperación de información y se prepara el entorno de desarrollo de software para resolver el problema en cuestión. Se utiliza el modelo de espacio vectorial que permite la representación del corpus textual de los archivos digitales mediante el álgebra de vectores. Basado en este modelo se diseña y desarrolla un algoritmo y se obtiene como resultado una matriz de semejanza que establece una medida de similitud entre los archivos textuales representados por vectores, lo que es necesario para la etapa de organización y clasificación automática de archivos digitales textuales así como tareas que incluyan la organización, clasificación, indexación y búsqueda en sistemas informáticos de recuperación de información.

Palabras clave: archivo digital textual, corpus textual, modelo de espacio vectorial, recuperación de información, representación de información.

ÍNDICE

INTRODUCCIÓN	1
CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA	9
Introducción	9
1.1 Aspectos teóricos de la investigación	9
1.2 Estudio de los antecedentes de la investigación	12
1.2.1 Modelos de recuperación de información.....	12
1.2.2 Soluciones existentes para la representación del corpus textual.	15
1.3 Metodología, Lenguajes y Herramientas de Desarrollo	19
1.3.1 Metodología de Desarrollo	19
1.3.2 Herramientas CASE	23
1.3.3 Lenguaje de programación y Entorno de desarrollo	23
Conclusiones del Capítulo.....	24
CAPÍTULO 2. ELEMENTOS FUNDAMENTALES DE LA PROPUESTA DE SOLUCIÓN.....	27
Introducción	27
Modelo de la propuesta de solución	27
Modelo de dominio	28
2.1 Planeación	29
2.1.1 Historias de Usuario	30
2.1.2 Tiempo de ejecución del proyecto	32
2.1.3 Iteraciones	32
2.1.4 Plan de entrega.....	33
2.2 Diseño de la propuesta de solución	34
2.2.1 Arquitectura de software	34
2.2.2 Diagrama de Clases.....	36
2.2.3 Patrones de diseño	37
2.2.4 Tarjetas Clase-Responsabilidad-Creador	42
2.3 Implementación de la Propuesta de Solución	43

2.3.1	Programación Concurrente.....	45
2.3.2	Estructuras para la representación del corpus textual	47
	Conclusiones del Capítulo.....	51
CAPÍTULO 3. COMPROBACIÓN DEL FUNCIONAMIENTO DE LA PROPUESTA DE SOLUCIÓN		
.....		53
	Introducción.....	53
3.1	Comprobación del funcionamiento del algoritmo implementado	53
3.2	Comprobación del funcionamiento del software desarrollado	56
	Estrategia de Prueba.	56
	Pruebas unitarias	57
	Pruebas de aceptación	63
	Conclusiones del Capítulo.....	67
CONCLUSIONES.....		68
REFERENCIAS.....		70
BIBLIOGRAFÍA		74

INTRODUCCIÓN

De forma natural, el hombre percibe la información del entorno y trata de aprovecharla para tomar decisiones acertadas, de lo contrario está en clara desventaja con sus semejantes. El desarrollo tecnológico de finales del siglo XX y comienzo del XXI ha perfeccionado las vías de procesamiento, transmisión y recopilación de la información. En este período dichos procesos se convirtieron en las fuentes fundamentales de la productividad debido a las nuevas condiciones tecnológicas (1).

La información es un conjunto de datos con un significado que constituye un mensaje sobre un determinado contexto, disponible para uso inmediato y que proporciona orientación en la toma de decisiones (2). Los datos son percibidos por los sentidos y procesados para crear el conocimiento necesario que es utilizado por el hombre en la toma de decisiones, en su interacción con el medio y sus procesos derivados. Para las empresas e instituciones es relevante la información generada en su quehacer diario que puede convertirse en un archivo de interés por su carácter histórico. Un archivo es un conjunto de documentos producidos o recibidos por toda persona, servicio u organismo, en el ejercicio de su actividad (3). Su importancia viene determinada por su uso; por lo que un buen manejo de la información para cualquier empresa, entidad, institución, gobierno o persona, fortalece la toma de decisiones y permite el logro de sus objetivos.

El uso de los archivos proviene desde la antigüedad; las civilizaciones más avanzadas contaban con archivos donde registraban sucesos astronómicos y matemáticos. En la edad media tuvo mayor utilidad, empleándose para esgrimir derechos legales sobre la propiedad de la tierra. En la edad moderna creció el interés por guardar todo tipo de documentos de origen histórico, de allí que se produzcan en todo el mundo esfuerzos serios para lograr ese fin. La preservación y organización de los archivos ha estado estrechamente relacionada con el devenir histórico y esto se debe en gran parte al carácter cultural, económico, político e histórico que tienen cada uno de ellos. Por tal motivo, surge la necesidad de darle un tratamiento adecuado con el fin de lograr una mejor comprensión, organización, recuperación de la información que contienen e identificar su importancia real para la organización.

El procesamiento de los archivos permite obtener información para la organización y trabajar en base a los resultados obtenidos con el fin de lograr mayor eficiencia en el desempeño de sus actividades (4). Con el auge de las Tecnologías de la Informática y las Comunicaciones (TIC) se ha acrecentado el papel fundamental de los procesos de digitalización y recuperación de la información para su preservación. Esto ha permitido que se extienda el uso de los archivos en formato digital, para un mejor tratamiento y consulta en el tiempo de su contenido. Un archivo digital es una unidad de datos o información almacenada en un medio digital para ser utilizada por aplicaciones de

computadora (5). Los mismos facilitan la aplicación de técnicas para el procesamiento e interpretación de su contenido. Además pueden ser almacenados por tiempo indefinido en perfectas condiciones y mediante el uso de las redes pueden ser consultados sin necesidad de desplazarse al lugar donde están almacenados.

Una vez digitalizados los archivos una de las tareas principales es la recuperación de la información que contienen, un problema que unido a las nuevas y robustas soluciones de almacenamiento, se agrava exponencialmente debido a la variedad de estructuras de los archivos existentes. La recuperación de información se define como el conjunto de tareas mediante las cuales se localiza y accede a los recursos de información para la resolución de un problema (6).

La recuperación de información incluye también la representación textual, el almacenamiento, la organización y el acceso a elementos de información (7). Esto permite la localización y presentación de información relevante ante la necesidad de conocimiento (8). Tradicionalmente existen tres modelos principales para la recuperación de información: el modelo booleano, basado en la teoría de conjuntos y álgebra de Boole, el modelo probabilístico que se basa en procesos estocásticos, operaciones de la teoría de la probabilidad y el teorema de Bayes y el modelo vectorial que opera mediante el álgebra de vectores. El problema de tales modelos estriba en su selección y mitigación de sus principales deficiencias con respecto a la representación del [corpus textual](#) de los archivos digitales.

El problema de la recuperación de información se analiza desde dos concepciones: una de ellas la constituye la pérdida tangible de la información por fallos en el almacenamiento, descuido de la organización, mal uso por parte de los usuarios o ataques malintencionados desde el exterior por hackers. Otra percepción es la recuperación de información con el fin de indexar, clasificar y agrupar registros vitales para las organizaciones; en esta última se enfoca la actual investigación. El proceso de recuperación de información está determinado por el tipo de información que se use y el medio de almacenamiento empleado. En el presente trabajo la forma de representar la información es la documental digital tal como: escritos, libros, revistas, documentos bibliográficos, cartas así como todos aquellos archivos digitales de carácter histórico que su contenido sea texto.

Los modelos y técnicas empleados en recuperación de información utilizan en algún momento recuento de frecuencias de los términos que aparecen en los datos y en las consultas. Usualmente, estos modelos se vuelcan en la representación de documentos para su búsqueda. Las mayores dificultades actualmente en la recuperación de documentos se encuentran en su derivación en términos, interpretación y representación, de su corpus textual. Dicha representación responde a la

implementación de un conjunto de criterios mediante los cuales se obtienen los términos y las relaciones entre estos. La representación del contenido del archivo digital textual, permite en términos de determinados algoritmos la descripción del mismo.

El exceso de información entrante en una entidad, como consecuencia del incremento masivo de fuentes documentales, entre otros aspectos, hace que el volumen de información esté muy por encima de la capacidad de desarrollo con la que suelen contar los recursos humanos de cualquier organización (10). Esto hace inminente la aplicación de técnicas de recuperación de información para tratar los archivos almacenados y obtener resultados relevantes del análisis de los eventos históricos registrados en ellos.

Para que cualquier tipo de representación textual pueda ser comprendida, debe estar estructurada. La estructura caracteriza el tipo de texto, independientemente de su contenido y es la que establece el esquema al que el texto se adapta (11). Los textos estructurados tienen la peculiaridad de estar enfocados a un área de estudio, además están divididos en secciones que permiten su interpretación. En los textos no estructurados ninguna parte del contenido es más importante que otra, por lo que la extracción e interpretación de sus palabras no es una tarea sencilla (11). Ejemplo de esto es cualquier tipo de texto desechable que no presente una estructura formal. La variedad de estructuras en los archivos históricos es un hecho tangible pues cualquier colección de los mismos podrá contener estructuras variadas y no definidas. Por tal motivo la investigación se centra en la recuperación de información en textos no estructurados.

La tarea de extracción del contenido de un archivo para su interpretación así como para tareas de clasificación archivística es normalmente un proceso largo y engorroso. Según el idioma de los archivos, la estructura que presenten, así como el objetivo que se desee lograr con la representación de su cuerpo textual, se dificultan ante la necesidad de diccionarios, expertos humanos y disponibilidad de espacio físico y digital para el almacenamiento de las colecciones de archivos. El avance tecnológico ha permitido, para agilizar este proceso la creación sistemas de cómputo denominados sistemas de recuperación de información (SRI).

Los SRI permiten identificar las fuentes de información relevantes a las áreas de interés así como analizar los contenidos de los documentos. Además facilitan la representación de los contenidos de las fuentes analizadas de una manera que sea adecuada para compararlas con las preguntas de los usuarios (12). Toda implementación de un SRI comienza con la tarea de procesamiento del corpus textual. Esto se debe a que no todos los términos que componen un documento son igualmente representativos de su contenido. Cuestiones como su posición, la cantidad de

ocurrencias o su función lingüística entre otras, definen el grado de importancia de cada uno de los términos. El resultado es una representación de la colección de palabras que lo componen, que es computacionalmente adecuada para los procesos siguientes a los modelos de recuperación de información.

Existen variadas soluciones descritas que de alguna forma tratan el problema de representación del cuerpo textual de los archivos. Muchas de estas soluciones informáticas son de plataforma Windows y realizan el análisis solamente de información estructurada (13) (14) (15) (16). Por tanto imposibilita su integración con otro software y limita la capacidad multiplataforma. Otras necesitan de un proceso de entrenamiento para determinar las clases en las que se dividen los documentos y solo analizan frases y palabras principales (14) (15). En cambio existen algunas que realizan el análisis lingüístico de textos pero no permiten comparar documentos, así como representar diversos contenidos simultáneamente (16) (17). Las más afines a la investigación tratan el problema de la representación textual de forma bastante acertada, pero basan su procesamiento en la comparación de una consulta con una colección y no entre colecciones de textos (18).

Estas soluciones son utilizadas principalmente en instituciones de gestión archivística, como medio para el perfeccionamiento de sus procesos. Como se ha expuesto, el desarrollo tecnológico guía los procesos de recuperación de información y representación del cuerpo textual de documentos de archivo a un continuo avance y con ello las organizaciones a nivel mundial se ajustan a tales condiciones.

Cuba desde hace varios años ha comenzado a dar pasos significativos en la digitalización y preservación de los archivos históricos de las instituciones competentes. La Universidad de las Ciencias Informáticas (UCI), centro educacional vinculado a la producción de software, está inmerso en esta tarea de vital importancia. Uno de sus centros productivos es el Centro de Informatización de la Gestión Documental (CIGED) que se dedica al desarrollo de sistemas y servicios informáticos para: Gestión de Documentos Administrativos, Gestión de Archivos Históricos y Gestión bibliotecaria y Centros de Documentación (19).

En este centro se desarrolla el sistema informático Xabal-Arkheia que contiene un conjunto de reglas generales para la descripción archivística que pueden aplicarse con independencia del tipo documental o del soporte físico de los documentos de archivo. El sistema maneja archivos sonoros, audiovisuales, fotográficos y textuales. De ellos, los archivos textuales brindan mayor información a partir del análisis de su contenido pues a pesar de ser fuente de información no estructurada, su análisis permite su organización por diversos criterios. Actualmente el sistema realiza la

digitalización de los documentos, pero no permite la representación del corpus textual de los archivos que maneja.

Existen un conjunto de inconvenientes derivados de la no representación textual del contenido digitalizado. Por tal motivo tareas como: indexación, extracción de metadatos, construcción del cuadro de clasificación archivística y organización de acuerdo al contenido se realizan de forma manual. Por otra parte impide la utilización de técnicas de resúmenes, [tesauros](#) u [ontologías](#) mediante descriptores que permitirán la búsqueda y recuperación de los documentos previamente indexados, así como la derivación del contenido en términos para su clasificación. Esto implica que se necesite de la intervención de un experto para la ejecución de estas tareas. Para las organizaciones esto trae consigo gastos en formación de personal, establecimiento de procedimientos que aseguren los criterios de categorización y auditorías de calidad lo que aumenta los costos y el tiempo de tratamiento de los documentos, sin que con ello se asegure la plena calidad de los procesos de obtención, recuperación, representación y utilización de la información.

Además para la organización del contenido el experto debe leer cada uno de los documentos lo que incurre en un costo en tiempo y esfuerzo elevado. Por tanto al aumentar considerablemente el cúmulo de información que almacenan los archivos, los recursos humanos disponibles no son suficientes afectando la capacidad de procesamiento y utilización de la información en la entidad competente. Por otro lado son inevitables los errores por parte del experto en la clasificación debido a las imprecisiones en los documentos o ambigüedades del lenguaje. Satisfacer múltiples consultas así como realizar un análisis de la información almacenada en un sistema mediado por indexación y clasificación manual con un gran volumen de información no permite en tiempo real obtener una respuesta acorde a las necesidades de las instituciones, organizaciones, gobiernos o personas.

Contar con un componente informático capaz de representar el cúmulo de archivos por su contenido, de manera que su representación se genere de forma automática, así como expresar el corpus textual de los documentos en términos e indexar los mismos utilizando modelos de recuperación de información es necesario para la etapa de organización y clasificación automática de los archivos digitales textuales.

De acuerdo a lo anteriormente expuesto se plantea el siguiente **problema a resolver**: ¿Cómo representar el corpus textual de los archivos digitales textuales en contribución a la recuperación de información?

El **objeto de estudio** del presente trabajo es la recuperación de información en archivos digitales.

De acuerdo al problema a resolver planteado se define como **objetivo general**: Desarrollar una solución informática para la representación del corpus textual de los archivos digitales textuales.

Se especifica como **campo de acción** la representación del corpus textual de los archivos digitales textuales.

Para dar solución al objetivo general se definen los siguientes **objetivos específicos**:

- ✓ Elaborar el marco teórico de la investigación a partir del análisis de los antecedentes a nivel nacional e internacional de las soluciones existentes orientadas a la representación del corpus textual de los archivos digitales textuales.
- ✓ Implementar un componente informático para la representación del corpus textual de archivos digitales textuales.
- ✓ Comprobar el funcionamiento del algoritmo y componente desarrollado a través de la realización de pruebas funcionales y de aceptación.

En aras de guiar la realización de la investigación, se definen las siguientes **preguntas de investigación**:

- ✓ ¿Cuáles son los conceptos principales en la recuperación de archivos digitales textuales?
- ✓ ¿Cuáles son los modelos y técnicas principales en la recuperación de información, ventajas y desventajas?
- ✓ ¿Cuál es el modelo más adecuado para la recuperación de información de archivos digitales textuales?
- ✓ ¿Cuáles soluciones informáticas existentes utilizan modelos de recuperación de información?
- ✓ ¿Cuáles son las principales estructuras de datos utilizadas para la representación del corpus textual de documentos digitales?
- ✓ ¿Qué estrategia es necesaria definir para comprobar el funcionamiento de la solución desarrollada?

Para dar cumplimiento al objetivo antes descrito se definen las siguientes **tareas de investigación**:

- ✓ Descripción de los conceptos fundamentales para la representación del corpus textual de documentos.
- ✓ Descripción de los diferentes modelos y técnicas de recuperación de información para la definición de los antecedentes de la investigación.

- ✓ Identificación de las principales soluciones informáticas que utilicen modelos o algoritmos de recuperación de información para la definición de los antecedentes de la investigación.
- ✓ Selección de un modelo de recuperación de información para la representación del corpus textual de archivos digitales.
- ✓ Selección de un algoritmo basado en un modelo de recuperación de información para la representación del corpus textual de archivos digitales.
- ✓ Selección de la metodología de desarrollo de software, herramientas, lenguajes y tecnologías a utilizar para el desarrollo de la solución informática que permita la representación del corpus textual de archivos digitales.
- ✓ Selección de las estructuras de datos para la representación del corpus textual de los documentos digitales.
- ✓ Selección de técnicas que mejoren el rendimiento computacional para disminuir el tiempo de corrida y respuesta del algoritmo seleccionado en la representación del corpus textual de documentos digitales.
- ✓ Comprobación del funcionamiento del algoritmo para demostrar la correcta representación del corpus textual.
- ✓ Definición de una estrategia de prueba para comprobar el funcionamiento del componente desarrollado.

En el desarrollo de la investigación se utilizan los siguientes **métodos de investigación científica**.

Métodos teóricos

Analítico-Sintético: este método fue utilizado para estudiar todas las teorías y documentos referentes a la representación del corpus textual de archivos digitales textuales, así como la extracción de los elementos y conceptos necesarios para el desarrollo de la investigación.

Histórico-Lógico: este método fue utilizado para estudiar los sistemas relacionados con la recuperación y clasificación de archivos digitales textuales, lo que permitió adquirir conocimiento sobre la forma en que se realizan estas tareas para la definición de los antecedentes de la investigación.

Modelación: este método fue utilizado para la modelación del algoritmo seleccionado para la representación del corpus textual de archivos digitales. Además su utilización permitió la modelación de la propuesta de solución, la creación de los diagramas de clases para representar los componentes y relaciones del sistema a desarrollar.

Métodos empíricos

Simulación: este método fue utilizado para la ejecución del algoritmo y componente implementados mediante el uso de datos artificiales de prueba, lo que permitió comprobar su funcionamiento para garantizar la calidad de ambos.

Análisis estático: este método fue utilizado para realizar un examen de la estructura del componente implementado. Su utilización permitió además la aplicación de pruebas unitarias y de aceptación al componente para detectar y corregir errores.

El presente documento está compuesto por introducción, desarrollo, conclusiones y bibliografía. El contenido del desarrollo está estructurado en tres capítulos que desarrollan su contenido de la siguiente manera:

Capítulo 1. Fundamentación teórica: Expone los elementos teóricos de la investigación. Se realiza el estudio e investigación de soluciones informáticas existentes para la representación textual de archivos digitales textuales. Se determinan la metodología de desarrollo, herramientas y tecnologías que se utilizan para desarrollar la solución propuesta.

Capítulo 2. Elementos fundamentales de la propuesta de solución: Expone el modelo de análisis, el de diseño y el de implementación que responden directamente a la solución del problema.

Capítulo 3. Comprobación del funcionamiento de la propuesta de solución: Incluye las definiciones y los resultados de las pruebas realizadas al sistema, para comprobar la efectividad de la aplicación.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

Introducción

En el presente capítulo se exponen los aspectos teóricos tratados en el desarrollo de la investigación. Se incluyen los principales conceptos relacionados con el tema del presente trabajo, así como el resultado del estudio realizado sobre las soluciones existentes a nivel nacional e internacional para la representación del corpus textual de archivos digitales textuales. Se describen además, la metodología, herramientas, tecnologías y lenguajes seleccionados para el desarrollo de la solución.

1.1 Aspectos teóricos de la investigación

Para tratar la representación del corpus textual de archivos digitales textuales en contribución a la representación de información, es necesario identificar los conceptos fundamentales que permitan la comprensión del presente trabajo. En esta sección se definen los aspectos teóricos de la investigación.

Información

La información es un conjunto de datos con un significado, que reduce la incertidumbre o aumenta el conocimiento de la realidad. Constituye un mensaje con significado en un determinado contexto, disponible para uso inmediato y que proporciona orientación a las acciones por el hecho de reducir el margen de incertidumbre con respecto a la toma de decisiones (2). Por lo tanto la información se extrae de la comprensión de determinados datos, sobre un tema específico que en su contexto refleja un determinado mensaje. En resumen la información no es más que los datos en un contexto determinado que se utiliza como recurso para la toma de decisiones. Por su estructura, tal recurso se divide en diferentes dominios: información estructurada e información no estructurada. Para la actual investigación, en ambos casos se utiliza la información contenida en documentos de texto.

Información textual estructurada

La información textual estructurada es almacenada en un formato riguroso que permita diferenciar las distintas partes en un documento. Por ejemplo: una base de datos, una tabla, libros académicos, artículos científicos, diccionarios o glosarios (20). En estos textos la ubicación de las palabras sirve como base para extraer información. Cuando se conoce previamente la estructura de los datos, la extracción y representación del contenido puede resultar favorecida.

Información textual no estructurada

Este tipo de información no posee una estructura definida, es un texto libre de formato. Ninguna parte del contenido tiene más importancia que otra. De ahí que la extracción de este tipo de

información no es sencilla e implica una etapa de preprocesamiento. El texto no estructurado se compone generalmente de prosa en forma natural, por ejemplo: documentos creados con algún procesador de textos, un correo electrónico, la información que se genera en la web, entre otros (20). La información independientemente de su estructura, cuyo valor está dado por su carácter histórico, es normalmente almacenada en archivos para su posterior consulta y utilización.

Archivo

Un archivo se define como un conjunto de documentos sean cuales sean su fecha, su forma y su soporte material, producidos o recibidos por toda persona física o moral y por todo servicio u organismo público o privado, en el ejercicio de su actividad. Son conservados por sus creadores o por sus sucesores para sus propias necesidades y transmitidos a la institución de archivos competente en razón de su valor archivístico (3). Por lo que se puede resumir que un archivo no es más que un conjunto de documentos que una sociedad, institución o persona produce mediante la realización de sus actividades y funciones. Con el desarrollo de las TIC, se ha agilizado el proceso de digitalización, y con ello muchos archivos en formato duro se han convertido en archivos digitales.

Archivo digital

Un archivo digital es una unidad de datos o información almacenada en algún medio digital que puede ser utilizada por aplicaciones de la computadora (5). Por lo que un archivo digital puede ser de audio, video, texto, entre otros. Para el caso de la actual investigación y teniendo en cuenta el concepto de archivo anteriormente expuesto, se hace referencia solamente a los documentos en forma de archivo digital de texto. El interés de trabajar con archivos digitales viene dado por la necesidad de recuperar y representar información a partir del contenido de los mismos. La rama de la ciencia dedicada al proceso de reconocimiento, preprocesamiento, extracción y estructuración de la información contenida en fuentes documentales se denomina Recuperación de Información (RI).

Recuperación de información

La RI se define como el conjunto de tareas mediante las cuales un sujeto localiza y accede a los recursos de información que son pertinentes para la resolución de un problema determinado. En estas tareas desempeñan un papel fundamental los lenguajes documentales, las técnicas de resumen, la descripción del objeto documental, entre otras técnicas (6). Estas técnicas permiten definir una determinada estructura sobre la información dispersa así como facilitar la búsqueda sobre la misma.

Esta ciencia permite además relacionar la estructura, el análisis, la organización, almacenamiento, búsqueda y recuperación de información (21). Incluye la representación, el almacenamiento, la

organización y el acceso a elementos de información (7). Facilita la localización y presentación a un sujeto de información relevante a una necesidad de información expresada como una pregunta (8). Para lograr la recuperación de información sobre archivos digitales, son utilizadas herramientas que a partir del contenido de tales archivos arrojan determinados resultados que permiten el análisis y representación de los mismos. Estas herramientas son denominadas analizadores léxicos.

Analizador léxico

Un analizador léxico es una parte de un programa de software que se encarga de reconocer los elementos básicos (símbolos o *tokens*) que conforman un lenguaje. Su principal tarea es realizar el análisis lexicográfico a una cadena de entrada (22).

Tokenización o análisis lexicográfico

Consiste en la conformación de las palabras o *tokens* a medida que el analizador léxico recorre el corpus de un documento. Este se encarga de detectar comienzo y fin de una palabra, además de reconocer símbolos no alfabéticos como números o caracteres especiales. También estandariza todas las palabras ya sea a mayúsculas o minúsculas (9). Por tanto el proceso de tokenización disminuye el corpus textual de los documentos y mejora la organización del contenido escrito. La estandarización a minúsculas permite que palabras de igual pronunciación pero escritas diferentes, ya sea por su importancia o ubicación sintáctica dentro de la oración, sean interpretadas como iguales por el derivador o *stemmer*. El resultado final es un corpus textual apto para el proceso de derivación o *stemming*.

Stemming o derivación

El proceso de derivación o *stemming* es un método de reducción el cual permite detectar variantes morfológicas de una misma palabra. Normalmente el *stemming* debe reemplazar las palabras por su raíz (en este punto de la investigación se considera como término la raíz semántica de cada palabra), así como detectar las palabras en plural y convertirlas en singular, logrando la estandarización del documento. Todo esto posibilita tener índices de menor tamaño, que disminuyen el costo computacional del proceso de búsqueda sobre la colección de documentos (9). Durante este proceso pueden encontrarse palabras que carecen de significado para la representación del texto las cuales son eliminadas para reducir el espacio de representación, dichas palabras se reconocen como palabras nulas.

Palabras nulas

Se conoce como palabras nulas o vacías a aquellos vocablos que ocurren muchas veces dentro del cuerpo de un documento y que carecen de algún significado semántico de importancia para el

procesamiento de la información, normalmente estos vocablos son: artículos, preposiciones y conjunciones. Estos son eliminados completamente de los documentos reduciendo considerablemente el tamaño de la representación del corpus textual (9). Algunos estudios determinan que se llegan a reducir los archivos de índices en un 40 por ciento (23).

1.2 Estudio de los antecedentes de la investigación.

Una vez aclarado los conceptos fundamentales involucrados en la investigación, el siguiente paso consiste en entender cómo se realiza la recuperación de la información. La situación relacionada con el alcance de la investigación, consiste en la representación del corpus textual de archivos digitales. La manera de representar los documentos y la forma de medir su similitud son características distintivas de los modelos de recuperación de información.

La definición de modelo utilizada en la investigación es una adaptación directa de su significado general (esquema teórico de un sistema o de una realidad compleja que se elabora para facilitar su comprensión y el estudio de su comportamiento) (24). Por tanto, se considera un modelo como un método para representar documentos en sistemas de recuperación de información y comparar la similitud de esas representaciones.

1.2.1 Modelos de recuperación de información

Los tres modelos que se analizan en esta sección (el modelo booleano, el modelo probabilístico y el modelo vectorial) fueron desarrollados para la recuperación de información a partir de una consulta realizada por un sujeto. Tales modelos reciben como entrada una combinación de palabras que conforman la consulta y a partir de ello utilizan un mecanismo de comparación entre cada documento de la colección y la consulta, devolviendo los documentos más semejantes clasificados por relevancia. La principal diferencia entre ellos reside en la forma de representar y relacionar dichos términos. Una deficiencia general la constituye que necesitan ser transformados para retornar resultados donde la propia consulta sea un documento como tal.

Otra diferencia radica en la manera de comprobar cómo se ajusta el documento recuperado con la consulta realizada. Si el documento debe corresponder exactamente con la consulta se trata de un modelo de ajuste exacto, mientras que si el documento puede corresponder sólo de manera parcial con la consulta se trata de un modelo de mejor ajuste. En los modelos del primer grupo el resultado de una consulta es un conjunto de elementos sin orden alguno que satisfacen la búsqueda. Por el contrario, en los modelos del segundo grupo el resultado es una lista (un ranking) donde se encuentran los documentos ordenados teniendo en cuenta el modo en que se ajustan a la consulta realizada. Entre las ventajas de los modelos del primer grupo están el ser muy eficientes, previsibles,

fáciles de explicar y que funcionan bien cuando el usuario es capaz de expresar exactamente lo que necesita (aunque esta es también su principal desventaja ya que es muy complicado que eso suceda). El modelo booleano es un ejemplo de este tipo. Los modelos de mejor ajuste, como el probabilístico o el vectorial, son más sencillos de emplear y por tanto, significativamente más eficaces (la evaluación de la eficacia se ha ido imponiendo como el aspecto más importante a evaluar en estos modelos) y existen implementaciones optimizadas que proporcionan una eficiencia similar a los de ajuste exacto (24).

Modelo Booleano

Este modelo está basado en el álgebra de Boole. En él cada documento está representado por un conjunto de términos, donde cada uno está asociado a una variable booleana. La aparición del término buscado en el documento se representa como verdadero y la no ocurrencia como falso. Emplea un esquema de pesos binarios asociados a los términos, lo cual no implica si el término es de alta o baja frecuencia de aparición en el documento, solo importa si está o no (9) (24).

Su principal desventaja radica en la simplicidad de la representación de la importancia de los términos, le concede la misma importancia a un término que aparece una sola vez en todo el documento que a otro término que se repite n veces, donde n es un número mayor con respecto al total de términos del documento (9).

Modelo Probabilístico

El modelo probabilístico define la recuperación como un proceso de clasificación, de tal forma que para cada consulta existen dos clases: la clase de los documentos relevantes y la clase de los documentos no relevantes. Por tanto, dado un documento D y una consulta determinada se puede calcular con qué probabilidad pertenece el documento a cada una de esas clases. Si la probabilidad de que pertenezca a la clase de los documentos relevantes es mayor que la probabilidad de que pertenezca a la clase de los no relevantes, el documento será relevante para la consulta. Del mismo modo, se puede elaborar un ranking de relevancia. Las diferentes maneras en las que se pueden estimar estas probabilidades dan lugar a los diferentes modelos probabilísticos. Este modelo asume que la relevancia de un documento es independiente del resto de documentos de la colección. Si se elabora un ranking en orden decreciente de probabilidad de que los documentos sean relevantes, se obtiene la mayor eficacia posible (es decir, se minimiza la probabilidad de error) (24).

Este modelo necesita un método para calcular las probabilidades iniciales. Aunque existen numerosas alternativas para realizar este cálculo, la más habitual es emplear las frecuencias de los términos en cada documento y la frecuencia en el total de los documentos. Además, el modelo es

iterativo, tras una primera aproximación donde se obtiene un subconjunto inicial se emplean los documentos en dicho subconjunto para refinar la búsqueda, recalculando las probabilidades. Este proceso se repite hasta obtener las probabilidades definitivas (24).

Los resultados de aplicar este modelo no son mucho mejores que los obtenidos en el modelo booleano, además de que todos los documentos seleccionados no son realmente relevantes. Entonces, se debe considerar la posibilidad de que un documento sea relevante o no, dado que haya sido ya seleccionado. Este modelo es complejo y de alto costo computacional, además ofrece pobres resultados debido a que solo organiza los documentos en dos clases, relevante y no relevante. Necesita de un proceso de entrenamiento por lo que inicialmente no es posible conocer el conjunto de documentos relevantes y se basa en un proceso de clasificación inicial para conocer las dos clases de documentos según la consulta utilizada.

Modelo Vectorial o Modelo de Espacio Vectorial

En el modelo vectorial cada documento de la colección se representa por un vector t -dimensional de términos indexados donde t es la cardinalidad del conjunto de términos. A cada elemento del vector le corresponde el peso del término asociado a esa dimensión (9) (24) (25). Esto significa, que en el modelo espacio vectorial cada documento de la colección tiene un vector asociado, cada componente del vector coincide con cada término del documento por lo que su dimensión es igual a la cantidad de términos.

La idea fundamental en la que se basa el modelo vectorial es considerar que tanto la importancia de un término con respecto a un documento como los documentos y las consultas se pueden representar como un vector en un espacio de alta dimensionalidad. Por tanto, para evaluar la similitud entre un documento y una consulta; es decir, para obtener la relevancia del documento con respecto a la consulta, hay que realizar una comparación de los vectores que los representan (9) (24).

La principal desventaja de este modelo es el uso de recursos computacionales. Un documento extenso trae consigo un vector de muy alta dimensionalidad, lo que eleva el almacenamiento en memoria y retarda las operaciones de procesamiento. Sin embargo es un modelo sensible a la medida de semejanza escogida. El valor de esta técnica consiste en que toda la importancia o peso asociado a cada término es determinada por una función independiente al modelo, por lo que una buena función de semejanza logra mejores resultados que los aplicados en los modelos booleano y probabilístico. Mediante esta representación se pueden establecer medidas de similitud entre los documentos de la colección e inclusive llevar un ranking para la agrupación y clasificación, pudiendo

realizar búsquedas aproximadas. Por otro lado, a partir de las bibliografías consultadas su uso es el más difundido y se considera como una buena estrategia para representar el corpus textual de documentos digitales textuales (26).

Conclusiones del estudio de los modelos de recuperación de información

Luego del análisis teórico de los principales modelos de recuperación de información, teniendo en cuenta sus ventajas y deficiencias, se decide para la presente investigación el uso del Modelo Vectorial. El criterio de selección se sustenta en la representación y clasificación que brinda para los archivos digitales textuales.

El análisis del modelo booleano arroja una pobre representación del contenido de los archivos digitales textuales, cada término dentro de la colección posee la misma importancia lo que no permite establecer medidas de similitud entre los documentos. El modelo probabilístico solamente permite la clasificación independiente de un documento, determina la importancia de un documento independientemente de la colección. Sin embargo el modelo vectorial permite representar un archivo digital textual sin afectar el contenido semántico del mismo, así como establecer criterios comparativos sobre cada término de la colección. La importancia de cada archivo digital se determina dentro de la colección y no fuera de ella. No compromete la representación de la información siendo totalmente independiente con respecto a la medida de semejanza seleccionada. Su uso es el más difundido en el área de la recuperación de información, se considera una buena alternativa y una fuerte estrategia para representar colecciones de documentos.

1.2.2 Soluciones existentes para la representación del corpus textual.

La evaluación de los sistemas desarrollados dentro del campo ha sido uno de los temas a los que más atención se le ha prestado desde los orígenes de la recuperación de información. Para cerrar el estudio de los antecedentes de la investigación se enuncian y analizan algunos sistemas que implementen modelos de recuperación de información bajo la tarea de representación del corpus textual de documentos. Esta actividad, destinada fundamentalmente a determinar si estos sistemas funcionan correctamente así como si un cambio determinado mejora su funcionamiento y resultados, se describe en la presente sección.

WordStat

WordStat es un software privativo de análisis cualitativo de datos desarrollado por la compañía Provalis Research con sede en Montreal, Canadá. Este analiza el texto y relaciona su contenido a información estructurada, que incluya datos numéricos y categoriales. Sus principales funciones son el análisis de contenido en colección de documentos ANSI o RTF y variables alfanuméricas cortas

(hasta de 255 caracteres). También permite la exclusión opcional de pronombres y conjunciones, mediante el uso de listas de exclusión definidas por el usuario, así como el análisis de frecuencia de palabras clave, frases, categorías, conceptos derivados o códigos definidos por el usuario e ingresados manualmente dentro de un texto. Actualmente su distribución solo funciona en sistemas operativos Windows, para su uso en Mac y Linux es necesario un emulador (13). Esta es una de sus principales desventajas, la necesidad de un emulador para que funcione en el sistema operativo Linux impide su integración con otro software de esta plataforma. Además de ser un sistema privativo lo cual indica que está sujeto a una licencia con costo para su adquisición y evita que pueda ser modificado de ser necesario para dar solución al problema que se presenta.

Alceste

Alceste es un software de análisis de datos textuales desarrollado bajo plataforma Windows por la compañía Image en colaboración con el Consejo Nacional de Investigaciones Científicas de nacionalidad francesa. Dicho software procede a un primer análisis del vocabulario de un corpus textual y hace el diccionario de estas palabras con sus raíces y frecuencias. Luego se corta el texto en segmentos homogéneos que contienen un número suficiente de palabras y procede a una clasificación de estos segmentos mediante la detección de oposiciones más fuertes. Este método permite la extracción de clases por significado, formado por la mayoría de las palabras y frases específicas, las clases restantes representan las ideas principales y temas del corpus. Muestra los resultados globales, ordenados en función de su relevancia, con varias representaciones gráficas e informes de análisis (14) (15). Al ser un software privativo contiene restricciones de uso marcados por la licencia con que cuenta, así como la imposibilidad de modificación e integración con otro software. Además de estar disponible en este caso para las plataformas Windows y Mac solamente, imposibilitando su uso en sistemas operativos Linux.

T-LAB

T-LAB es un software desarrollado por el psicólogo italiano Franco Lancia. Está compuesto por un conjunto de herramientas lingüísticas y estadísticas para el análisis de contenido, el análisis del discurso y la minería de textos, desarrollada para plataforma Windows. Durante el proceso de importación, T-LAB realiza los tratamientos siguientes: normalización del corpus textual, detección de multi-palabras y palabras vacías, segmentación en contextos elementales, [lematización](#) automática y selección de palabras clave (16).

A partir del estudio realizado acerca de este software no se pudo determinar el modelo de recuperación y algoritmo que emplea el mismo debido a que es un software privativo y este tipo de información no es brindada en los sitios consultados. Esta condición de software privativo impide

que pueda modificarse o integrarse a otra aplicación y al estar disponible solamente en sistemas operativos Windows impide su uso en Linux. De igual forma es necesario señalar que su objetivo es el análisis lingüístico de textos, no permite comparativa de documentos u obtención del corpus representado. Además como se observa en su descripción anteriormente expuesta, uno de sus principales objetivos es la selección de palabras clave.

IRAMUTEQ

IRAMUTEQ es un software gratuito desarrollado bajo la lógica de código abierto, con licencia de GNU GPL (v2). Es un software de origen francés que permite diferentes tipos de análisis de datos textuales, así como la lexicografía básica (cálculo de frecuencia de las palabras) para el análisis multivariado (clasificación jerárquica descendente, análisis de similitud). Identifica el número de palabras, la frecuencia media, el número de palabras ([legomena](#)) y reduce el vocabulario de palabras basadas en sus raíces (17).

El análisis de similitud basado en la teoría de grafos le posibilita identificar las co-ocurrencias entre las palabras y el resultado trae indicaciones de conexión entre las palabras, ayudando en la identificación de la estructura de un corpus textual. La nube de palabras se agrupa y se organiza gráficamente de acuerdo a su frecuencia. Se trata de un análisis léxico simple, sin embargo gráficamente bastante interesante, ya que permite la rápida identificación de las palabras clave de un corpus. Este análisis puede ser realizado tanto a partir de un grupo de textos sobre un tema determinado reunido en un solo archivo, como a partir de un conjunto de tablas de personas por palabras organizadas en hojas de cálculo. Los textos o tablas se deben generar preferentemente por OpenOffice o LibreOffice para evitar errores relacionados con la codificación (17).

La incapacidad por parte de este software de realizar los diferentes tipos de análisis de datos textuales a una colección de archivos de textos le resta factibilidad como solución para la presente investigación. Además de que al tratar con un único archivo de texto, los textos contenidos en este deben ser de un mismo tema y la identificación de palabras claves se realiza en base a un único documento y no a una colección de documentos.

Lucene

Lucene surge como respuesta al creciente interés por los sistemas de clasificación automática de datos a raíz del desarrollo masivo de las redes de computadoras y medios de almacenamiento. Es un software de recuperación de información de código abierto y multiplataforma, apoyado por la Fundación Apache radicada en Estados Unidos. Es una interfaz de programación de aplicaciones ([API](#) por sus siglas en inglés) flexible que permite añadir capacidades de indexación, búsqueda y

representación del corpus textual a cualquier sistema que se esté desarrollando. Lucene permite dividir el contenido de los documentos en campos y así poder realizar consultas con un mayor contenido semántico, en vez de tratar los documentos como simples flujos de palabras. Se pueden buscar términos en los distintos campos del documento concediéndole más importancia según el campo en el que aparezca. Por ejemplo, si se dividen los documentos en dos campos, título y contenido, puede concederse mayor importancia a aquellos documentos que contengan los términos de la búsqueda en el campo título (18). Es necesario resaltar que este software necesita para la recuperación de la información que el contenido esté previamente indexado. Además se basa en la comparación de consultas contra documentos para establecer la semejanza entre ellos en base a la consulta recibida pero no realiza una comparación entre documentos.

Conclusiones del estudio de las soluciones informáticas existentes para la recuperación de información

A pesar de que las soluciones informáticas analizadas anteriormente abarcan un gran número de funcionalidades que les permiten a los usuarios utilizarlas como herramientas para la recuperación de información y representación del corpus textual de documentos, presentan deficiencias que son necesarias resolver. WordStat, Alceste y T-LAB muestran un conjunto de características que se pueden tener en cuenta para el desarrollo de la solución, entre las cuales se encuentra la exclusión de pronombres y conjunciones del contenido de los documentos digitales a través del uso de listas de exclusión. Además conforman diccionarios con las raíces de los términos y sus frecuencias de aparición. Sin embargo son soluciones informáticas de plataforma Windows, esto imposibilita su integración con varios tipos de software y limita su capacidad multiplataforma, además de ser software propietario lo cual no se corresponde con las proyecciones de la Universidad de las Ciencias Informáticas y Cuba con respecto al uso del software libre y la soberanía tecnológica. Dentro de las limitaciones individuales se encuentra que: WordStat basa su análisis sobre información estructurada, por lo que queda fuera del campo de la investigación al no poder realizar ningún tipo de análisis sobre información no estructurada. Alceste realiza la extracción de textos y los representa organizados en clase por significados por lo que se supone que necesita un proceso de entrenamiento que determine las clases en las que se dividen los documentos. Además solo permite analizar las ideas, frases y palabras principales del texto en cuestión.

De igual forma es necesario señalar que el objetivo de T-LAB es el análisis lingüístico de textos y no permite comparativa de documentos u obtención del corpus representado. Su objetivo principal, luego del análisis del corpus y su representación está dado por la localización y definición de las palabras clave. IRAMUTEQ permite el cálculo de las frecuencias de aparición de las palabras dentro

de los documentos digitales textuales y la reducción del número de palabras basada en sus raíces. A pesar de esto, los diferentes análisis de datos textuales son realizados sobre un único archivo de texto y todos los textos deben pertenecer a un mismo tema, por lo que no permite representar diversos contenidos simultáneamente y no realiza o permite el análisis de información no estructurada.

Lucene presume ser la solución informática más cercana para resolver el objetivo de la investigación. Cuenta con un conjunto de funcionalidades para tratar el corpus textual de los documentos digitales. Alberga todas las características encontradas en las otras cuatro soluciones informáticas explicadas, es multiplataforma y permite la integración de sus componentes con otros sistemas. Sin embargo su principal deficiencia es que al crecer el volumen de datos a procesar el tiempo de respuesta es elevado, lo que disminuye su utilización por potenciales deficiencias de rendimiento. A pesar de que se puede integrar con diferentes sistemas su incapacidad para la comparación entre documentos resulta ser un problema para su selección, debido a que su procesamiento se basa en la comparación de una consulta con una colección de textos y no entre colecciones de textos. Por otro lado necesita una indexación previa para realizar búsquedas sobre el corpus textual de los documentos digitales.

Por tanto es necesario implementar una solución capaz de integrarse a sistemas informáticos de múltiples plataformas, que permita el procesamiento de datos textuales así como su representación en una determinada estructura. Además es necesario analizar la mayor cantidad de archivos de texto sin importar el tema que traten en el menor tiempo posible, lo que se deriva en un análisis independiente de la estructura de la información. La representación y extracción de términos debe ir más allá que a la indexación o identificación de frases o palabras clave. Todo esto con el objetivo final de lograr una representación textual del corpus de archivos digitales de texto.

1.3 Metodología, Lenguajes y Herramientas de Desarrollo

En aras de preparar el ambiente de desarrollo del software, se realiza la identificación y el análisis de la metodología de desarrollo, lenguaje de programación, entorno de desarrollo, lenguaje y herramienta de modelado. En esta sección se introduce el análisis de las principales características y ventajas de cada uno de los elementos antes mencionados que justifican su elección como vía para dar solución a la problemática planteada en cumplimiento de los objetivos definidos.

1.3.1 Metodología de Desarrollo

Existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo de software. Por una parte se tienen aquellas más tradicionales que se centran

especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir así como las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos, ya sea debido a la numerosa documentación que generan, al tamaño de los equipos que le utilizan así como otras múltiples causas.

Por otra parte están las metodologías ágiles, las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas y generando poca documentación. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo calidad. Las metodologías ágiles están revolucionando la manera de producir software, pues de forma efectiva logran que bajo su uso la mayoría de los proyectos lleguen a feliz término con buenos resultados (27). Para guiar el desarrollo del software a implementar se decide el empleo de una metodología ágil. Dicha selección se basa en que el equipo de desarrollo involucrado es pequeño, existe una estrecha relación con el cliente, llegando este a formar parte del equipo y se necesitan constantes entregas del avance en el proceso de desarrollo. También se tiene en cuenta que durante este proceso el software se puede ver sometido a constantes cambios. A continuación se describen cuatro metodologías de desarrollo ágil y se selecciona una de ellas.

Feature Driven Development

Feature Driven Development (FDD por sus siglas en inglés) es una metodología ágil diseñada para el desarrollo de software, basada en la calidad y el monitoreo constante del proyecto. Esta se enfoca en iteraciones cortas, que permiten entregas tangibles del producto en un período corto de tiempo, de como máximo dos semanas. Se preocupa por la calidad, por lo que incluye un monitoreo constante del proyecto y propone tener etapas de cierres cada dos semanas por lo que se obtienen resultados periódicos y tangibles. Además se basa en un proceso iterativo con iteraciones cortas que producen un software funcional que el cliente puede ver y monitorear. Define claramente entregas tangibles y formas de evaluación del progreso del proyecto. No hace énfasis en la obtención de los requerimientos sino en cómo se realizan las fases de diseño y construcción (28).

La metodología FDD define cinco procesos desarrollados de forma iterativa: desarrollar el modelo global, construir una lista de características, planificar, diseñar y construir por características. El modelo global se construye teniendo en cuenta la visión, el contexto y los requisitos que debe tener el sistema a construir. Luego se elabora una lista que resume las funcionalidades que debe tener el sistema, la cual es evaluada por el cliente. Como tercera actividad se procede a ordenar los conjuntos de funcionalidades conforme a su prioridad y dependencia y se asigna a los

programadores jefes. Posteriormente en la etapa de diseño se selecciona un conjunto de funcionalidades de la lista y se procede a diseñar y construir la funcionalidad mediante un proceso iterativo, decidiendo que funcionalidad se van a realizar en cada iteración. Este proceso iterativo incluye inspección de diseño, codificación, pruebas unitarias, integración e inspección de código. Luego se pasa a la última actividad que es la construcción del proyecto (28).

Es necesario resaltar que si bien el diseño del modelo global al inicio del desarrollo elimina confusiones y ayuda a comprender mejor las características que deben ser implementadas, se necesita tener un conocimiento amplio sobre los procesos del negocio y contar con un profesional de alta experiencia para el diseño del mismo. Además el código no puede tomarse como una documentación, esto significa que es necesario generar documentación adicional para explicar el mismo.

SCRUM

Es una metodología especialmente desarrollada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir de la siguiente forma: el desarrollo de software se realiza mediante iteraciones denominadas sprints, con una duración de 30 días. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo proyecto, especialmente la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración. Al finalizar un sprint, las tareas que se han realizado y el cliente ha quedado conforme con ellas, no se vuelven a modificar. El equipo de desarrollo trata de seguir el orden de prioridad que marca el cliente pero puede modificarlo si determina que es mejor hacerlo y cada miembro del equipo trabaja de forma individual (27). Esta metodología se basa más en la administración del proyecto que en la propia programación o creación del producto.

Crystal Methodologies

Se trata de un conjunto de metodologías para el desarrollo de software caracterizadas por estar centradas en las personas que componen el equipo y la reducción al máximo del número de artefactos producidos. Han sido desarrolladas por Alistair Cockburn. El desarrollo de software se considera un juego cooperativo de invención y comunicación, limitado por los recursos a utilizar. El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipo definidas. Estas políticas dependerán del tamaño del equipo, estableciéndose una clasificación por colores, por ejemplo Crystal Clear (3 a 8 miembros) y Crystal Orange (25 a 50 miembros) (27).

Programación Extrema

La Programación Extrema (XP por sus siglas en inglés) es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. Se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. Se define como adecuada para proyectos con requisitos imprecisos y muy cambiantes y donde existe un alto riesgo técnico (27). Esta metodología propone un desarrollo iterativo e incremental que se basa en realizar pequeñas mejoras, unas tras otras. Las tareas de desarrollo se deben llevar a cabo por dos personas en un mismo puesto, asume que la calidad del código escrito de esta manera es más importante que la posible pérdida de productividad inmediata. Recomienda que un representante del cliente trabaje junto al equipo de desarrollo, así como la corrección de todos los errores antes de añadir una nueva funcionalidad, haciendo entregas frecuentes.

Para la investigación se decide emplear la metodología de desarrollo XP. Para ello se tienen en cuenta tres factores fundamentales: equipo de desarrollo, proyecto a desarrollar y cliente.

Equipo de desarrollo

El equipo de desarrollo está formado por dos programadores que trabajan en pareja (delante del mismo ordenador). Las ventajas de este estilo de programación son: muchos errores son detectados conforme son introducidos en el código (inspecciones de código continuas), por consiguiente la tasa de errores del producto final es más baja, los diseños son mejores y el tamaño del código menor (continua discusión de ideas de los programadores). Los problemas de programación se resuelven más rápido, se posibilita la transferencia de conocimientos de programación entre los miembros del equipo, varias personas entienden las diferentes partes del sistema, los programadores conversan mejorando así el flujo de información y la dinámica del equipo.

Proyecto a desarrollar

El proyecto a desarrollar se debe entregar en menos de un año, lo que se considera un período de tiempo corto. Se necesitan ciclos cortos tras los cuales se muestren resultados del proceso de desarrollo. Esto permite minimizar el tener que rehacer partes que no cumplen con los requisitos y ayuda a los programadores a centrarse en lo que es más importante, en este caso la entrega del producto de software.

Cliente

El cliente forma parte del desarrollo del software, su opinión sobre el estado del proyecto se conoce en tiempo real y la comunicación entre este y el equipo de desarrollo es fluida. Esta integración posibilita que el cliente conozca en todo momento el estado de desarrollo del software. Además de que le permite al equipo de desarrollo conocer inmediatamente las inconformidades o cambios necesarios a medida que el cliente prueba el software.

Lenguaje de Modelado

El lenguaje unificado de modelado (UML por sus siglas en inglés) es un lenguaje para especificar, construir, visualizar y documentar los artefactos de un sistema de software (29). Este lenguaje permite a los analistas y desarrolladores del software representar todos los elementos necesarios para el proceso de desarrollo de software. En general UML representa en un lenguaje común y formal el desarrollo de cualquier producto de software. A pesar de que la metodología XP no define el uso UML, este se escoge para el modelado de los artefactos diagrama de dominio y diagrama de clases, que permiten establecer las relaciones entre los objetos para la comprensión de la solución.

1.3.2 Herramientas CASE

La Ingeniería de Sistemas Asistida por Computadoras (CASE por sus siglas en inglés) es la aplicación de tecnología informática a las actividades, las técnicas y las metodologías propias de desarrollo, su objetivo es acelerar el proceso para el que han sido diseñadas, en el caso de CASE para automatizar o apoyar una o más fases del ciclo de vida del desarrollo de sistemas (30).

Visual Paradigm 8.0

Visual Paradigm es una herramienta para UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y documentación. Presenta licencia gratuita y comercial (31). Además de brindar soporte UML, permite utilizar técnicas de ingeniería inversa para llevar de código a diagramas de clases, manteniendo de esta manera la sincronización entre el modelo y el código. Posibilita la generación de código a partir de un modelo o diagrama y cuenta con un generador de informes. También realiza la distribución automática de diagramas así como la reorganización de las figuras y conectores de los diagramas UML.

1.3.3 Lenguaje de programación y Entorno de desarrollo

Java es un lenguaje desarrollado por Sun Microsystems, está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de trabajo, sobre arquitecturas distintas y con

sistemas operativos diversos. Es un lenguaje simple, orientado a objetos, distribuido, robusto, seguro, de arquitectura neutra, portátil, de alto desempeño, dinámico y que tiene soporte para hilos múltiples, es libre y de código abierto (32) (33). El tiempo estimado para desarrollar la solución es corto, por lo que no es aconsejable el aprendizaje de otro lenguaje, ya que el equipo tiene experiencia utilizando Java. Para el desarrollo de la solución se utilizará Java en su versión 8.0.

Entorno de Desarrollo Integrado

Un Entorno de Desarrollo Integrado (IDE por sus siglas en inglés) es un entorno de programación que ha sido empaquetado como un programa de aplicación. Consiste en un editor de código de programación, un compilador, un depurador y un constructor de interfaz gráfica (34).

Eclipse

Es una plataforma de desarrollo, que contiene un IDE diseñado para ser extendido a través de plugins. Este IDE no está pensado para un lenguaje en específico, sino que es genérico. Brinda la funcionalidad de programar en el lenguaje Java usando el plugin Herramientas de desarrollo en Java (JDT por sus siglas en inglés) que viene incluido en la distribución estándar del IDE. Dispone además de un editor de texto con un analizador sintáctico y realiza la compilación en tiempo real. Cuenta con pruebas unitarias usando JUnit, así como asistentes para creación de proyectos y permite la refactorización de código (35).

NetBeans 8.0

Es un entorno de desarrollo integrado hecho principalmente para el lenguaje de programación Java, desarrollado por la compañía Sun Microsystems en el año 2000. Desde su lanzamiento NetBeans IDE es un producto libre y gratuito sin restricciones de uso, de código abierto, escrito completamente en Java usando la plataforma NetBeans. Soporta el desarrollo de todos los tipos de aplicación Java, siendo el IDE oficial de Java 8. Cuenta con un constantemente mejorado Editor de Java, muchas características enriquecidas y una amplia gama de herramientas, plantillas y muestras. Su editor permite refactorizar código fácilmente, también proporciona plantillas de código, consejos de codificación y generadores de código (36).

Para la codificación de la solución se selecciona NetBeans 8.0, el motivo de la elección es el conocimiento que tiene el equipo de desarrollo sobre esta herramienta, en múltiples ocasiones ha sido utilizada como entorno de desarrollo.

Conclusiones del Capítulo

En el capítulo se elaboró el marco teórico de la investigación a partir del análisis de los antecedentes de las soluciones existentes orientadas a la representación del corpus textual de los archivos

digitales textuales. Para el cumplimiento del objetivo se identificaron y definieron los conceptos fundamentales en la representación del corpus textual de archivos que permitieron la descripción del objeto de estudio de la investigación. También se identificaron y describieron los diferentes modelos y técnicas de recuperación de información para la definición de los antecedentes de este trabajo.

Se seleccionó el modelo de recuperación de información de espacio vectorial para la solución del problema teniendo en cuenta que permite la representación de los documentos digitales textuales sin afectar su contenido semántico, así como establecer criterios comparativos sobre la colección. Tal modelo es totalmente independiente a la medida de semejanza seleccionada y su uso es el más difundido en el área de la recuperación de información. Por tanto, se considera como alternativa adecuada para la solución del problema planteado.

Con el objetivo de identificar las principales soluciones informáticas existentes en el mundo que utilizan los modelos de recuperación de información, se realizó un estudio de antecedentes en esta área a partir del análisis de las soluciones WordStat, Alceste, T-LAB, IRAMUTEQ y Lucene. Los resultados arrojados durante el mismo determinaron que: tales soluciones (excepto Lucene e Iramuteq) son de plataforma Windows y esto imposibilita su integración con varios tipos de software limitando su capacidad multiplataforma. WordStat basa su análisis sobre información estructurada, Alceste necesita un proceso de entrenamiento que determine las clases en las que se dividen los documentos y T-LAB no permite comparativa de documentos u obtención del corpus representado. IRAMUTEQ no permite representar diversos contenidos simultáneamente y no realiza o permite el análisis de información no estructurada. Lucene se basa en la comparación de una consulta con una colección de textos y no entre colecciones de textos y necesita una indexación previa para realizar búsquedas sobre el corpus textual de los documentos digitales. Por tanto se concluyó que es necesario implementar una solución para realizar dicha representación capaz de integrarse a sistemas informáticos de múltiples plataformas, que permita el procesamiento de archivos digitales textuales y que analice la mayor cantidad sin importar el tema que traten.

Para preparar el ambiente de desarrollo de la investigación se realizó el estudio de las metodologías ágiles FDD, SCRUM, Crystal Methodologies y XP. A partir de este estudio se seleccionó la metodología XP para guiar el desarrollo de software teniendo en cuenta las condiciones y características del equipo de desarrollo, el cliente y la solución a implementar. En apoyo a la metodología se seleccionó el lenguaje UML, así como la herramienta CASE Visual Paradigm 8.0 que permiten el modelado de artefactos como diagrama de clases y modelo de dominio para la

comprensión de la propuesta de solución. También se caracterizó el lenguaje de programación Java y entorno de desarrollo NetBeans 8.0 para la codificación e implementación de la solución.

CAPÍTULO 2. ELEMENTOS FUNDAMENTALES DE LA PROPUESTA DE SOLUCIÓN

Introducción

En el presente capítulo se describen los principales artefactos relacionados con el análisis, diseño e implementación de la propuesta de solución al problema identificado y señalado en la actual investigación. El capítulo se divide en tres secciones: Planeación, Diseño e Implementación de la Propuesta de Solución.

En la primera sección se describen los artefactos generados de acuerdo a la metodología de desarrollo utilizada, tales como: historias de usuario, ejecución del proyecto, plan de iteraciones y el plan de entrega al cliente. En la segunda sección se especifica la arquitectura de software, los patrones de diseño a utilizar, el modelado UML y las tarjetas clase-responsabilidad-creador. Finalmente se expone el desarrollo del algoritmo implementado así como las estructuras de datos empleadas para la solución del problema a resolver.

Modelo de la propuesta de solución

A partir del estudio de los antecedentes y consecuentemente con el objetivo de la investigación, se concluye que es necesario diseñar e implementar un componente informático que permita la representación del corpus textual de archivos digitales. La solución debe implementar el modelo de espacio vectorial y proporcionar la arquitectura necesaria para la utilización de las estructuras de datos que permitan representar y organizar una colección de documentos. Tal arquitectura debe permitir explotar las estructuras seleccionadas para aplicar algoritmos que, utilizando la representación del corpus obtenido permitan realizar actividades posteriores que faciliten el indexado, búsqueda, representación y organización de los archivos en los sistemas informáticos pertinentes.

La figura 2.1 presenta un esquema general que aclara el proceso de representación del corpus textual de acuerdo a las especificaciones que se realizan en la actual investigación. La misma se compone por elementos generales que definen la propuesta a desarrollar y sus posibles componentes.

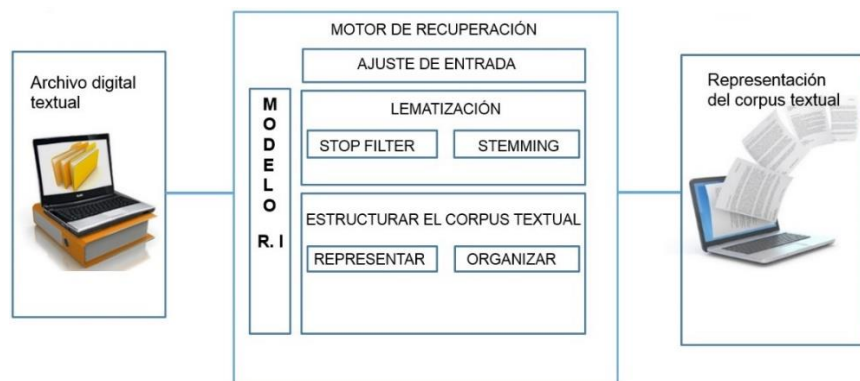


Figura 2.1. Modelo de la propuesta de solución

Como se observa en la figura 2.1 el modelo de propuesta de solución cuenta con tres partes fundamentales: entrada, motor de recuperación y salida. El flujo de entrada consiste en una colección de archivos digitales a procesar por el motor de recuperación. Este primeramente realiza un ajuste de la entrada para luego aplicar el modelo de recuperación. Dicho modelo está dividido en dos etapas fundamentales: la primera etapa cuenta con el lematizador, que se encarga de filtrar y reducir cada palabra a su raíz semántica. Posteriormente se conforman las estructuras de datos que permiten representar y organizar el corpus textual de los archivos digitales. Finalmente se obtiene una representación textual del corpus de tales archivos.

Como etapa posterior al diseño general del modelo de la propuesta de solución, se pretende mostrar un modelo conceptual de carácter técnico que ilustre los conceptos que dan origen a la solución propuesta y sus relaciones. El modelo seleccionado es el de dominio, artefacto UML que permite la representación general de los conceptos a desarrollar y sus relaciones específicas. Tal modelo, aunque no es contemplado como necesario por la metodología de desarrollo empleada, permite representar de forma más adecuada la situación inicial ante el potencial desarrollo de software y a partir de ella definir las posibles funcionalidades como historias de usuario.

Modelo de dominio

Es una representación visual de clases conceptuales o de objetos reales en un dominio de interés. No constituyen clases de software, son representaciones de los conceptos involucrados para la comprensión de la solución y las asociaciones establecidas describen las relaciones entre los conceptos (37). Por tanto un modelo de dominio constituye una representación conceptual de las relaciones entre los objetos involucrados en el proceso de construcción de software. La figura 2.2 muestra una representación estática del entorno real del proyecto mediante el modelo de dominio.

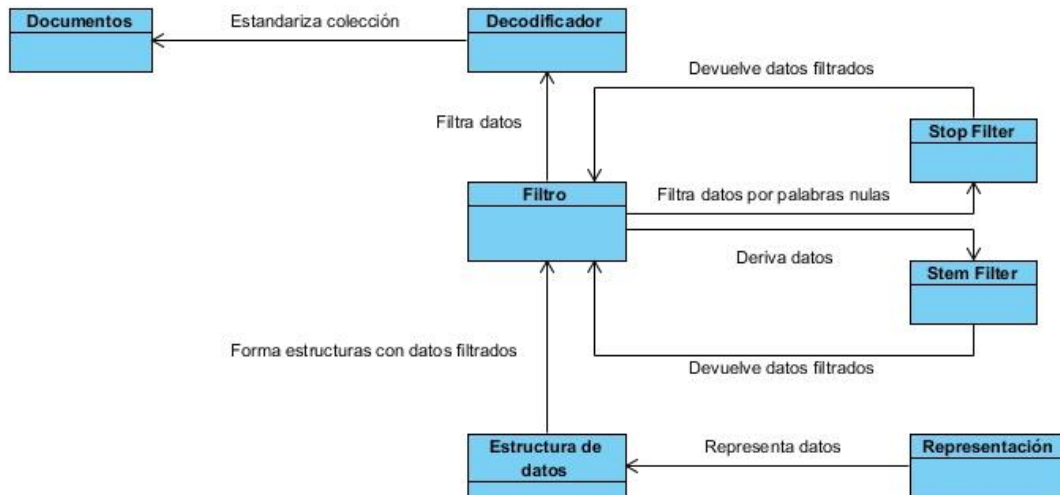


Figura 2.2. Modelo de dominio

En el modelo de dominio mostrado se aprecia que los documentos a procesar necesitan ser estandarizados, tarea que se realiza por medio de un decodificador. Esto se debe a que existen múltiples extensiones para la escritura de textos digitales como: doc, odt, pdf, docx, txt, rtf, entre otras. Independiente de la extensión, la codificación de los caracteres que conforman los textos es otro parámetro a tener en cuenta, la misma puede ser: ISO-8859-1, UTF-8, UTF-16, ANSI, entre otras. Por tal motivo estandarizar la colección de entrada a un formato y codificación predeterminada garantiza el funcionamiento y acoplamiento del proceso de lectura de los documentos.

Luego del proceso de estandarización de los documentos es necesario depurar el contenido de los textos mediante un filtro de datos. Para realizar este proceso se llevan a cabo dos tipos de filtrado, el Stop Filter que consiste en eliminar de los documentos las apariciones de palabras que están incluidas en una lista denominada stopwords (palabras nulas) y luego el Stem Filter que realiza la reducción de cada palabra a su raíz semántica. Aplicar dichos filtros reduce la dimensión de los corpus textuales y posibilita tener índices de menor tamaño. Esto disminuye el costo computacional del proceso de búsqueda sobre la colección de documentos así como la posterior representación del corpus textual. Una vez filtrados los datos se prosigue a conformar las estructuras necesarias para almacenarlos y por último se lleva a cabo el proceso de representación del corpus de la colección de documentos.

2.1 Planeación

La planeación es la etapa inicial en todo desarrollo de software bajo la metodología XP. En este punto comienza la interacción con el cliente para identificar los requerimientos del sistema y se identifican las iteraciones y ajustes necesarios a la metodología según las características de la

solución (27). En esta sección se describen dos elementos fundamentales, los cuales son: Historia de Usuario y Ejecución del Proyecto.

Dichos elementos resumen la etapa inicial planteada por la metodología de desarrollo de software seleccionada (38) (39) (40) (41) (42). En ellos se recogen las características principales del sistema a construir y se concreta el tiempo que se necesitará para implementarlas. Las especificidades de la planeación y definición del tiempo de ejecución del desarrollo en cuestión son mostradas a continuación.

2.1.1 Historias de Usuario

Las historias de usuario son un instrumento propuesto por la metodología seleccionada para el levantamiento de requerimientos en el desarrollo de software. Constituyen tarjetas dónde el cliente describe brevemente (con el fin de que sean dinámicas y flexibles) las características que el sistema debe poseer, sean requisitos funcionales o no funcionales (43). Cada historia de usuario debe ser lo suficientemente comprensible y no muy extensa en cuanto a la cantidad de requisitos a incluir en ella, con el fin de que se puedan implementar en poco tiempo. No existe un estándar determinado para redactar una historia de usuario, según la metodología seleccionada, el equipo de desarrollo determina que es lo importante que debe quedar escrito y como llevar el control de cada una. A partir de un estudio realizado de los elementos más importantes recomendados por varios autores (44) (45) (46) se concluye en la selección de los siguientes campos para conformar una historia de usuario:

- ✓ Número de historia de usuario: permite que la historia sea rápidamente identificada en los pasos posteriores que se llevarán a cabo en la etapa de planificación. La idea es que posean un número consecutivo, que solo proporciona información respecto al orden en el que fueron redactadas las historias.
- ✓ Usuario: persona involucrada en el desarrollo de la historia de usuario.
- ✓ Fecha: corresponde con la fecha en la cual la historia de usuario es redactada.
- ✓ Título: corresponde con el nombre que se le otorga a la historia de usuario por parte del cliente.
- ✓ Descripción: lugar donde el cliente describe que se debe hacer de forma comprensible y no muy extensa, evitando el uso de terminología y la acumulación de varias funcionalidades en una misma historia de usuario.
- ✓ Observaciones: corresponden a aquellos detalles relevantes que serán resueltos tras la conversación del equipo desarrollador con el cliente.
- ✓ Puntos Estimados: tiempo que se asigna o se supone al desarrollo de la historia de usuario.

- ✓ Puntos Reales: tiempo real que demora el desarrollo de la historia de usuario.

De acuerdo con los principios ágiles del desarrollo de software y las técnicas propuestas por la metodología XP, se describen en esta sección las principales historias de usuario del sistema, el resto se encuentra en el anexo número 1. Tales descripciones tienen el objetivo de ilustrar, los elementos relevantes de las funcionalidades más importantes a desarrollar para dar solución al problema planteado.

Las tablas 2.1, 2.2 y 2.3 muestran las historias de usuario Estandarizar colección de documentos, Construir vector de términos y Construir matriz de semejanza respectivamente. Los campos que contienen son los definidos anteriormente.

Tabla 2.1. HU-1. Estandarizar colección de documentos

Historia de Usuario					
Numero de Historia:	HU-2	Usuario:	Programadores	Fecha:	13/04/2015
Título:	Estandarizar colección de documentos				
Descripción:	El componente debe estandarizar la colección de documentos a un formato y codificación específica. Inicialmente recibe una colección de documentos en distintos formatos y codificaciones. Mediante un proceso de conversión de los textos produce como salida una colección de textos en formato txt con codificación UTF-8.				
Observaciones:	El proceso de estandarización debe aceptar varios tipos de formatos y codificación: ISO-2022-CN, BIG5, EUC-TW, GB18030, HZ-GB-23121, ISO-8859-5, KOI8-R, WINDOWS-1251, IBM866, IBM855, ISO-8859-7, WINDOWS-1253, ISO-8859-8, WINDOWS-1255, ISO-2022-JP, SHIFT_JIS, EUC-JP, ISO-2022-KR, EUC-KR, Unicode, UTF-8, UTF-16BE / UTF-16LE, UTF-32BE / UTF-32LE.				
Puntos Estimados:	2		Puntos Reales:	2	

Tabla 2.2. HU-5. Construir vector de términos

Historia de Usuario					
Numero de Historia:	HU-5	Usuario:	Programadores	Fecha:	13/04/2015
Título:	Construir vector de términos				
Descripción:	El componente debe construir el vector asociado a cada documento de la colección de entrada. Para la construcción del vector de un documento recibe la sucesión de los términos que conforman el texto. El resultado es un vector que contiene cada término de la colección.				
Observaciones:	El texto de entrada debe estar previamente filtrado por Stop Filter, Spanish Character Filter y Stem Filter.				
Puntos Estimados:	3		Puntos Reales:	3	

Tabla 2.3. HU-7. Construir matriz de semejanza

Historia de Usuario					
Numero de Historia:	HU-8	Usuario:	Programadores	Fecha:	13/04/2015
Título:	Construir matriz de semejanza				
Descripción:	El componente debe construir la matriz de semejanza asociada a la colección de documentos.				

Observaciones:	Para la construcción de la matriz de semejanza debe haber culminado el proceso de construcción de todos los vectores de la colección.		
Puntos Estimados:	3	Puntos Reales:	3

2.1.2 Tiempo de ejecución del proyecto

El tiempo de ejecución del proyecto es una medida que tiene el equipo de desarrollo para completar las historias de usuario en una determinada iteración. Dicha medida se calcula totalizando los puntos estimados de las historias de usuario realizadas en una iteración. En este sentido un punto, equivale a una semana ideal de programación donde se trabaje sin interrupciones (44) (45) (46). La planificación se realiza basándose en el tiempo de implementación de una historia de usuario. El tiempo de ejecución se utiliza para establecer cuántas historias de usuario se pueden implementar antes de una fecha determinada o cuánto tiempo tomará implementar un conjunto de historias. La tabla 2.4 presenta la estimación de esfuerzo por historia de usuario propuesta para el desarrollo de la aplicación:

Tabla 2.4. Estimación por historia de usuario

Número	Título de Historia de Usuario	Estimación en Puntos
HU-1	Recibir colección de entrada	1
HU-2	Estandarizar colección de documentos	2
HU-3	Filtrar colección de entrada	2
HU-4	Realizar stemming a la colección de entrada	2
HU-5	Construir vector de términos	3
HU-6	Calcular peso asociado a cada término obtenido	2
HU-7	Construir diccionario de términos	3
HU-8	Construir matriz de semejanza	3
HU-9	Calcular similitud entre representaciones vectoriales	3
HU-10	Almacenar diccionario de términos	1
HU-11	Almacenar matriz de semejanza	1

2.1.3 Iteraciones

En la metodología XP, el desarrollo del software se divide en etapas para facilitar su realización, dichas etapas se nombran iteraciones. Para cada iteración se define un número determinado de historias de usuario a implementar y al final de cada iteración se obtiene como resultado la entrega de las funcionalidades correspondientes (27). Esto ocurre conjuntamente con la aceptación por parte del cliente de los requerimientos solucionados. De acuerdo con lo planteado por la metodología y en consecuencia con el proceso de desarrollo se divide el mismo en cuatro iteraciones. El número de iteraciones así como la cantidad de historias de usuario de cada una, constituyen el comienzo y fin de cada fase del proceso de representación del corpus textual. La tabla 2.5 muestra el plan de iteraciones propuesto por el equipo de desarrollo.

Tabla 2.5. Plan estimado de duración de las iteraciones

Iteración	Título de Historia de Usuario	Duración Total
1	Recibir colección de entrada, Estandarizar colección de documentos, Filtrar colección de entrada	5
2	Realizar stemming a la colección de entrada, Construir vector de términos, Calcular peso asociado a cada término obtenido.	8
3	Construir diccionario de términos, Construir matriz de semejanza, Calcular similitud entre representaciones vectoriales.	9
4	Almacenar diccionario de términos, Almacenar Matriz de Semejanza	2
Total		24

El plan de iteraciones mostrado en la tabla 2.5 se confecciona teniendo en cuenta la complejidad de las funcionalidades involucradas, las pruebas y la necesidad de producir rápidamente versiones del sistema que sean operativas. Para la investigación se define una versión operativa del sistema como una unidad atómica del componente desarrollado que se comporta de forma independiente y delimita el final de una iteración. Objetivamente cada iteración constituye una parte fundamental del sistema a desarrollar, posee una entrada y salida determinada consecuentemente con la iteración contigua. Como se observa en la figura 2.1 el modelo de la propuesta de solución cuenta con 4 etapas principales que culminan en una versión operativa del mismo. Conjuntamente las pruebas planificadas durante el proceso de desarrollo necesitan de partes individuales completamente funcionales para verificar cada una de las etapas fundamentales. Además el tiempo estimado para cada iteración no supera el máximo establecido por la metodología XP de 3 meses (27) y la complejidad de las tareas permite agruparlas en el mismo orden del flujo de datos.

2.1.4 Plan de entrega

El plan de entrega consiste en establecer conjuntamente el equipo de desarrollo y el cliente la duración y fecha de entrega de cada iteración hasta lograr el producto final. En este plan se detalla la fecha fin de cada iteración, mostrando una versión desarrollada del producto en ese momento, hasta lograr el producto final en la fecha establecida (27). A partir del plan de iteraciones y la fecha de inicio del proceso de desarrollo se conforma el plan de entrega coincidiendo cada entrega con el fin de una iteración. La tabla 2.6 muestra el plan de entrega definido por el equipo de desarrollo.

Tabla 2.6. Plan de entrega de las iteraciones

Iteraciones	Fecha de entrega
Iteración 1	20 de enero del 2015
Iteración 2	24 de febrero del 2015
Iteración 3	7 de abril del 2015
Iteración 4	21 de abril del 2015

El plan de entrega mostrado en la tabla 2.6 se confecciona teniendo en cuenta la necesidad de realizar entregas operativas del sistema, cada entrega es una versión incremental del mismo. Para la primera fecha el sistema debe ser capaz de realizar la estandarización y filtrado de la colección de entrada. Una vez culminado este proceso, para la segunda entrega el sistema debe permitir confeccionar los términos y vectores de cada documento de la colección, así como sus pesos asociados. Para la tercera entrega se deben utilizar las estructuras de datos diccionario y matriz de semejanza, para la representación del corpus textual de los documentos. En la entrega final el producto debe ser completamente funcional, permitiendo la realización completa del proceso de representación de los textos.

2.2 Diseño de la propuesta de solución

El diseño se encuentra en el núcleo técnico de la ingeniería de software. Una vez que se analizan y especifican los requisitos del software, esta es la primera de las tres actividades técnicas que se requieren para construir y verificar el software (47). En esta sección se realiza la descripción de la arquitectura de software y los patrones de diseño aplicados a la propuesta de solución. Se presenta el diagrama de clases persistentes del componente a desarrollar y se expone brevemente en qué consisten las tarjetas clase-responsabilidad-creador, así como la descripción mediante el uso de dichas tarjetas de las principales clases del sistema.

Estos elementos muestran una vista más técnica y lógica del software a desarrollar, describen la arquitectura utilizada basada en las necesidades de la solución, así como las clases y sus relaciones. Además permiten la definición de una visión más específica sobre las funcionalidades a desarrollar y son el punto de partida para la posterior codificación.

2.2.1 Arquitectura de software

De acuerdo al Instituto de Ingeniería de Software (SEI por sus siglas en inglés), la arquitectura de software se refiere a las estructuras de un sistema, compuestas de elementos con propiedades visibles de forma externa y las relaciones que existen entre ellos. Los elementos son entidades que existen en tiempo de ejecución (objetos, hilos), entidades lógicas que existen en tiempo de desarrollo (clases, componentes) y entidades físicas (nodos, directorios). Las relaciones entre elementos dependen de sus propiedades visibles, quedando ocultos los detalles de implementación (48). Por lo cual implementar una arquitectura de software permite representar el diseño a alto nivel del sistema a desarrollar y sirve como guía para el proceso de desarrollo de software. Además permite dividir el sistema en estructuras arquitectónicas relacionadas, posibilitando la reutilización y el acoplamiento entre las mismas.

Arquitectura N capas

El estilo arquitectónico en n capas se basa en una distribución jerárquica de los roles y las responsabilidades para proporcionar una división efectiva de los problemas a resolver. Los roles indican el tipo y la forma de la interacción con otras capas y las responsabilidades la funcionalidad que implementan. La programación por capas tiene como objetivo principal separar la lógica de negocios de la de diseño, esto se aprecia al separar la capa de datos de la de negocio y la de negocio de la de presentación (29).

El diseño empleado en la presente investigación es el diseño en tres capas. Aunque la solución implementada no define las capas clásicas de presentación, control y negocio, si cuenta con una capa dedicada a la captura de datos para obtener la colección de documentos a procesar y tratarla. Las otras dos capas se encargan de procesar y almacenar el resultado de la representación del corpus textual de los archivos proporcionados. En la figura 2.3 se ilustra la arquitectura de la propuesta de solución.

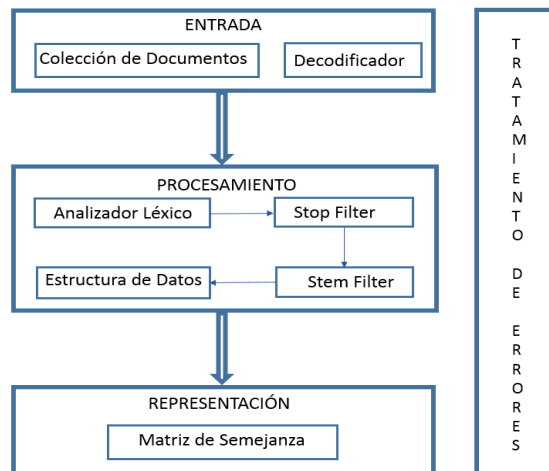


Figura 2.3. Arquitectura de la propuesta de solución

Como se muestra en la figura 2.3, el diseño de la arquitectura está dividido en tres capas: entrada, procesamiento y representación. En la primera capa se prepara la colección de documentos atendiendo a la codificación de los textos. El decodificador selecciona los documentos siguiendo los estándares de codificación permitidos y estos son enviados a la capa intermedia. La capa de procesamiento es la responsable de las operaciones de la aplicación, en ella se realizan los distintos tipos de filtros y se conforman las estructuras de datos que son entregadas a la capa de representación. Dicha capa es la encargada de representar la matriz de semejanza como resultado final del proceso. El diseño presentado explota las ventajas del uso de los patrones de asignación

de responsabilidades. El bajo acoplamiento se evidencia al mantener la independencia entre las capas de la aplicación, esto permite que los cambios que ocurran en una capa no repercutan en el resto. Cada componente de las capas se encarga de mantener las responsabilidades asignadas lógicamente evidenciando la alta cohesión.

2.2.2 Diagrama de Clases

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases. Son utilizados durante el proceso de diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará y los componentes que se encargarán del funcionamiento así como sus relaciones (49). Por tal motivo el modelado del diagrama de clases permite en términos visuales la descripción del modelo así como los atributos y comportamientos que poseen los objetos que intervienen en el diseño. De igual manera permite reflejar las relaciones en el uso del enfoque orientado a objetos tales como: generalizaciones, agregaciones, y asociaciones que son de vital importancia para el diseño del sistema.

La figura 2.4 ilustra el diagrama de clases que representa la estructura estática del sistema, así como las relaciones y dependencias entre cada una de las clases que lo conforman. Dicho diagrama contribuye a la documentación del diseño de la propuesta de solución.

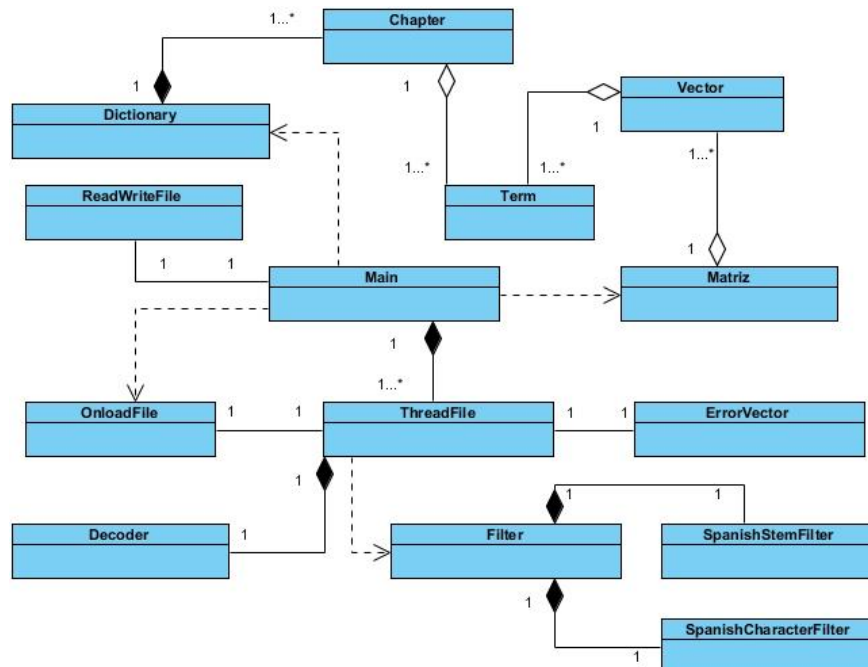


Figura 2.4. Diagrama de clases UML

El sistema a implementar cuenta con una jerarquía de clases asociada a las tareas necesarias para la representación del corpus textual. Dicha jerarquía está formada por la clase principal Main que se

encarga de crear y controlar la ejecución del flujo principal del sistema. La misma tiene una relación de asociación con la clase `ReadWriteFile`, una asociación por composición con la clase `ThreadFile`, así como relaciones de uso con las clases `Dictionary`, `OnloadFile` y `Matriz`. La clase `ReadWriteFile` tiene la responsabilidad de la lectura y escritura de los ficheros donde se guardan el diccionario y la matriz de datos.

La clase `ThreadFile` se encarga del proceso de lectura y filtrado de la colección de entrada. Esta se relaciona con `OnloadFile` y `ErrorVector` por asociación, con `Decoder` a través de una asociación por composición y con `Filter` mediante una relación de uso. Para la colección de entrada es necesario construir un diccionario de términos y de esto se responsabiliza la clase `Dictionary`. La construcción del mismo se realiza a partir del vocabulario de los textos procesados.

La tarea de carga de la colección de entrada la realiza la clase `OnloadFile`. El resto de las clases brindan el soporte necesario para el procesamiento de los textos tales como: `Decoder` que se especializa en obtener la codificación de cada archivo de la colección y `ErrorVector` que contiene un listado de los textos que no son procesados.

A cada documento se le aplican una serie de filtros para depurar su contenido, eliminando caracteres extraños, palabras nulas y estandarizando el mismo a minúsculas, así como la conformación de los *tokens* y términos. De estas tareas se encargan las clases `Filter` de conjunto con `SpanishCharacterFilter` y `SpanishStemFilter` con las que posee relaciones de asociación por composición.

Se muestra también la clase `Matriz` que se relaciona con `Vector` mediante una asociación por agregación y contiene la matriz de todos los términos de la colección de documentos. `Vector` tiene además una relación de asociación por agregación con `Term` y es la encargada de construir el vector para cada documento de la colección de entrada. La construcción la hace término a término, documento a documento. Recalcula con cada aparición del término su peso dentro del documento y a cada vector construido le asigna un identificador único dentro de la colección. `Term` contiene la información de un término específico, posee una asociación por agregación con `Chapter` y esta a su vez almacena en varias listas todos los términos que inicien con una misma letra del alfabeto. El anexo número 2 ilustra el diseño de cada una de las clases mencionadas anteriormente para una mejor comprensión de sus atributos y métodos.

2.2.3 Patrones de diseño

Los patrones de diseño son soluciones para problemas típicos y recurrentes que enfrentan los programadores al desarrollar una aplicación. Son soluciones probadas y documentadas que

explican cómo resolver un determinado problema bajo determinadas circunstancias. Por lo que constituyen una guía para el rápido desarrollo de software (50). Por tanto un patrón de diseño es una solución demostrada, eficiente y reutilizable en diferentes contextos y escenarios de desarrollo de software.

Patrones de Asignación de Responsabilidades (GRASP)

Indican que la responsabilidad de la creación de un objeto o la implementación de un método debe recaer sobre la clase que conoce toda la información necesaria para crearlo (29). De este modo se obtiene un diseño con mayor cohesión y así la información se mantiene encapsulada (disminución del acoplamiento). Los patrones analizados en esta sección son: Experto, Creador, Alta Cohesión y Bajo Acoplamiento.

Experto

El experto en información establece el principio básico de asignación de responsabilidades. Indica que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo (29). Por tanto la utilización de este patrón conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. El comportamiento se distribuye entre las clases que cuentan con la información requerida. La figura 2.5 muestra el uso de este patrón.

```

public class Decoder {
    public Decoder(FileInputStream fileInputStream) {...}

    public String getEncoding() throws IOException {
        int nread;
        final byte[] buf = new byte[4096];
        while ((nread = fileInputStream.read(buf)) > 0 && !detector.isDone()) {
            detector.handleData(buf, 0, nread);
        }
        detector.dataEnd();
        if (detector.isDone()) {
            return detector.getDetectedCharset();
        }
        return "";
    }
}

```

Figura 2.5. Implementación del patrón Experto

Como se observa en la figura 2.5 la clase Decoder es la encargada de la decodificación de la colección de entrada. Esta cuenta con toda la información necesaria para la implementación del método `getEncoding()` encargado de obtener la codificación de los documentos de entrada.

Creador

El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos. El propósito fundamental es encontrar un creador que se debe conectar con el objeto producido en cualquier evento (29). Por tanto este patrón pretende lograr un bajo acoplamiento (patrón que se describe más adelante) lo que supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización. La figura 2.6 muestra el uso de este patrón.

```

public class Vector implements Serializable {
    public Vector(String url) {
        listTerm = new ArrayList<Term>();
        this.url = url;
        key = UUID.randomUUID().toString();
    }

    public boolean add(String stem) {
        int sizeV = listTerm.size();
        for (int i = 0; i < sizeV; i++) {
            if (listTerm.get(i).getStem().equals(stem)) {
                listTerm.get(i).incrementTF();
                return false;
            }
        }
        return listTerm.add(new Term(stem));
    }
}

```

Figura 2.6. Implementación del patrón Creador

Como se observa en la figura 2.6 la clase Vector es la encargada de la creación de los objetos tipo términos mediante una asociación por agregación con la clase Term (figura 2.4).

Alta Cohesión

La alta cohesión indica que los datos y responsabilidades de una entidad están fuertemente ligados a la misma en un sentido lógico. La información que maneja una entidad de software tiene que estar conectada lógicamente con esta, no deben existir entidades con atributos que describan comportamientos que en realidad no le corresponden (29). Este patrón permite la obtención de clases fácilmente mantenibles y reutilizables durante el proceso de desarrollo de software. La figura 2.7 muestra el uso del mismo.

```

+ public Chapter(char index) {...}
+ public List<Term> getTopic(String stem) {...}
- public boolean addTerm(String stem, int tf) {
    List<Term> list = getTopic(stem);
    int size = list.size();
    for (int i = 0; i < size; i++) {
        if (stem.equals(list.get(i).getStem())) {
            list.get(i).incrementTF(tf);
            return false;
        }
    }
    return list.add(new Term(stem));
}

```

Figura 2.7. Implementación del patrón Alta Cohesión

En la figura 2.7 se muestra la clase Chapter encargada de mantener una lista con las instancias de tipo Term en el diccionario, cada instancia necesita recalcularse su frecuencia de aparición mediante el método `incrementTF()`. La responsabilidad de implementar este método le corresponde a la clase Term teniendo en cuenta lo que plantea el patrón Experto, manifestándose además la alta cohesión entre las clases ya que la entidad Chapter no lo implementa, solo hace uso del método.

Bajo Acoplamiento

El bajo acoplamiento sostiene la idea de mantener las entidades y clases lo menos ligadas entre sí posible, de tal forma que en caso de producirse modificaciones en alguna de ellas, la repercusión sea la mínima posible en el resto de las entidades o clases (29). Por tanto este patrón potencia la reutilización, y disminuye la dependencia entre clases. Soporta un diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que acrecientan la oportunidad de una mayor productividad. El ejemplo analizado en la figura 2.7 soporta tanto la alta cohesión como el bajo acoplamiento ya que la responsabilidad de calcular la frecuencia de aparición de un término recae sobre la clase Term por lo que una modificación en este proceso no repercute en el resto de las entidades.

Patrones de Creación (GOF)

Los patrones de creación muestran la guía de cómo crear objetos cuando sus creaciones requieren tomar decisiones. Estas decisiones normalmente serán resueltas dinámicamente decidiendo que clases instanciar o sobre que objetos un objeto delegará responsabilidades (51). La valía de los patrones de creación ilustra como estructurar y encapsular estas decisiones. En esta sección se analiza el patrón Singleton, la aplicación del mismo se describe en la sección 2.3.2.

Singleton

El patrón Singleton es propio de la programación orientada a objetos y consiste en crear una sola instancia de un objeto para toda la aplicación. Puede considerarse como una especie de variable global que almacena un objeto de interés (51). Este patrón garantiza que durante toda la ejecución de la aplicación solo se pueda crear una única instancia de un objeto y provee un punto de acceso global al mismo. Todos los objetos que utilizan una instancia de esa clase usan la misma instancia.

Patrones de Concurrencia

Los patrones de esta categoría permiten coordinar las operaciones concurrentes. Estos patrones se dirigen principalmente a dos tipos diferentes de problemas:

- ✓ Recursos compartidos: Cuando las operaciones concurrentes acceden a los mismos datos u otros tipos de recursos compartidos, puede darse la posibilidad de que las operaciones interfirieran unas con otras si acceden a los recursos al mismo tiempo. Para garantizar que cada operación se ejecuta correctamente, la operación debe ser protegida para acceder a los recursos compartidos en solitario (51).
- ✓ Secuencia de operaciones: Si las operaciones son protegidas para acceder a un recurso compartido una cada vez, entonces podría ser necesario garantizar que ellas acceden a los recursos compartidos en un orden particular. Por ejemplo, un objeto nunca será borrado de una estructura de datos antes de que este sea añadido a la estructura de datos (51).

Por tal motivo aplicar patrones de concurrencia dentro de la programación permite que el sistema gestione de manera eficiente los recursos de hardware disponibles, microprocesador, memoria y disco duro. La utilización de cada uno de ellos corresponde con el límite físico de sus capacidades. Permite que procesadores multinúcleos ejecuten de forma paralela una secuencia de tareas determinadas e influye considerablemente en el tiempo de ejecución y respuesta de los sistemas informáticos. En esta sección se analizan los patrones Single Threaded Execution y Scheduler, la aplicación de los mismos se describe en la sección 2.3.1.

Single Threaded Execution

Algunos métodos acceden a datos u otros recursos compartidos de tal forma, que producen resultados incorrectos si hay llamadas concurrentes y acceden a los datos o recursos compartidos al mismo tiempo. Este patrón soluciona este problema impidiendo las llamadas concurrentes a un método que provocan ejecuciones concurrentes del mismo (51). Esto permite que el acceso a los recursos compartidos se realice de forma organizada, lo que evita la pérdida de datos y posibles errores en tiempo de ejecución.

Scheduler

Controla el orden en el cual los hilos son organizados para ejecutar el código. Potencia el empleo de un hilo único, utilizando un objeto que sigue explícitamente en la sucesión de hilos que esperan. El patrón Scheduler provee un mecanismo para implementar una política de organización. Esto es independiente de cada una de las políticas de organización específicas (51).

2.2.4 Tarjetas Clase-Responsabilidad-Creador

La utilización de tarjetas clase-responsabilidad-creador (CRC) es una técnica de diseño propuesta por la metodología XP. Consiste en hacer, mediante determinadas tarjetas, un inventario de las clases que se necesitan para implementar el sistema y la forma en que interactúan (27). De este modo se pretende facilitar el análisis y discusión de las mismas por parte del equipo de proyecto con el objetivo que el diseño sea lo más simple posible verificando las especificaciones del sistema.

Las tarjetas CRC contienen una serie de campos que deben ser llenados por el equipo de desarrollo, ellos son (46):

- ✓ Nombre de la Clase: especifica el nombre de la clase en el sistema.
- ✓ Responsabilidades de la Clase: describen a alto nivel el propósito de la existencia de la clase. Una clase no debe tener más de tres o cuatro responsabilidades.
- ✓ Colaboradores de la Clase: describe las clases con las que va a colaborar para ejecutar las responsabilidades.

Las tablas 2.7, 2.8 y 2.9 describen mediante tarjetas CRC las clases más importantes que intervienen en el desarrollo de la aplicación:

Tabla 2.7. Tarjeta CRC. Clase ThreadFile

Nombre de la Clase: ThreadFile
Responsabilidades de la Clase: ThreadFile se encarga de la lectura y filtrado de la colección de entrada. También invoca a las clases encargadas de construir las estructuras de datos representativas para la aplicación.
Colaboradores de Clase
OnloadFile, Decoder, Dictionary, Matriz, Filter, Vector, SpanishCharacterFilter, TokenStream, TermAttribute

Tabla 2.8. Tarjeta CRC. Clase Vector

Nombre de la Clase: Vector
Responsabilidades de la Clase: Esta clase es la encargada de construir el vector para cada documento de la colección de entrada. La construcción se realiza término a término, documento a documento.
Colaboradores de Clase
Term, ThreadFile, Dictionary, Matriz

Tabla 2.9. Tarjeta CRC. Clase Dictionary

Nombre de la Clase: Dictionary
<p>Responsabilidades de la Clase: Esta clase se encarga de construir el diccionario de términos a partir del vocabulario de la colección de entrada.</p>
Colaboradores de Clase
Term, Chapter, ThreadFile, Main, ReadWriteFile

2.3 Implementación de la Propuesta de Solución

En esta sección se describe implementación de la propuesta de solución. Para ello se detalla el algoritmo implementado para la representación del corpus textual de los documentos digitales, así como las estructuras de datos y los patrones de diseño utilizados durante la codificación. Tal algoritmo sigue la base teórica del modelo de espacio vectorial, pero incluye mejoras potenciales en el filtrado del corpus, definición y almacenamiento de términos así como cálculo del peso asociado al vector. Además intenta en toda su extensión reducir la dimensión de los vectores obtenidos con la ayuda de estructuras de datos específicas y el apoyo de la programación concurrente.

La tabla 2.10 muestra el pseudocódigo del algoritmo desarrollado para la representación del corpus textual de documentos digitales textuales. Se asume que comienza su ejecución a partir de una colección de documentos que contengan texto plano escrito en español. Dicha colección constituye la entrada al algoritmo propuesto.

Tabla 2.10. Pseudocódigo. Algoritmo para la representación de corpus textual

Algoritmo para la representación de corpus textual	
1	Inicio
2	CD ← Entrada (Colección de documentos)
3	SW ← {Stop Word del idioma, definido para el algoritmo}
4	V ← {Vector del documento, inicia en vacío}
5	Q ← {Diccionario de términos, inicia en vacío}
6	M ← {Matriz de Semejanza, inicia en vacío}
7	Para cada D de CD hacer
8	D ← StopFilter (SW,D)
9	D ← Stemming (D)
10	Para cada T de D hacer
11	Si T pertenece V
12	entonces Recalcular (T, V)
13	Sino V ← Vector (T)
14	Si T pertenece Q
15	entonces Recalcular (T, Q)
16	Sino Q ← Dictionary (T)
17	M ← Matriz (V)
18	M ← Weigh (M)
17	Salida (M)
18	Fin

Entre las líneas 2 y 6 del pseudocódigo mostrado se inicializan las variables y estructuras de datos necesarias para la corrida del algoritmo. Inicialmente CD recibe la colección de documentos de entrada a procesar y SW contiene el Stop Word (palabras nulas) con los términos definidos para la corrida del algoritmo. En la línea 4, 5 y 6 se crean las estructuras de datos que son utilizadas para la representación del corpus textual de los documentos de entrada. A partir de la línea 7 comienza la corrida del algoritmo, donde para cada D “documento” de la colección se realizan las siguientes operaciones:

StopFilter → Es un mecanismo de filtrado que elimina SW de D. Retorna D sin la aparición de SW.

Stemming → Motor de extracción de la raíz semántica de todos los términos por cada lectura de documento. Retorna D con todos los T “términos” llevados a la raíz semántica.

Una vez llegado a este punto, en la línea 11 para cada T de D comienza la construcción del vector del documento y puede ocurrir una de dos opciones.

- a) Si T pertenece a V entonces recalcular pesos.
- b) Sino añadir T a V.

Similar a la construcción del vector en la línea 14 se comienza a introducir los términos en el diccionario y puede ocurrir una de dos opciones.

- a) Si T pertenece a Q entonces recalcular pesos.
- b) Sino añadir T a Q.

En la línea 17 el mecanismo encargado de la construcción de la matriz de semejanza se encarga de insertar el nuevo vector formado. Una vez que se tengan todos los vectores de la colección de entrada, se actualizan los pesos de todos los términos en cada vector en la matriz **Weigh (M)**.

La tabla 2.11 muestra el pseudocódigo del procedimiento para la actualización de los pesos de los términos de los vectores en la matriz de semejanza.

Tabla 2.11. Pseudocódigo. Procedimiento para actualizar pesos

Procedimiento para actualizar los pesos	
1	Weigh(M,D)
2	Inicio
3	S ← Size(M)
4	Para cada V de M hacer
5	Para cada T de V hacer
6	TF ← Frecuencia (T)
7	NI ← Frecuencia (T,D)
8	TF ← TF * Log S / NI
9	Retorna M

En la línea 3, S representa los vectores que conforman la matriz de semejanza. A partir de esto para cada vector de la matriz se obtienen todos sus términos y se recalculan sus pesos, donde $TF \leftarrow \text{Frecuencia}(T)$ es la frecuencia de aparición del término T en la colección y $NI \leftarrow \text{Frecuencia}(T,D)$ es la frecuencia de aparición del término T en el vector. Cada peso se calcula como la frecuencia de aparición de T en la colección por el logaritmo de la frecuencia invertida de aparición de T en el documento.

El algoritmo representado supone una mejora con respecto al diseño teórico del algoritmo original del modelo de espacio vectorial en cuanto al uso de los recursos de hardware, ya que incluye un mejor aprovechamiento de los mismos. Esto se debe a la implementación de patrones concurrentes durante todo el proceso de representación de la colección de entrada. Las particularidades del uso de la concurrencia en la implementación del algoritmo se especifican en las secciones 2.3.1 y 2.3.2.

2.3.1 Programación Concurrente

La programación concurrente es la simultaneidad en la ejecución de múltiples tareas interactivas. Estas tareas pueden ser un conjunto de procesos o hilos de ejecución creados por un único programa. Los programas de software concurrentes tienen más de una línea lógica de ejecución y permiten que varias partes del código se ejecuten simultáneamente (52). Esto permite que arquitecturas multiprocesadores ejecuten al menos tantas tareas como cantidad de procesadores posean.

La concepción del software como programa concurrente puede conducir a una reducción del tiempo de ejecución. Cuando se trata de un entorno monoprocesador, permite solapar los tiempos de entrada/salida o de acceso al disco de unos procesos con los tiempos de ejecución de procesador de otros procesos. Cuando el entorno es multiprocesador, la ejecución de los procesos es realmente simultánea en el tiempo, y esto reduce el tiempo de ejecución del programa. También permite mayor flexibilidad de planificación, y con ella, los procesos con plazos límites de ejecución más urgentes pueden ser ejecutados antes de otros de menor prioridad (52).

Implementación de la concurrencia

El algoritmo descrito en la sección 2.8 no evalúa el factor tiempo para la representación del corpus textual de la colección de entrada. La figura 2.5 muestra dos corridas del algoritmo en cada instante. La colección de entrada varía desde 1000 hasta 6000 documentos para cada par de corridas.

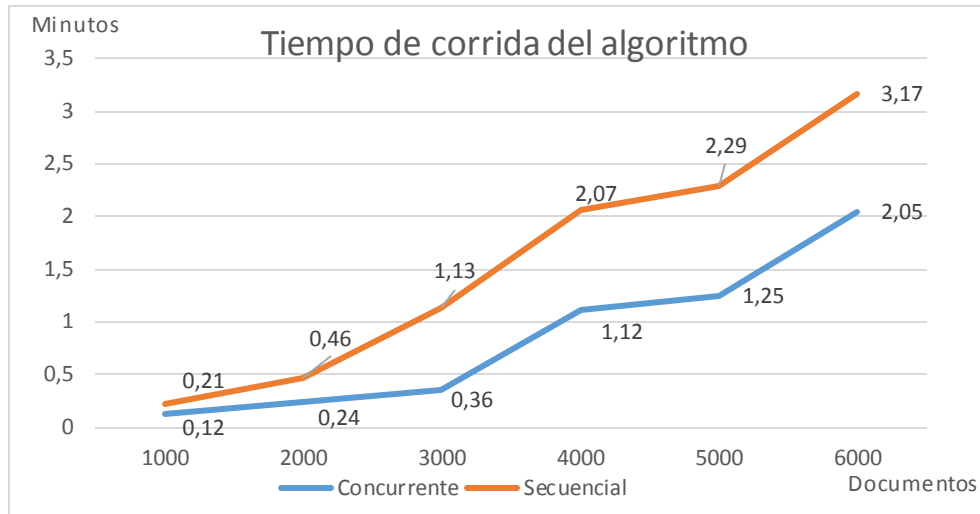


Figura 2.8. Tiempo de corrida del algoritmo

Como se observa en la gráfica, el tiempo de corrida secuencial es mayor en todos los casos que la corrida concurrente. Esto se debe a que a medida que aumenta el tamaño de la colección aumenta el número de operaciones secuenciales que debe realizar el componente. Para el algoritmo implementado se fija la concurrencia en la construcción de las estructuras de datos. Esto se realiza de manera que una arquitectura multiprocesador permite construir simultáneamente un máximo de estructuras por vez. Aun así, la evidencia demuestra que textos muy extensos pueden presumir tiempos de corrida elevados. La solución a este inconveniente se explica en la sección 2.3.2 al tratar el diccionario de datos.

Para el control de la concurrencia se emplean los patrones: Single Threaded Execution y Scheduler. Las figuras 2.9 y 2.10 muestran su implementación respectivamente.

```

public class Chapter implements Serializable{
    + INDICES DE LOS CAPITULOS
    private static char index;

    public Chapter(char index) {

        this.index = index;
        list1_3 = Collections.synchronizedList(new ArrayList<Term>(100));
        list4_6 = Collections.synchronizedList(new ArrayList<Term>(1000));
        list7_9 = Collections.synchronizedList(new ArrayList<Term>(1000));
        list10_12 = Collections.synchronizedList(new ArrayList<Term>(1000));
        list13_15 = Collections.synchronizedList(new ArrayList<Term>(1000));
        list16_18 = Collections.synchronizedList(new ArrayList<Term>(100));
        list19_21 = Collections.synchronizedList(new ArrayList<Term>(100));
        list22_UP = Collections.synchronizedList(new ArrayList<Term>(100));
    }
}
    
```

Figura 2.9. Implementación del patrón Single Threaded Execution

La clase Chapter que se muestra en la figura 2.9 es la encargada de almacenar las listas con los términos del vocabulario de la colección. El constructor de la clase es el encargado de establecer la sincronización en el acceso a cada lista. El método `synchronizedList()` restringe el acceso a los objetos almacenados en tiempo de ejecución cuando más de un hilo intenta realizar modificaciones sobre ellos, para esto el primer hilo en apoderarse del objeto establece un bloqueo hasta que termina sus operaciones. Sin embargo la lectura puede realizarse de forma simultánea sin afectar la integridad o pérdida de objetos en la colección.

```

public class Main implements Runnable {
    + public Main() {...}

    @Override
    - public void run() {
    +     INICIALIZACIÓN DE LOS HILOS
        thread0.start();
        thread1.start();
        thread2.start();
        thread3.start();

        try {
            thread0.join();
            thread1.join();
            thread2.join();
            thread3.join();
        } catch (InterruptedException ex) {
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}

```

Figura 2.10. Implementación del patrón Scheduler

La clase Main que se muestra en la figura 2.10 es el hilo principal del programa. Esta controla el orden en el cual los hilos hijos son organizados para ejecutar el código del hilo único, utilizando un objeto que sigue explícitamente en la sucesión de hilos que esperan. El método `start()` inicia la ejecución del hilo que lo invoca y el método `join()` le indica al hilo padre que debe esperar a que termine el hilo hijo para continuar su ejecución.

2.3.2 Estructuras para la representación del corpus textual

El diccionario de datos es una estructura que crea un índice para cada término de la colección. Su propósito es llevar el recuento de la cantidad de apariciones de cada término, así como la de construir el vocabulario de las palabras de la colección. Para ello cada término se ordena por su letra inicial. El objetivo de esta técnica es mejorar la eficiencia en el almacenamiento, al no realizar una búsqueda exhaustiva para reconocer si un término existe o no en el diccionario. También permite que varios hilos concurrentes recorran la estructura sin afectar la integridad de los datos, mientras que un hilo busca o inserta un término indexado con la letra A, otro puede hacer lo mismo con la

letra B. Aunque esta técnica supone una mejora en el tiempo de corrida, se desconoce la densidad de las listas construidas. Como es de esperar existe una desigual distribución en la cantidad de términos dada su letra inicial.

El diccionario de la real academia española de la lengua contiene 88 500 entradas, se suman las conjugaciones de cada verbo, y las diferentes flexiones para adjetivos, nombres y adverbios, para una suma de 750 000 palabras (53). A partir del diccionario se construye una bolsa de palabras con un total de 631 146 palabras y se contabilizan cuántas empiezan por cada una de las letras del alfabeto. Los resultados son los que se muestran en la figura 2.11.

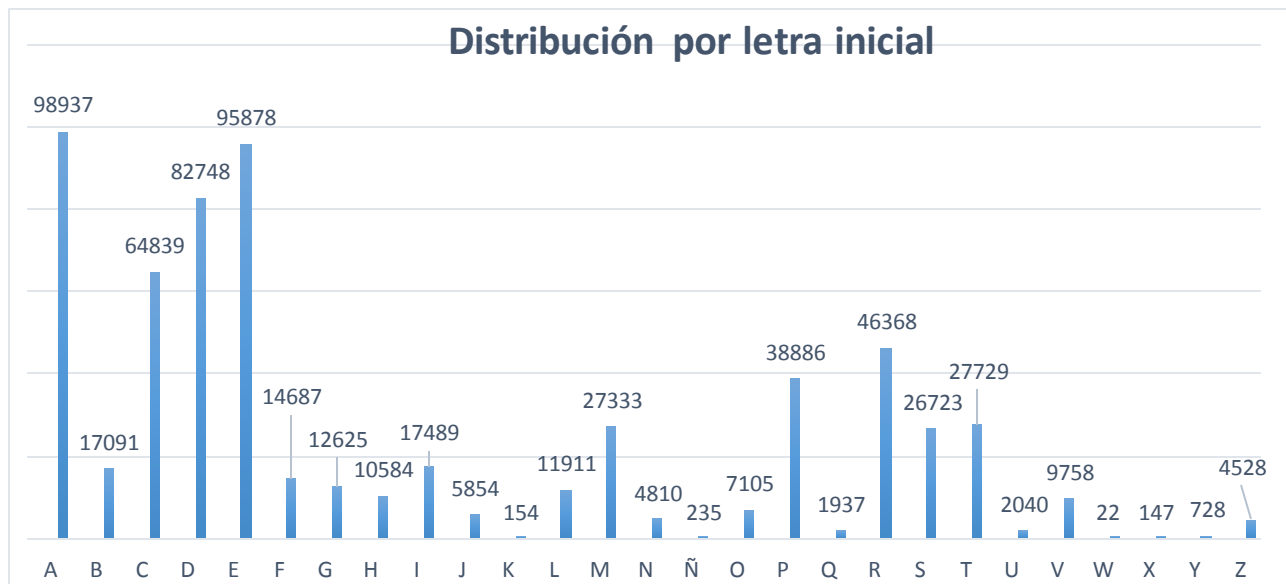


Figura 2.11. Distribución de palabras por letra inicial

Como se observa en la figura 2.11, existe una gran diferencia en la distribución de las primeras letras del alfabeto. Las letras A, C, D y E son las que más palabras inician, por tanto existe una alta probabilidad de que durante la construcción del diccionario las listas encargadas de almacenar estos términos serán las más cargadas y de mayor concurrencia. Para resolver este problema se particionan las listas y se protege su acceso, así mientras un hilo realiza alguna operación sobre un sector, este bloquea la entrada hasta que termine. El resultado de esta técnica es la disminución del tiempo de búsqueda en la estructura. La corrida del algoritmo implementando el diccionario para un total de 600 000 palabras se muestra en la figura 2.12.

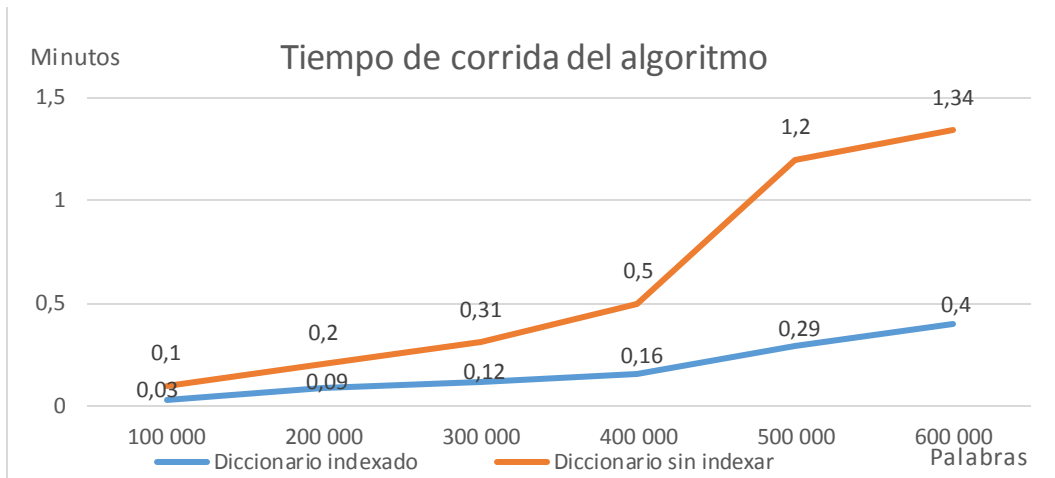


Figura 2.12. Tiempo de corrida del algoritmo

Como se muestra en la figura 2.12 para cada par de corridas del algoritmo el tiempo de respuesta del sistema es siempre menor con el diccionario indexado. Por tanto el uso de técnicas de indexación solventa el tiempo de respuesta de las búsquedas en el diccionario.

Para la creación y control de las operaciones en el diccionario se implementa el patrón Singleton. Esto asegura, tal como lo definido en el patrón, un acceso único y global al objeto de la clase Dictionary. En la figura 2.13 se muestra su implementación.

```

private static Dictionary dictionary;

private Dictionary() {...}

public static Dictionary getDictionary () {
    if (dictionary == null) {
        dictionary = new Dictionary();
    }
    return dictionary;
}
    
```

Figura 2.13. Implementación del patrón Singleton

Como se observa en la figura 2.13 solamente se permite crear una sola instancia del objeto Dictionary. El constructor privado y el método getDictionary () constituyen los mecanismos para la construcción de la única instancia Dictionary. La primera vez que se invoca getDictionary () crea una instancia de la clase Dictionary, cualquier otra llamada a este método no crea un objeto de este tipo, sino que retorna el objeto dictionary ya creado.

Matriz de semejanza

La matriz de semejanza es una estructura de datos en forma de matriz documento–término que contiene el vocabulario de la colección de referencia y los documentos existentes. En la intersección

de un término y un documento se almacena un valor numérico de importancia del término en el documento; tal valor representa su poder de discriminación. En teoría, los documentos que contengan términos similares estarán a muy poca distancia entre sí sobre tal espacio de búsqueda. Para calcular la semejanza entre los vectores que representan los documentos se utiliza la función de similitud del coseno (7).

La función del coseno establece una medida de la similitud entre dos vectores en un espacio que posee un producto interior con el que se evalúa el valor del coseno del ángulo comprendido entre ellos. Esta función trigonométrica proporciona un valor igual a 1 si el ángulo comprendido es 0, es decir si ambos vectores apuntan a un mismo lugar. Cualquier ángulo existente entre los vectores, el coseno es un valor inferior a 1. Si los vectores son ortogonales el coseno se anula y si apuntan en sentido contrario su valor es -1. De esta forma, el valor de esta métrica se encuentra en el intervalo cerrado [-1,1] (7) (54).

El proceso de equiparación de los documentos es posible cuando en los vectores existen términos coincidentes. Por tal motivo es necesario determinar el ángulo que forman los vectores de los documentos que se están comparando. A continuación se enuncia la función de similitud del coseno (7):

$$\text{Cos}(d1,d2) = \frac{\sum_{n=1} (P(n,d1) \times P(n,d2))}{\sqrt{\sum_{n=1} P(n,d1)^2 \times \sum_{n=1} P(n,d2)^2}}$$

Por lo tanto, la función aplicada para calcular el coeficiente de similitud del coseno entre dos documentos (d1 y d2) es aquella que permite relacionar los vectores asociados a los documentos. El numerador no deja de ser un producto escalar entre los pesos P de los documentos; y el denominador la raíz cuadrada del producto del sumatorio de los pesos P de los documentos al cuadrado, donde n es la cantidad de términos del documento. La formulación del denominador con raíz cuadrada y cálculo de cuadrados, se diseña para conseguir un resultado final de la división, inferior a 1, de tal manera que el coeficiente sea de fácil manejo y lectura.

A partir de lo anterior es posible medir cuál es la desviación o semejanza entre dos documentos, por el número de grados del ángulo que forman. Esto es posible porque crean una estructura triangular a la que se aplica el cálculo del ángulo que forma la hipotenusa (en este caso el vector del documento d1) y el adyacente (el vector del documento d2) que resulta ser el coseno del triángulo. Esto se demuestra a continuación en la figura 2.14.

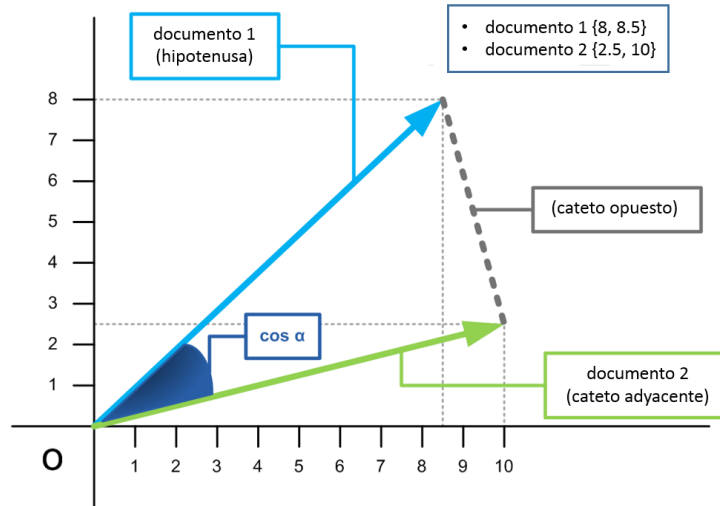


Figura 2.14. Cálculo del coseno

Se comprueba visualmente que existe cierta distancia entre el vector del documento d_1 y el vector del documento d_2 ; cuando ambos vectores se muestran tan próximos como para superponerse, implicará que el ángulo que forman será menor y que su nivel de coincidencia será superior. Por tanto, un coseno de 0° implicaría una similitud máxima.

De esta forma la función de similitud del coseno establece una medida de semejanza entre un documento consulta y la colección de entrada, posibilitando en grados matemáticos comparar un documento con el resto de la colección. El orden matemático de los coeficientes de similitud que se obtienen de la aplicación de la función coseno sobre toda la colección, permite establecer criterios de indexación dentro del dominio. Por tanto el valor numérico entre el coseno del documento consulta y un documento de la colección permite organizarlos en términos de su semejanza y establecer un orden de relevancia.

Conclusiones del Capítulo

El presente capítulo se inició con el enunciado de la propuesta de solución y la presentación de un esquema simplificado que aclara el proceso de representación de la información de acuerdo a las especificaciones analizadas. A partir de esto se describieron los principales artefactos relacionados con el análisis, diseño e implementación de la propuesta de solución. Para el cumplimiento de este objetivo se elaboraron los instrumentos correspondientes a la etapa de planeación de la metodología XP, tales como: historias de usuario, tiempo de ejecución, plan de iteraciones y el plan de entrega al cliente. En la segunda sección se especificó la arquitectura de software, el modelado UML, patrones de diseño y tarjetas clase-responsabilidad-creador. Finalmente se expuso el algoritmo

implementado así como las estructuras de datos utilizadas para el apoyo a la representación de la información.

En la etapa de planeación se describieron las historias de usuario: HU-2.Estandarizar colección de documentos, HU-5.Construir vector de términos y HU-8.Construir matriz de semejanza. Las mismas contienen una descripción de los requerimientos funcionales con los que debe contar el componente. Mediante la confección de las historias de usuarios se calculó y planificó el tiempo de ejecución en semanas de trabajo y se confeccionó el plan de entrega por iteraciones, quedando establecido un total de 24 semanas de trabajo con cuatro iteraciones.

Para el diseño de la propuesta de solución se empleó una arquitectura N capas, la misma quedó estructurada en tres capas: Entrada, Procesamiento y Representación, definiéndose los roles y responsabilidades de cada una. Se confeccionó además el diagrama de dominio, diagrama de clases y se describieron los patrones de diseño: Experto, Creador, Alta Cohesión, Bajo Acoplamiento, Singleton, Single Threaded Execution y Scheduler, así como las tarjetas clases-responsabilidad-creador.

En la implementación de la propuesta de solución se presentó el algoritmo a partir de su pseudocódigo y se describieron las principales características del mismo. Se describió y analizó el uso de la concurrencia en dicho algoritmo, así como las ventajas de su aplicación a través de una gráfica de comparación de la ejecución del algoritmo de forma secuencial y concurrente. Finalmente se presentaron y explicaron las estructuras diccionario de datos y matriz de semejanza, utilizadas en contribución a la representación de la información. Por tanto en cumplimiento del objetivo específico afín se implementó un componente informático para la representación del corpus textual de archivos digitales textuales.

CAPÍTULO 3. COMPROBACIÓN DEL FUNCIONAMIENTO DE LA PROPUESTA DE SOLUCIÓN

Introducción

El desarrollo de sistemas de software implica una serie de actividades de producción en las cuales es posible la aparición de errores. La imposibilidad humana de trabajar de forma perfecta hace necesario acompañar el desarrollo de software de una actividad que garantice su calidad (55). La realización de pruebas es la actividad encargada de comprobar el funcionamiento del software para garantizar el correcto funcionamiento del producto final. El presente capítulo está enfocado a la verificación del funcionamiento de la solución implementada. Para ello se define la estrategia de prueba, las pruebas de unidad y de aceptación además de ilustrar su aplicación al componente implementado para la comprobación del correcto funcionamiento del mismo.

3.1 Comprobación del funcionamiento del algoritmo implementado

El proceso de *stemming* en el algoritmo implementado constituye la base en la representación del corpus textual de una colección de archivos digitales de entrada. La correcta derivación de los términos de los archivos permite tener índices de menor tamaño, reduciendo el costo computacional de la búsqueda sobre la colección de entrada. Por tal motivo es necesario comprobar si la derivación en términos realizada por el componente implementado se corresponde con la raíz semántica de cada vocablo de la colección de archivos en cuestión. Para ello es necesario establecer un vocabulario de prueba en el cual se conozca la raíz semántica de cada una de sus palabras, a partir de esto es posible comparar si el resultado emitido por el algoritmo se corresponde con el resultado esperado.

El listado de lexemas utilizado se encuentra en el diccionario de lexicografía práctica del Ministerio de Educación, Cultura y Deporte del gobierno de España (56). La colección asciende a un total de 28 377 palabras con sus correspondientes lexemas. Con el objetivo de sintetizar en este documento la representación completa de los 28 377 vocablos involucrados en el proceso de prueba, solo se ejemplifican 100. La tabla 3.1 muestra la colección de prueba formada por la dupla palabra-lexema.

Tabla 3.1. Colección de prueba

Palabra	Lexema	Palabra	Lexema	Palabra	Lexema
aarón	aaron	baccantes	baccant	charchina	charchin
abaco	abac	bacco	bacc	charchino	charchin
abajo	abaj	batcheando	bach	charco	charc
abandera	abander	bachilleres	bachiller	charla	charl
abandona	abandon	bacteria	bacteri	charlaron	charl
abandonada	abandon	badmington	badmington	charlas	charl
abandonadas	abandon	bae	bae	charles	charl
abandonado	abandon	báez	baez	charlie	charli
abandonados	abandon	bafsa	bafs	charló	charl

abandonamos	abandon	bagaje	bagaj	charly	charly
abandonan	abandon	bagdad	bagd	charnela	chamel
abandonar	abandon	bahamas	baham	charnell	charnell
abandonarlo	abandon	bahía	bah	charreada	charr
abandonaron	abandon	baila	bail	charreamos	charr
abandono	abandon	bailable	bailabl	charreando	charr
abandonó	abandon	bailando	bail	charro	charr
abaratar	abarat	bailar	bail	charros	charr
abarca	abarc	bailarines	bailarin	chase	chas
abarcamos	abarc	bailarlos	bail	chasis	chasis
abarcan	abarc	baile	bail	chararra	chatarr
abarcar	abarc	bailes	bail	chatarrero	chatarrer
abarcará	abarc	bailleres	bailler	chato	chat
abarcarán	abarc	baja	baj	chaumu	chaumu
abarcaría	abarc	bajacaliforniano	bajacalifornian	chauvet	chauvet
abarcó	abarc	bajadas	baj	chavarría	chavarr
abaroa	abaro	bajado	baj	chavas	chav
abarroteros	abarroter	bajan	baj	chavero	chaver
abarrotó	abarrot	bajando	baj	chavez	chavez
abastece	abastec	bajar	baj	chávez	chavez
abastecedora	abastecedor	bajará	baj	chavitas	chavit
abastecer	abastec	bajarle	baj	chavo	chav
abastecimiento	abastec	bajaron	baj	chavos	chav
abastecimientos	abastec	bajarse	baj		
abasto	abast	bajas	baj		

El resultado obtenido luego de ejecutar el *stemming* en el software desarrollado para la colección de prueba se muestra en la figura 3.1. Como se puede observar, tal resultado es producto de la variación del algoritmo de espacio vectorial implementado con la intención de disminuir el espacio de dimensionalidad de los vectores.

```

-----Opciones de la Matriz-----
Presione (1) Mostrar Matriz
Presione (2) Para ir Atrás

[<aaron, 1> <abac, 1> <abaj, 1> <abander, 1> <abandon, 12> <abarat, 1> <abarc, 8> <abaro, 1>
<abarroter, 1> <abarrot, 1> <abastec, 4> <abastecedor, 1> <abast, 1> <baccant, 1> <bacc, 1> <
bach, 1> <bachiller, 1> <bacteri, 1> <badmington, 1> <bae, 1> <baez, 1> <bafs, 1> <bagaj, 1>
<bagd, 1> <baham, 1> <bah, 1> <bail, 6> <bailabl, 1> <bailarin, 1> <bailler, 1> <baj, 11> <ba
jacalifornian, 1> <charchin, 2> <charc, 1> <charl, 5> <charli, 1> <charly, 1> <charnel, 1> <c
harnell, 1> <charr, 5> <chas, 1> <chasis, 1> <chatarr, 1> <chatarrer, 1> <chat, 1> <chaumu, 1
> <chauvet, 1> <chavarr, 1> <chav, 3> <chaver, 1> <chavez, 2> <chavit, 1> ]
    
```

Figura 3.1. Salida del algoritmo

Como se observa en la figura 3.1 cada término contiene el lexema de la palabra con la frecuencia de aparición del mismo en el texto analizado. Por tanto, si se comparan las palabras y lexemas de la tabla 3.1 con los términos de la figura 3.1 se obtienen los siguientes resultados.

Tabla 3.2. Comparación de resultados

Colección de prueba	No.	Resultado del Stemming	No.	Colección de prueba	No.	Resultado del Stemming	No.
aaron	1	aaron	1	bail	6	bail	6
abac	1	abac	1	bailabl	1	bailabl	1
abaj	1	abaj	1	bailarin	1	bailarin	1
abander	1	abander	1	bailler	1	bailler	1
abandon	12	abandon	12	baj	11	baj	11
abarat	1	abarat	1	bajacalifornian	1	bajacalifornian	1
abarc	8	abarc	8	charchin	2	charchin	2
abaro	1	abaro	1	charc	1	charc	1
abarroter	1	abarroter	1	charl	5	charl	5
abarrot	1	abarrot	1	charli	1	charli	1
abastec	4	abastec	4	charly	1	charly	1
abastecedor	1	abastecedor	1	charnel	1	charnel	1
abast	1	abast	1	charnell	1	charnell	1
baccant	1	baccant	1	charr	5	charr	5
bacc	1	bacc	1	chas	1	chas	1
bach	1	bach	1	chasis	1	chasis	1
bachiller	1	bachiller	1	chatarr	1	chatarr	1
bacteri	1	bacteri	1	chatarrer	1	chatarrer	1
badmington	1	badmington	1	chat	1	chat	1
bae	1	bae	1	chaumu	1	chaumu	1
baez	1	baez	1	chauvet	1	chauvet	1
bafs	1	bafs	1	chavarr	1	chavarr	1
bagaj	1	bagaj	1	chav	3	chav	3
bagd	1	bagd	1	chaver	1	chaver	1
baham	1	baham	1	chavez	2	chavez	2
bah	1	bah	1	chavit	1	chavit	1

Como se muestra en la tabla 3.2 los resultados obtenidos coinciden con los datos de la colección de prueba, con una efectividad del 100 %. La misma prueba realizada mediante un componente implementado por el equipo de desarrollo con este propósito arroja los siguientes resultados:

- ✓ Para la colección de 28377 palabras se obtienen 28192 coincidencias con 185 errores, para una tasa de efectividad del 99.34 %.

La figura 3.2 muestra la salida del software para la colección de prueba. El listado con los lexemas que produjeron errores se muestran en el anexo número 3.

```
run:
Total de términos procesados:28377
Total de coincidencias entre lexemas:28192
Total de errores detectados:185

BUILD SUCCESSFUL (total time: 0 seconds)
```

Figura 3.2. Resultado de prueba

El origen de los errores detectados durante las pruebas realizadas al algoritmo se debe a que no existe una correspondencia entre las derivaciones producidas por el algoritmo para una familia de palabras y las derivaciones registradas en el diccionario de prueba para la misma familia. El algoritmo es capaz de derivar cada palabra de la colección y esta es la misma para cada ocurrencia de la palabra. Por tanto en términos de representación del corpus textual de los documentos el algoritmo implementado funciona correctamente.

3.2 Comprobación del funcionamiento del software desarrollado

Uno de los pilares de XP es el proceso de pruebas (57). Esto permite aumentar la calidad de los sistemas reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección. También permite aumentar la seguridad de evitar efectos colaterales no deseados al realizar modificaciones y refactorizaciones.

La metodología XP divide las pruebas del sistema en dos grupos: pruebas unitarias, encargadas de verificar el código y diseñadas por los programadores y pruebas de aceptación destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida por el cliente (57). Para comprender como realizar estas pruebas es necesario definir en qué consisten las pruebas del sistema. Las pruebas del sistema tienen como objetivo verificar las funcionalidades comprobando que su resultado esté en función de los requisitos del sistema (58); en este caso historias de usuario. Generalmente estas pruebas son desarrolladas por los programadores para verificar que su solución se comporta de la manera esperada, por lo que se podrían acoplar dentro de la definición de pruebas unitarias que propone XP. Sin embargo, las pruebas tienen como objetivo verificar que el sistema cumple los requisitos establecidos por el usuario por lo que también pueden conectar dentro de la categoría de pruebas de aceptación. Ambas pruebas se ejecutarán mediante la definición de una estrategia de prueba que especificará los niveles y métodos de prueba necesarios para la aplicación de las mismas.

Estrategia de Prueba.

La estrategia de prueba describe el enfoque y los objetivos generales de las actividades de prueba. Incluye además los niveles de prueba a ser diseccionados y el tipo de prueba a ser ejecutada (59). Por tanto la estrategia de prueba permite definir las técnicas y criterios a tener en cuenta para realizar las pruebas al componente desarrollado.

La estrategia de prueba define:

- ✓ Técnicas de pruebas (manual o automática) y herramientas a ser usadas.
- ✓ Qué criterios de éxitos y culminación de la prueba serán usados.

- ✓ Consideraciones especiales afectadas por requerimientos de recursos o que tengan implicaciones en la planificación.

Los niveles de pruebas que se distinguen son:

- ✓ Prueba de unidad.
- ✓ Prueba de aceptación.

Ambos niveles son los definidos por la metodología de desarrollo seleccionada para la realización de las pruebas.

Los métodos de pruebas definidos son:

- ✓ Prueba de caja negra.
- ✓ Prueba de caja blanca.

La prueba de caja negra se refiere a las pruebas que se llevan a cabo sobre la interfaz del software. O sea, los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene. Por otro lado la prueba de la caja blanca comprueba los caminos lógicos del software proponiendo casos de prueba que se ejerciten con conjuntos específicos de condiciones y/o bucles. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado o mencionado.

La estrategia seguida para la realización de las pruebas al componente desarrollado contempla dos niveles: el nivel de pruebas de unidad y el nivel de prueba de aceptación. En el primer nivel, se realizan pruebas automáticas haciendo uso de tecnologías que permiten la comprobación del código. En el segundo incluye la técnica manual mediante las pruebas de aceptación con el diseño de casos de prueba utilizando el método de caja negra. Las secciones Pruebas unitarias y Pruebas de aceptación describen detalladamente las pruebas realizadas según la estrategia planteada así como la descripción de la herramienta utilizada.

Pruebas unitarias

Una prueba unitaria es un método que prueba una unidad estructural de código, por lo general son simples y rápidas de codificar. Las pruebas unitarias son una forma para verificar la estructura y el buen funcionamiento de una parte del código, están dirigidas a probar clases aisladamente y relacionadas con el código y la responsabilidad de cada clase, así como sus fragmentos de código

más críticos (60). Las mismas son diseñadas y realizadas por los programadores para verificar constantemente las funcionalidades implementadas. Para el desarrollo de la solución estas pruebas se realizan de forma automática utilizando el framework de pruebas JUnit 4.12.

JUnit 4.12

Es un framework para automatizar las pruebas unitarias de aplicaciones Java en la fase de desarrollo. Este permite controlar la ejecución de las clases y de esta forma evaluar si el funcionamiento de cada uno de los métodos se comporta como se espera (61). Para realizar esta evaluación se compara el resultado obtenido durante la ejecución del método con el resultado esperado, partiendo de los datos de entrada necesarios para realizar la prueba. Si ambos resultados son iguales JUnit retorna que el método pasó la prueba correctamente, de lo contrario devuelve un fallo en la misma.

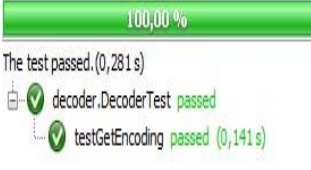
Las funcionalidades con que cuenta el sistema fueron comprobadas independientemente una vez implementadas. A continuación se muestran mediante iteraciones las pruebas realizadas a las principales funcionalidades del sistema, las restantes se encuentran en el anexo número 4.

Pruebas unitarias. Iteración 1

La primera iteración de pruebas unitarias fue realizada sobre las historias de usuario siguientes: HU-1 Recibir colección de entrada y HU-3 Filtrar colección de entrada.


Las tablas 3.3 y 3.4 muestran el resultado de la ejecución de las baterías de prueba realizadas a las clases Decoder y SpanishCharacterFilter, generadas para la resolución de las historias de usuario HU-1 Recibir colección de entrada y HU-3 Filtrar colección de entrada respectivamente. En ellas se muestran el método u operación a probar, la entrada proporcionada, los resultados esperados y obtenidos, si funciona correctamente y en el campo observaciones una captura del resultado mostrado por el framework de prueba empleado para la ejecución de la prueba.

Tabla 3.3. HU-1 Recibir colección de entrada. Clase Decoder

Pruebas unitarias					
Método a probar	Entrada	Resultado esperado	Resultado obtenido	Funciona correctamente	Observaciones
getEncoding()	El método recibe como entrada un documento en texto plano y codificación UTF-8.	El método debe retornar la codificación del documento recibido como entrada. En	Documento codificado a UTF-8.	Si	 <p>The test passed. (0,281 s) decoder.DecoderTest passed testGetEncoding passed (0,141 s)</p>

		este caso UTF-8.			
--	--	------------------	--	--	--

Tabla 3.4. HU-3 Filtrar colección de entrada. Clase SpanishCharacterFilter

Pruebas unitarias					
Método a probar	Entrada	Resultado esperado	Resultado obtenido	Funciona correctamente	Observaciones
spanishFilter()	Recibe como entrada una línea de texto del documento que se esté analizando.	Debe eliminar las direcciones de correo electrónico, las direcciones URL, las letras acentuadas así como la letra ü y la letra ñ. Sustituyéndolas por espacios en blanco y devolver la cadena de texto modificada.	Se obtuvo la línea de texto parcialmente modificada. La dirección URL presente en la cadena de texto se sustituyó.	No	

Durante la primera iteración de pruebas realizada se detectaron errores de codificación en el método spanishFilter de la clase SpanishCharacterFilter al filtrar las direcciones URL en una cadena de texto. Los mismos fueron solucionados por el equipo de desarrollo permitiendo dar paso a la próxima iteración.

Pruebas unitarias. Iteración 2

La segunda iteración de pruebas unitarias se realizó sobre las historias de usuario siguientes: HU-4 Realizar *stemming* a la colección de entrada, HU-5 Construir vector de términos, HU-6 Calcular peso asociado a cada término obtenido.

Las tablas 3.5 y 3.6 muestran el resultado de la ejecución de las baterías de prueba realizadas a las clases Filter y Vector, generadas para la resolución de las historias de usuario HU-4 Realizar *stemming* a la colección de entrada y HU-5 Construir vector de términos respectivamente. En ellas se muestran los mismos campos descritos anteriormente.

Tabla 3.5. HU-4 Realizar stemming a la colección de entrada. Clase Filter

Pruebas unitarias


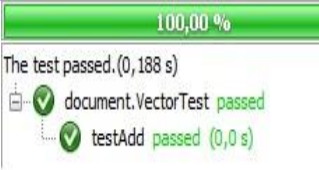
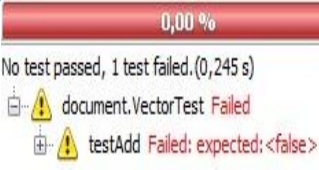
Método a probar	Entrada	Resultado esperado	Resultado obtenido	Funciona correctamente	Observaciones
tokenStream()	El método recibe como entrada una cadena de texto para conformar los tokens.	El método debe retornar los tokens conformados con las raíces semánticas de las palabras existentes en la cadena de texto entrada.	Se obtuvo cada uno de los <i>tokens</i> esperados con la raíz semántica de cada palabra de la cadena de texto entrada.	Si	 <p>100,00 % The test passed.(0,447 s) process.FilterTest passed testTokenStream passed (0,279 s)</p>

Tabla 3.6. HU-5 Construir vector de términos. Clase Vector

Pruebas unitarias					
Método a probar	Entrada	Resultado esperado	Resultado obtenido	Funciona correctamente	Observaciones
add()	Recibe la raíz semántica de una palabra.	Debe añadir un nuevo término al vector y retornar el valor booleano true.	El término fue añadido correctamente al vector y se obtuvo el valor booleano true.	Si	 <p>100,00 % The test passed.(0,188 s) document.VectorTest passed testAdd passed (0,0 s)</p>
	Recibe la raíz semántica de una palabra.	Debe actualizar la frecuencia del término compuesto por dicha raíz en el vector y retornar el valor booleano false.	La frecuencia del término no fue actualizada. En su lugar el mismo fue adicionado nuevamente y se obtuvo el valor booleano true.	No	 <p>0,00 % No test passed, 1 test failed.(0,245 s) document.VectorTest Failed testAdd Failed: expected:<false></p>

En la segunda iteración se detectaron errores de codificación en el método add de la clase Vector al verificar la existencia del término que pretende adicionar al vector, evitando de esta forma valores duplicados. En este caso el término fue añadido al vector, en lugar de actualizar su frecuencia como se esperaba. Se dio solución a cada uno de los errores encontrados garantizando el correcto funcionamiento del método y se pasó a la siguiente iteración.

Pruebas unitarias. Iteración 3

La tercera iteración de pruebas unitarias se realiza sobre las historias de usuario siguientes: HU-7 Construir diccionario de términos y HU-10 Almacenar diccionario de términos.

Las tablas 3.7 y 3.8 muestran el resultado de la ejecución de las baterías de prueba realizadas a las clases Dictionary, Chapter y ReadWriteFile. Las dos primeras fueron generadas para la resolución de la historia de usuario HU-7 Construir diccionario de términos y la tercera para solucionar la historia de usuario HU-10 Almacenar diccionario de términos. Las mismas mantienen los mismos campos que las tablas anteriores.

Tabla 3.7. HU-7 Construir diccionario de términos. Clase Dictionary

Pruebas Unitarias					
Método a probar	Entrada	Resultado esperado	Resultado obtenido	Funciona correctamente	Observaciones
add	Se recibe un vector de términos.	Cada término del vector debe ser añadido al diccionario o actualizada su frecuencia de aparición.	Los términos del vector fueron correctamente añadidos o actualizados en el diccionario.	Si	

Tabla 3.8. HU-7 Construir diccionario de términos. Clase Chapter

Pruebas Unitarias					
Método a probar	Entrada	Resultado esperado	Resultado obtenido	Funciona correctamente	Observaciones
addTerm()	Recibe como datos la raíz de una palabra y su frecuencia de aparición en el documento al que pertenece.	Debe adicionar un nuevo término compuesto por la raíz de palabra recibida al capítulo correspondiente y retornar el valor booleano true.	El término fue añadido correctamente al capítulo, retornando el valor booleano true.	Si	



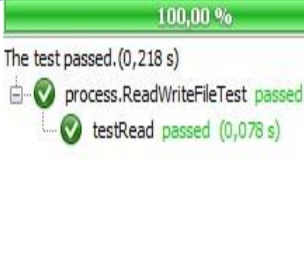
	Recibe como datos la raíz de una palabra y su frecuencia de aparición en el documento al que pertenece.	Debe actualizar la frecuencia de aparición del término formado por la raíz de la palabra recibida en el capítulo y retornar el valor booleano false.	La frecuencia del término no fue actualizada. El término se añadió al capítulo y se obtuvo el valor booleano true.	Si	 <p>100,00 % The test passed.(0,14 s) document.ChapterTest passed test1AddTerm passed (0,002 s)</p>
--	---	--	--	----	--

Tabla 3.9. HU-10 Almacenar diccionario de términos. Clase ReadWriteFile

Pruebas Unitarias					
Método a probar	Entrada	Resultado esperado	Resultado obtenido	Funciona correctamente	Observaciones
Write()	Recibe como datos el objeto a guardar (para este caso es el objeto diccionario) y la dirección del lugar donde se va a almacenar.	Debe crear un fichero en la dirección indicada que contenga el objeto de entrada.	El fichero fue creado correctamente en la dirección indicada.	Si	 <p>100,00 % The test passed.(0,25 s) process.ReadWriteFileTest passed testWrite passed (0,094 s)</p>
Read()	Recibe la dirección completa (incluyendo el nombre) del lugar donde se encuentra el archivo a leer.	Debe retornar el objeto contenido en el fichero leído.	El objeto contenido en el fichero leído fue devuelto de forma correcta.	Si	 <p>100,00 % The test passed.(0,218 s) process.ReadWriteFileTest passed testRead passed (0,078 s)</p>

Durante la tercera iteración no se detectaron errores en la codificación de los métodos probados, obteniéndose el resultado esperado en cada una de las pruebas realizadas. De esta forma concluye el proceso de pruebas unitarias realizado a las funcionalidades seleccionadas.

Como se observa en la figura 3.3 durante el proceso de pruebas unitarias se verificaron un total de 10 operaciones del componente implementado. Durante la primera iteración se encontraron errores en una de las dos operaciones comprobadas y en la segunda de tres operaciones solo una presentó problemas. En la tercera iteración se comprobaron cinco operaciones y no se obtuvieron errores en

ninguna de ellas, por tanto solo se necesitó corregir errores en dos de las operaciones involucradas en el proceso de pruebas.

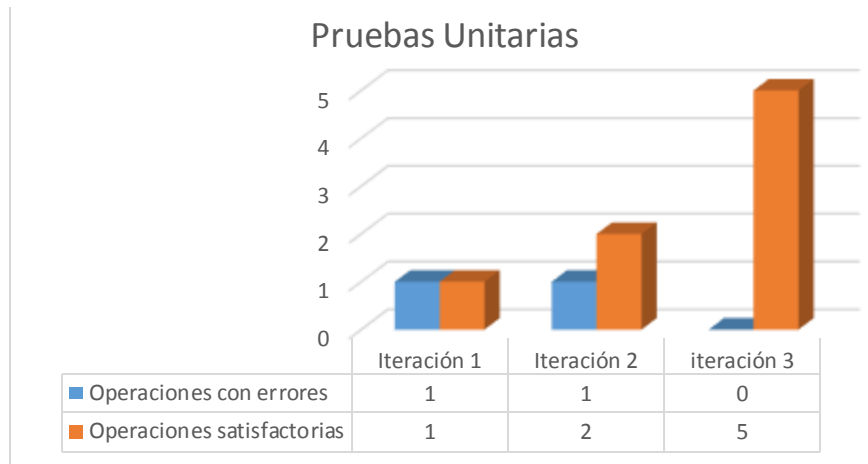


Figura 3.3. Pruebas unitarias

Pruebas de aceptación

Las pruebas de aceptación son una parte integral del desarrollo incremental definido por XP. Todas las historias de usuario son apoyadas por pruebas de aceptación, que se definen por el propio cliente. Las pruebas de aceptación son encaminadas a la satisfacción del cliente lo que permite el fin de una iteración y comienzo de otra (62). Las mismas obligan al cliente a un profundo conocimiento del estado de la aplicación y lo que debe hacer en circunstancias específicas (57).

La realización de las pruebas de aceptación en cada iteración del proceso de desarrollo de software está asociada a cada historia de usuario. Para ello se realiza un modelo que involucra: acción a probar, datos de prueba, resultado esperado, resultado obtenido y evaluación de la prueba.

Pruebas de aceptación. Iteración 1

La tabla 3.10 muestra los datos obtenidos a partir de la aplicación de las pruebas de aceptación a las historias de usuario HU-2, HU-3 y HU-4 (por su número de historia).

Tabla 3.10. Pruebas de aceptación. Iteración 1

Pruebas de Aceptación				
Acción a probar	Datos de prueba	Resultado esperado	Resultado obtenido	Evaluación de la prueba
HU-1: el componente debe ser capaz de decodificar la colección de	La prueba da inicio con una colección de 10 000 documentos en distintos formatos.	Todos los documentos deben ser convertidos a formato de texto	El componente procesó correctamente los datos de prueba y generó una lista de	Aceptada

documentos de entrada.		plano con codificación UTF-8.	errores con aquellos que no pudo convertir. El resultado obtenido muestra una precisión del 95 %.	
HU-3: el proceso de filtrado de los textos está compuesto por dos filtros fundamentales: LowerCaseFilter y StopFilter.	La prueba da inicio con 1000 documentos	Para el LowerCaseFilter el componente debe emitir como salida todos los textos en letra minúscula. El StopFilter debe eliminar de todos los textos un listado de palabras predefinidas por el cliente.	Para el LowerCaseFilter el componente emite la salida esperada. En el StopFilter se decide añadir nuevas palabras y expresiones a reconocer.	Pedido de cambio: Se necesita que se eliminen números, enlaces a url y direcciones a correos electrónicos.
HU-4: los textos filtrados deben pasar por un proceso de lematización que reduzca todas las palabras a su raíz semántica	La prueba da inicio con 1000 documentos	El lematizador debe reducir todas las palabras del contenido de los documentos a su raíz semántica	El lematizador reduce todas las palabras del dominio a la raíz semántica.	Aceptada

Para la primera iteración se solicita un pedido de cambio en el proceso de StopFilter. Este consiste en incluir que de los documentos filtrados se eliminen números, enlaces url y direcciones de correos electrónicos. Los cambios son aceptados y realizados por el equipo de desarrollo lo que permite dar paso a la siguiente iteración.

Pruebas de aceptación. Iteración 2

La tabla 3.11 muestra los datos obtenidos a partir de la aplicación de las pruebas de aceptación a las historias de usuario HU-5, HU-7 y HU-8 (por su número de historia).

Tabla 3.11. Pruebas de aceptación. Iteración 2

Pruebas de Aceptación				
Acción a Probar	Datos de Prueba	Resultado Esperado	Resultado Observado	Pasa la prueba
HU-5: cada documento de la colección debe ser presentado en un vector de términos donde cada componente del vector constituye el	La prueba da inicio con 10 000 documentos	El motor de representación debe presentar toda la colección en términos del vector en un tiempo prudente.	El tiempo de representación demoró un total 2 minutos.	No conformidad: Tiempo de ejecución demasiado alto.

lema de la palabra correspondiente en la colección.				El vector solo muestra los términos, se necesita manejar más información
HU-7: el motor de representación debe construir un diccionario de términos con el vocabulario de la colección	La prueba da inicio con 10 000 documentos	El diccionario debe construirse a partir de la colección de entrada y debe tenerse en cuenta la disminución del tiempo de ejecución del proceso.	El tiempo de representación demoró un total 3 minutos.	No conformidad: Tiempo de ejecución demasiado alto.
HU-8: el componente debe construir la matriz de términos a partir de los vectores de términos y calcular los pesos asociados a cada término del vector.	La prueba da inicio con 10 000 documentos	Los pesos asociados a cada término deben estar debidamente calculados e incluir solamente hasta 5 cifras significativas después de la coma.	El tiempo de representación demoró un total 4 minutos.	No conformidad: Tiempo de ejecución demasiado alto. Pedido de cambio: Matriz de semejanza.
HU-8: el componente debe realizar la representación textual de los documentos en un tiempo prudencial, se considera un máximo de 20 segundos por cada 1000 documentos de tamaño medio (2000 palabras) en la colección.	La prueba da inicio con 10000 documentos con dimensiones de entre 1000-90 000 palabras, para un total de 8 000 000 de palabras.	El tiempo estimado de respuesta debe ser como máximo 5 minutos.	El componente demoró en procesar los documentos un total de 9 minutos.	No conformidad: Tiempo de ejecución demasiado alto.

En la segunda iteración se detectaron cuatro no conformidades asociadas a las historias de usuarios: HU-5, HU-7, y HU-8 respectivamente, además de un pedido de cambio en la HU-6. Como se observa en cada historia de usuario prevalece la no conformidad en el tiempo de ejecución. Para esta iteración se solucionan las no conformidades mediante la implementación del diccionario de datos indexado y la aplicación de patrones de concurrencia. Se adiciona el pedido de cambio, generando además de la representación en términos de cada documento, una matriz de semejanza resultante de la comparación de cada documento representado con la función del coseno.

Pruebas de aceptación. Iteración 3

La tabla 3.12 muestra los datos obtenidos a partir de la aplicación de las pruebas de aceptación a las historias de usuario HU-10 y HU-11 (por su número de historia).

Tabla 3.12. Pruebas de aceptación. Iteración 3

Pruebas de Aceptación				
Acción a Probar	Datos de Prueba	Resultado Esperado	Resultado Observado	Pasa la prueba
HU-10, HU-11: el componente debe almacenar los resultados obtenidos para su posterior uso.	Diccionario de datos, Matriz de semejanza	El componente almacena el Diccionario de datos y la matriz de semejanza en un archivo binario.	El componente almacena el Diccionario de datos y la matriz de semejanza en un fichero binario.	Aceptada

La tercera iteración pasa satisfactoriamente la prueba de aceptación y culmina el proceso de desarrollo del software. Como se muestra en la figura 3.4 el proceso de pruebas de aceptación se realizó en tres iteraciones. Durante la primera iteración se obtuvo solamente un pedido de cambio sobre la solución desarrollada, el cual fue desarrollado satisfactoriamente. En la segunda iteración se detectaron cuatro no conformidades y un pedido de cambio que fueron corregidas e incluido de forma correcta. Luego se realizó la tercera iteración de forma satisfactoria. En total se obtuvieron cuatro no conformidades y dos pedidos de cambio durante el proceso de pruebas.

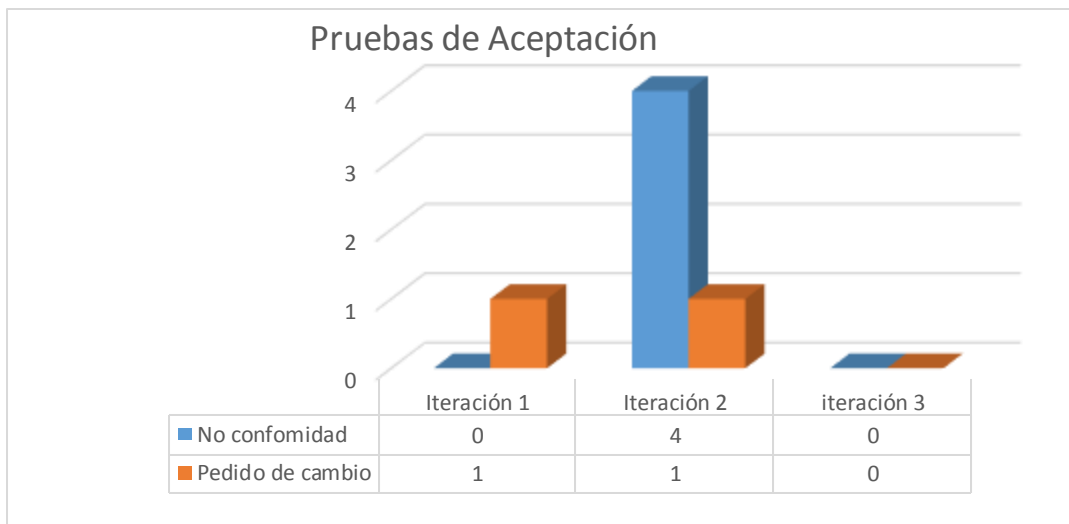


Figura 3.4. Pruebas de aceptación

Conclusiones del Capítulo

En este capítulo se realizó la comprobación del algoritmo implementado mediante la aplicación del proceso de *stemming* a una colección de 28 377 palabras, comparando las derivaciones obtenidas con los lexemas correspondientes a dichas palabras, extraídos del diccionario de lexicografía práctica del Ministerio de Educación, Cultura y Deporte del Gobierno de España. La efectividad obtenida para una muestra de 100 palabras fue de un 100% y para la colección completa se obtuvo una efectividad del 99.34 %. Luego se definió la estrategia de prueba a seguir para la comprobación del funcionamiento del sistema, en la cual se decidió la aplicación de pruebas unitarias y de aceptación. Para la realización de las pruebas unitarias se empleó el framework JUnit 4.12 que permitió automatizar dicho proceso, lo que posibilitó comprobar todas las funcionalidades del sistema y comprobar el correcto funcionamiento de las mismas.

El proceso de prueba a las funcionalidades involucradas se dividió en tres iteraciones, describiendo cada una de las baterías de pruebas realizadas en las tablas elaboradas. Durante la primera iteración se detectaron varios errores en la operación `spanishFilter` de la clase `SpanishCharacterFilter`, cada uno de ellos fue resuelto en tiempo dando paso a la próxima iteración de pruebas. En la segunda iteración se detectaron errores en la operación `add` de la clase `Vector` y los mismos fueron corregidos por el equipo de desarrollo de forma correcta. Luego se dio paso a la última iteración, la misma se realizó satisfactoriamente culminando así el proceso de pruebas unitarias a las funcionalidades seleccionadas. Se comprobó de esta forma que cada una de ellas devuelve el resultado esperado acorde a los datos de entrada empleados.

Para la realización de las pruebas de aceptación se dividió el período en tres iteraciones. Se elaboró una tabla por cada iteración donde se describieron las pruebas realizadas, mostrando la acción a probar, los datos de prueba, el resultado esperado y observado así como la evaluación de la prueba. En la primera iteración se solicitó y aceptó un pedido de cambio que fue resuelto por el equipo de proyecto. Luego se dio paso a la segunda iteración en las cuáles se detectaron 4 no conformidades y un pedido de cambio, resolviéndose las no conformidades en su totalidad y adicionando el pedido de cambio solicitado. La tercera y última iteración se realizó de forma satisfactoria y se culminó el período de pruebas de aceptación, mostrando los resultados de dicho período en un gráfico de barras. A través de las pruebas descritas anteriormente se comprobó el funcionamiento del algoritmo y el componente desarrollado, corrigiéndose los errores encontrados durante el proceso de desarrollo.

CONCLUSIONES

El presente documento describe el proceso de investigación realizado para dar solución al problema de representar el corpus textual de los archivos digitales textuales en contribución a la recuperación de información. Para ello se identificaron y definieron los conceptos fundamentales que permitieron contextualizar la representación del corpus textual de archivos digitales. Se mostraron los diferentes modelos y técnicas de recuperación de información concluyendo en la selección del modelo espacio vectorial como técnica para la representación del corpus textual de archivos digitales textuales.

A partir de lo anterior se realizó el análisis de las principales soluciones informáticas existentes que utilizan modelos de recuperación de información y se basan en la representación del corpus textual de archivos digitales. Esto permitió determinar que ninguno de los sistemas estudiados constituía una solución al problema planteado puesto que en su mayoría son soluciones informáticas de plataforma Windows, por lo que se imposibilita su integración con varios tipos de software y limita su capacidad multiplataforma. Otros solo basan su análisis sobre información estructurada o bien necesitan un proceso de entrenamiento que determine las clases en las que se dividen los archivos. Además, solo analizan las ideas, frases y palabras principales del texto en cuestión o bien no permiten comparativa de archivos u obtención del corpus representado, pues su objetivo principal es la localización y definición de las palabras clave. Otras soluciones realizan análisis de datos textuales sobre un único archivo de texto y todos los textos deben pertenecer a un mismo tema, por lo que no permiten representar diversos contenidos simultáneamente además de no analizar información no estructurada.

Para preparar el ambiente de desarrollo se realizó el estudio de las metodologías de desarrollo de software y se seleccionó la metodología XP para guiar el desarrollo de la solución. En apoyo a la metodología se seleccionó el lenguaje de modelado UML y Visual Paradigm 8.0 para el modelado de artefactos de la propuesta de solución. También se caracterizó el lenguaje de programación Java 8 y entorno de desarrollo NetBeans 8.0 utilizados para la codificación e implementación de la solución. Todo ello da cumplimiento a la fundamentación teórica de la investigación a partir del análisis de los antecedentes con respecto a la recuperación de información y representación del corpus textual de archivos digitales textuales.

Se expuso además la propuesta de solución y a partir de ello se ilustraron los elementos fundamentales de la misma. Fueron diseñadas un total de 11 historias de usuario, planificadas en 4 iteraciones además de crearse el plan de entregas de las funcionalidades. Se diseñó y mostró la arquitectura de software para el desarrollo de la solución escogiendo para ello una arquitectura en

3 capas lógicas: Entrada, Procesamiento y Presentación. Fueron descritos los patrones de diseño utilizados tales como: Experto, Creador, Alta Cohesión, Bajo Acoplamiento, Singleton, Single Threaded Execution y Scheduler y se mostró un ejemplo de la utilización de cada uno. Se expuso el algoritmo implementado así como las estructuras de datos utilizadas para el apoyo a la representación de la información de los documentos digitales.

La codificación del componente se realizó bajo la concepción de la programación concurrente, lo que permitió una reducción del tiempo de corrida del algoritmo. Esto se debe a que se solaparon los tiempos de entrada/salida, así como el aprovechamiento del entorno multiprocesador realizándose la ejecución de los procesos simultáneamente. Lo anterior da cumplimiento al desarrollo de una solución informática para la representación del corpus textual de archivos digitales textuales en materia de planificación, diseño y codificación de software.

Fue comprobado el funcionamiento del algoritmo implementado realizando las pruebas necesarias que arrojaron una efectividad del 99.34% cumpliendo con el objetivo del diseño del mismo. Para verificar el correcto funcionamiento de la solución implementada se definió una estrategia de prueba, en la que se decidió la aplicación de pruebas unitarias y de aceptación. Tales pruebas fueron realizadas en 3 iteraciones y arrojaron 4 no conformidades y 2 pedidos de cambio que fueron resueltas y realizados respectivamente. Estas pruebas permitieron detectar y corregir errores de codificación en las funcionalidades durante la implementación, aportando un nivel de calidad aceptable al producto final. Por tanto se logró la comprobación del funcionamiento de la propuesta de solución.

Por lo expuesto anteriormente, se puede concluir que se desarrolló satisfactoriamente una solución informática que permite representar el corpus textual de una colección de archivos digitales textuales contribuyendo de este modo a la recuperación de información.

REFERENCIAS

1. **CASTELLS, M.** *La Era de la Información: Economía, Sociedad y Cultura: La sociedad Red.* Madrid : Alianza Editorial, 2005. pág. 47. ISBN 10370-10-5500-4.
2. **CHIAVENATO, I.** *Introducción a la Teoría General de la Administración.* Séptima. México : McGraw-Hill Interamericana, 2006. ISBN 10-970-10-5500-02.
3. **MARTÍN GAVILÁN, C.** *Concepto y función de archivo. Clases de archivos. El Sistema Archivístico Español.* Barcelona : s.n., 2009.
4. **FERNÁNDEZ GARCÍA, M. A.** Organización del archivo de una empresa. *mundoarchivístico.* [En línea] 2011. [Citado el: 8 de marzo de 2015.] <http://www.mundoarchivístico.com>.
5. **DUQUE, D.** Los Archivos Digitales. *Los archivos y algo más.* [En línea] 2009. [Citado el: 18 de marzo de 2015.] <http://doraduque.wordpress.com/tag/que-es-un-archivo-digital/>.
6. *Approaches to intelligent information retrieval.* **CROFT, W.B.** 4, Massachusetts : s.n., 1987, Information Processing and Management: an International Journal - Artificial Intelligence and Information Retrieval, Vol. 23, págs. 249-254.
7. **BAEZA YATES, R. y RIBEIRO NETO, B.** *Modern Information Retrieval.* New York : ACM Press, 1999.
8. **KORFHAGE, R. R.** *Information Storage and Retrieval.* New York : Wiley Computer Publishing, 1997.
9. **TOLOSA, G. H. y BORDIGNON, F. R.** *Introducción a la Recuperación de Información. Conceptos, modelos y algoritmos básicos.* Universidad Nacional de Luján : Creative Commons Atribución-No Comercial-Compartir Obras Derivadas Igual 2.5 Argentina License, 2008.
10. **YERBABUENA.** Gestión Documental Inteligente en la Empresa y e-Administración. *Yerbabuena Software.* [En línea] 2011. [Citado el: 8 de marzo de 2015.] <http://www.yerbabuenasoftware.com/>.
11. **LILIANA, M.** La estructura del texto. *Estructuras Textuales.* [En línea] 19 de abril de 2010. [Citado el: 15 de marzo de 2015.] <http://comunicacionlinguisticai.blogspot.com/p/la-estructura-del-texto.html>.
12. **CHOWDHURY, G.** *Introduction to modern information retrieval.* Londres : Library Association, 1999.
13. **RESEARCH, P.** *Un análisis de contenido sumamente avanzado y software de minería de texto con inigualables manejo y capacidades de análisis.* [En línea] [Citado el: 24 de marzo de 2015.] <http://provalisresearch.com/es/products/software-de-analisis-de-contenido/>.
14. **IMAGE.** Alceste. A textual statistics software. [En línea] 2012. [Citado el: 24 de marzo de 2015.] <http://www.image-zafar.com/sites/default/files/telechargements/alceste2012plusen.pdf>.
15. **BICQUELET, A.** *Introduction to Alceste.* [Documento] London : LSE, Methodology Institute, 2012.

16. **FRANCO, L.** Herramientas para el Análisis de Textos. [En línea] [Citado el: 24 de marzo de 2015.] <http://tlab.it/es/presentation.php>.
17. **BRIGIDO VIZEU, C. y JUSTO, A. M.** IRAMUTEQ: un software libre para el análisis de datos textuales. *PEPSIC Periódicos Electrónicos en Psicología*. [En línea] PEPSIC, 2 de mayo de 2013. [Citado el: 24 de marzo de 2015.] http://pepsic.bvsalud.org/scielo.php?script=sci_arttext&pid=S1413-389X2013000200016. ISSN 1413-389X.
18. **APACHE.** Apache Lucene. *Apache Software Foundation*. [En línea] [Citado el: 24 de marzo de 2015.] <http://apachefoundation.wikispaces.com/Apache+Lucene>.
19. **UCI.** *gespro. Suite de Gestión de Proyectos*. [En línea] [Citado el: 2 de marzo de 2015.] <http://gespro.ciged.prod.uci.cu>.
20. **ORTEGA MENDOZA, R. M.** *Descubrimiento Automático de Hipónimos a partir de textos no estructurados*. [Documento] Tonantzintla, Puebla : s.n., 2007.
21. *Extended Boolean information retrieval*. **SALTON, G., FOX, E.A. y WU, H.** 11, New York : s.n., 1983, Communications of the ACM, Vol. 26, págs. 1022-1036.
22. **AHO, A. V., SETHI, R. y ULLMAN, J. D.** *Compiladores: Principios, técnicas y herramientas*. s.l. : Pearson Educación, 1998. ISBN 9684443331.
23. **FRANCIS, W. y KUCERA, H.** Frequency analysis of english usage. *Lexicon and Grammar*. Boston : Houghton Mifflin, 1982.
24. **SECO NAVEIRAS, D.** *Técnicas de indexación y recuperación de documentos utilizando referencias geográficas y textuales*. 2009.
25. **SALTON, G., [ed.]**. *The SMART Retrieval System - Experiments in Automatic Document Processing*. s.l. : Prentice Hall In, 1971.
26. **ZAZO RODRÍGUEZ, A. F., FIGUEROA PIANAGUA, C. G. y ALONSO BERROCAL, J. L.** *Recuperación de información utilizando el modelo vectorial*. Salamanca : s.n., 2002.
27. **LETELIER, P.** Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP). [En línea] 15 de enero de 2006. [Citado el: 29 de marzo de 2015.] <http://www.cyta.com.ar/ta0502/v5n2a1.htm>. ISSN 1666-1680.
28. **AMBLER, S. W.** Feature Driven Development (FDD) and Agile Modeling. *Agile Modeling*. [En línea] Amblysoft Inc, 2014. [Citado el: 5 de mayo de 2015.] <http://www.agilemodeling.com/essays/fdd.htm>.
29. **LARMAN, C.** *UML y Patrones*. New York : Prentice Hall, 1999.

30. **VILLAGÓMEZ, E.** Herramienta Case. *Prezi*. [En línea] 15 de noviembre de 2013. [Citado el: 30 de marzo de 2015.] <https://prezi.com/pdojfgbqbwlz/herramientas-case/>.
31. **VÁZQUEZ ORIHUELA, A.** Visual Paradigm. *Scribd*. [En línea] [Citado el: 30 de marzo de 2015.] <http://es.scribd.com/doc/166415572/Visual-Paradigm#scribd>.
32. **MEDIAWIKI.** Programación en Java/Características del lenguaje. *WIKILIBROS*. [En línea] 9 de abril de 2014. [Citado el: 29 de marzo de 2015.] http://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Java/Caracter%C3%ADsticas_del_lenguaje.
33. **EXES.** Curso de Introducción a Java. *El lenguaje java*. [En línea] [Citado el: 29 de marzo de 2015.] http://www.mundojava.net/caracteristicas-del-lenguaje.html?Pg=java_inicial_4_1.html.
34. **PROGRAMACIÓNDESARROLLO.** Qué es un entorno de desarrollo integrado, IDE. [En línea] 15 de febrero de 2011. [Citado el: 10 de Abril de 2015.] <http://programaciondesarrollo.es/que-es-un-entorno-de-desarrollo-integrado-ide/>.
35. **RODRÍGUEZ, T.** Genbeta: dev. *Eclipse IDE*. [En línea] 10 de enero de 2014. [Citado el: 25 de abril de 2015.] <http://www.genbetadev.com/herramientas/eclipse-ide>.
36. **NETBEANS.** *Oracle Corporation and/or its affiliates*. [En línea] [Citado el: 29 de marzo de 2015.] <http://netbeans.org/>.
37. **CAROLINA MARTÍNEZ, I.** *Modelo Conceptual / Modelo de Dominio*. [Documento] Caracas : Universidad Simón Bolívar, 2010.
38. **WELLS, D.** The Rules of Extreme Programming. *Extreme Programming*. [En línea] [Citado el: 20 de mayo de 2015.] <http://www.extremeprogramming.org/rules.html>.
39. **AMBLER, S. W.** Agile Modeling and eXtreme Programming (XP). *Agile Modeling*. [En línea] Ambyssoft Inc., 2012. [Citado el: 20 de mayo de 2015.] <http://agilemodeling.com/essays/agileModelingXP.htm>.
40. **BOLIVARIANA, U.** Programación extrema XP. *Ingeniería de Software*. [En línea] [Citado el: 20 de mayo de 2015.] http://ingenieriadesoftware.mex.tl/52753_XP---Extreme-Programing.html.
41. **UNIVERSO.** Programación extrema: "Metodología para desarrollo ágil de aplicaciones". *Universo: El periódico de los universitarios*. [En línea] 11 de junio de 2012. [Citado el: 20 de mayo de 2015.] http://www.uv.mx/universo/486/infgral/infgral_15.html.
42. **GOMEZ, D., ARANDA, E. y FABREGA, J.** Programación Extrema. *Universidad del Valle*. [En línea] [Citado el: 20 de mayo de 2015.] <http://eisc.univalle.edu.co/materias/WWW/material/lecturas/xp.pdf>.
43. **PMO.** ¿Qué son las historias de usuario? *PMOinformatica.com*. [En línea] 24 de abril de 2013. [Citado el: 14 de abril de 2015.] <http://www.pmoinformatica.com/>.

44. **A. SÁNCHEZ, E., LETELIER, P. y CANÓS, J. H.** *Mejorando la gestión de historias de usuario en eXtreme Programming*. [Documento] Valencia : s.n., 2004.
45. **VILLAMIZAR SUAZA, K.** *Definición de equivalencias entre historias de usuario y especificaciones en UNLENCEP para el desarrollo ágil de software*. [Documento] Medellín : s.n., 2013.
46. **EHEVERRY TOBÓN, L. M. y DELGADO CARMONA, L. E.** *Caso práctico de la metodología ágil XP al desarrollo de software*. [Documento] Pereira : s.n., 2007.
47. **PRESSMAN, R. S.** *Ingeniería de Software. Un enfoque práctico*. Quinta. s.l. : Mc Graw Hill, 2002.
48. **L. BASS, P. y CLEMENTS, R. K.** *Software Architecture in Practice*. Segunda. New York : Addison Wesley, 2003.
49. **FLORES CUETO, J. J. y BERTOLOTTI ZUÑIGA, C.** Diagrama de clases en UML. *Diagrama de clases en UML*. [En línea] [Citado el: 14 de abril de 2015.] <http://es.scribd.com/doc/31096724/Diagrama-de-Clases-en-UML#scribd>.
50. **INNOCUO.** Patrones de diseño, la importancia de aprenderlos. *innocuo*. [En línea] 11 de septiembre de 2006. [Citado el: 14 de abril de 2015.] <http://blog.innocuo.com/2006/09/patrones-de-diseno-la-importancia-de-aprenderlos/>.
51. **MARTÍNEZ JUAN, F. J.** *Guía de construcción de software en java con patrones de diseño*. Oviedo : s.n.
52. **DRAKE, J.M.** Programación Concurrente. *CTR - Computadores y Tiempo Real*. [En línea] [Citado el: 27 de abril de 2015.] http://www.ctr.unican.es/asignaturas/procodis_3_ii/doc/procodis_1_01.pdf.
53. **BUSCAPALABRA.** Letras iniciales y longitud de las palabras en español. *Busca palabra*. [En línea] 2015. [Citado el: 15 de abril de 2015.] <http://www.buscapalabra.com/blog/78-iniciales-longitud-palabras-espanol.html>.
54. **TAN, P.N., STEINBACH, M. y KUMAR, V.** *Introduction to Data Mining*. s.l. : Addison-Wesley (2005), 2005. pág. 500. ISBN 0-321-32136-7.
55. **DEUTSCH, M.** Verification and Validation. [ed.] Jensen R. y Tonies C. *Software Engineering*. New York : Prentice Hall, 1979, págs. 329-408.
56. **MARTÍNEZ DE SOUSA, J.** *Diccionario de Lexicografía Práctica*. [Documento] Barcelona : s.n., 1995.
57. **BECK, K.** *Extreme Programming Explained*. New York : Addison-Wesley Professional, 1999.
58. **MOORE, J. W., y otros, [ed.].** *SWEBOOK: Guide to the Software Engineering Body of Knowledge*. California : IEEE Computer Society, 2004. ISBN 0-7695-2330-7.
59. **UNAD.** Capítulo 9. Estrategia de prueba de software. *Universidad Nacional Abierta y a Distancia*. [En línea] [Citado el: 20 de mayo de 2015.]

http://datateca.unad.edu.co/contenidos/301404/301404_ContenidoEnLinea/captulo_9_estrategias_de_prueba_del_software.html.

60. **SOMMARIVA, A.** Pruebas Unitarias, Parte 1: Introducción y utilización de objetos simulados (Mock). *MicroGestion*. [En línea] 24 de febrero de 2014. [Citado el: 8 de mayo de 2015.] <http://www.microgestion.com/index.php/mg-developers/articulos/74-unit-test-part1-mock>.
61. **MAVEN.** JUnit. [En línea] 12 de abril de 2014. [Citado el: 13 de mayo de 2015.] <http://junit.org/>.
62. **KEN AUER, R. M.** *Extreme Programming Applied*. New York : Addison-Wesley Professional, 2001.
63. **VILLAYANDRE LLAMAZARES, M.** Lingüística computacional II. Curso monográfico sobre Lingüística de corpus. *Departamento de Filología Hispánica y Clásica*. [En línea] [Citado el: 27 de marzo de 2015.] <http://fhyc.unileon.es/Milka/LCII/Corpus4.pdf>.
64. **TORUELLA, J. y JOAQUIM, L.** Diseño de Corpus Textuales y Orales. *Laboratorio de Tecnologías Lingüísticas*. [En línea] 1999. [Citado el: 27 de marzo de 2015.] http://latel.upf.edu/traductica/lc/material/torruella_llisterri_99.pdf.
65. **GÓMEZ DÍAZ, R.** La lematización en español: Una aplicación para la recuperación de información. *Revista española de Documentación Científica*. [En línea] 2005. [Citado el: 27 de marzo de 2015.] <http://redc.revistas.csic.es/index.php/redc/article/viewFile/301/348>. ISBN 84-9704-186-0.
66. **ALEGSA, L.** ¿Qué significa API? - Información sobre API. *Diccionario de Informática y Tecnología*. [En línea] 30 de mayo de 2013. [Citado el: 27 de marzo de 2015.] <http://www.alegsa.com.ar/Dic/api.php>.
67. *Técnicas de representación de textos para clasificación no supervisada*. **COBO RODRÍGUEZ, G.** 37, Barcelona : s.n., 2006, Procesamiento del lenguaje natural. ISSN 1135-5948.

BIBLIOGRAFÍA

1. *Métodos "I + D" de la Informática*. **BARCHINI, G. E.** 5, Santiago del Estero: Revista de Informática Educativa y Medios Audiovisuales, 2005, Vol. 2. ISSN 1667-8338.
2. *Técnicas de representación de textos para clasificación no supervisada*. **COBO RODRÍGUEZ, G.** 37, Barcelona: s.n., 2006, Procesamiento del lenguaje natural. ISSN 1135-5948.
3. **ALVAREZ DE ZAYAS, C.** *Metodología de la Investigación Científica*. Santiago de Cuba: Universidad de Oriente, 1995.
4. **HERNÁNDEZ LEÓN, R. A.** *El Proceso de Investigación Científica*. Ciudad de la Habana: Editorial Universitaria, 2011. ISBN 978-959-16-1307-3.
5. **GRAU, R.** *Metodología de la Investigación*. Ibagué: EL POIRA Editores S. A, 2004. ISBN 958-8028-10-8.

6. **GONZÁLEZ CASTELLANOS, R. A.** *Metodología para la Investigación Científica para las Ciencias Técnicas. 1era Parte: Diseño Teórico y Formulación del Proyecto de Investigación.* Matanzas: Universidad de Matanzas, 2003.
7. **GONZÁLEZ CASTELLANOS, R. A.** *Metodología para la Investigación Científica para las Ciencias Técnicas. 2da Parte: Organización y Ejecución de la Investigación.* Matanzas: Universidad de Matanzas, 2003.
8. **MOREIRO GONZÁLEZ, J. A.** *El Contenido de los Documentos Textuales: Su Análisis y Representación Mediante el Lenguaje Natural.* [Documento] Gijón: Ediciones Trea, 2004.
9. **VELÁZQUEZ MOO, M.** *Clasificación de documentos usando Máquinas de Vectores de Apoyo.* [Documento] Yucatán: Universidad de Yucatán, Facultad de Matemática, 2012.
10. **CERVANTES CANALAES, J.** *Clasificación de grandes conjuntos de datos vía Máquinas de Vectores Soporte y aplicaciones en sistemas biológicos.* [Documento] México D.F, 2009.
11. **HAN, J.** *Data Mining Concepts and Techniques Second Edition.* Illinois: Morgan Kaufman Publishers, 2006. ISBN 978-1-55860-901-3.
12. **ECKEL, B.** *Piensa en Java Segunda Edición.* Madrid: Prentice Hall, 2002. ISBN 84-205-3192-8.