Universidad de las Ciencias Informáticas

Facultad 2

Solución Informática para la representación matemática de archivos digitales textuales

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

Autor: Osciel Martínez Silva

Tutores: Ing. Pável Reyes Estévez

Ing. Andis Eloy Yero Guevara

La Habana, junio de 2018

"Año 60 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor del trabajo de diploma “Solución informática para la representación matemática de archivos digitales textuales” y autorizo 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 2018

_____

Osciel Martínez Silva

Firma Autor

_____ _____

Ing. Andis Eloy Yero Guevara

Firma Tutor

Ing. Pável Reyes Estévez

Firma Tutor

AGRADECIMIENTOS

Les agradezco a mí:

Mamá por ser todo para mí, por estar siempre al tanto de lo que me pasa y por ese cariño infinito que no existen palabras para describir.

Papá por enseñarme a ser un hombre desde que nací, por inculcarme esos defectos y virtudes que hoy me han convertido en lo que soy.

Mis abuelas Mirella por ese cariño incondicional y esa ternura que la caracteriza y a mi abuela Regla que aunque hoy ya no se encuentre entre nosotros físicamente siempre la llevo conmigo y sé que desde donde esta se siente súper orgullosa de mí.

A mis hermanos que nunca dejaron de confiar en mí.

A mis tíos en especial a mi tío Carlito que más que un tío lo considero mi segundo papá.

A Dianelys por ser esa personita especial que en estos últimos dos años ha sabido quererme como soy sin ni más.

A mis suegros que desde que llegue a sus vidas han sabido acogerme como un hijo más.

A mis dos cuñaditos por ser mis chamas en el FIFA y demostrarme su carriño.

Agradecer a todos esos hermanos que convivieron el día a día en estos 5 años al Mandy, al Josue , Migue, Arain, Handy, Yaicel, Néstor, Osniel, el Wilber, el Yoa, el Bienve, el Ale, Alexander, el Chino, Pablito Pablon, Ronny y otros que no mencione, gracias de verdad a todos por esos momentos que se quedaran siempre en mí.

Gracias a todos los hermanos del futbol por todos los días aguantar mis locuras y si no me quedo aquí que es lo más probable recuerden siempre que Papa es Papa.

Agradecer sobre todo a mis tutores Andis y Pável por la paciencia, el tiempo que tuvieron en el transcurso de este año para mi formación como ingeniero.

DEDICATORIA

Dedico mi Trabajo de Diploma:

A toda mi familia, amigos y en especial a mí Mamá y a mí Papá.

RESUMEN

La recuperación de información es el área del conocimiento mediante la cual se localiza y accede a los recursos de información con el propósito de dar solución a necesidades específicas. La representación, el almacenamiento, la organización y el acceso a elementos de información forman parte de la misma. En la actualidad, la representación del contenido de archivos digitales en la recuperación de información se dificulta, debido a la necesidad de clasificar archivos digitales para la posterior identificación y el agrupamiento de documentos con características comunes. El objetivo de esta investigación consiste en el desarrollo de un componente informático para la representación de archivos digitales textuales en contribución a la recuperación de información. Para dar cumplimiento al mismo 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 la resolución del problema planteado. Se hace uso del modelo de espacio vectorial, el cual permite la representación del corpus textual de los archivos digitales mediante el álgebra de vectores. Basado en dicho modelo se diseña y desarrolla un algoritmo que permite obtener una matriz de semejanza que contiene los valores de similitud entre archivos textuales representados por vectores. Este proceso es necesario para una posterior etapa de organización y clasificación automática de archivos digitales textuales en sistemas informáticos de recuperación de información.

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

Abstract

Information retrieval is the area of knowledge through which information resources are located and accessed with the purpose of solving specific needs. The representation, storage, organization and access to information elements are part of it. At present, the representation of the content of digital files in information retrieval is difficult, due to the need to classify digital files for the subsequent identification and grouping of documents with common characteristics. The objective of this research is the development of a computer component for the representation of textual digital files in contribution to information retrieval. In order to comply with it, a background study of information retrieval systems is carried out and the software development environment is prepared to solve the problem. The vector space model is used, which allows the representation of the textual corpus of the digital files through the vector algebra. Based on this model, an algorithm is designed and developed to obtain a similarity matrix that contains the values of similarity between textual files represented by vectors. This process is necessary for a later stage of organization and automatic classification of textual digital files in information retrieval systems.

Keywords: textual digital file, textual corpus, vector space model, information retrieval, mathematical representation.

ÍNDICE

INTRODUCCIÓN 11

Capítulo 1. Fundamentación teórica 18

1.1 Conceptos fundamentales 18

1.2 Estudio de los antecedentes de la investigación 21

1.2.1 Modelos de recuperación de información 21

1.2.2 Estado del arte 24

Metodología de desarrollo 28

1.3 Herramientas y lenguajes de desarrollo 30

1.3.1 Lenguaje de desarrollo 31

1.3.2 Herramientas de modelado 31

Conclusiones del capítulo 32

Capítulo 2. Análisis y diseño de la solución propuesta 33

Modelo de la propuesta de solución 33

2.1 Planeación 34

2.1.1 Modelo de dominio 34

2.1.2 Historias de usuario 35

2.1.3 Matriz de trazabilidad 37

2.1.4 Tiempo de ejecución del proyecto 38

2.1.5 Iteraciones 39

2.1.6 Plan de entrega 40

2.2 Diseño de la Propuesta de Solución 40

2.2.1 Arquitectura de software 41

2.2.2 Diagrama de clases 42

2.2.3 Patrones de diseño 44

2.3 Implementación de la propuesta de Solución 48

2.3.1 Programación concurrente 51

2.3.2 Estructuras para la representación del corpus textual 52
2.3.3 Corrida del componente implementado 53
2.3.4 Ejemplo del algoritmo que implementa el Modelo Espacio Vectorial 54
Conclusiones del capítulo 57
Capítulo 3. Comprobación del funcionamiento de la propuesta de solución 58
3.1 Comprobación del funcionamiento del componente desarrollado 58
Estrategia de prueba 58
Pruebas unitarias 59
Pruebas de aceptación 66
Conclusiones del capítulo 70
CONCLUSIONES 71
RECOMENDACIONES 72
REFERENCIAS 73
BIBLIOGRAFÍA 78
ANEXOS 79
Anexo 1. Historias de usuario 79
Anexo 2. Entidades del diagrama de clases del sistema 80
Anexo 3. Pruebas unitarias 83
GLOSARIO DE TÉRMINOS 88

ÍNDICE DE FIGURAS

Figura 1.1. Modelo de dominio (30) 28
Figura 2.1 Modelo de la propuesta de solución 33
Figura 2.2. Modelo de dominio 34
Figura 2.3. Matriz de trazabilidad RF-RF 38
Figura 2.4. Arquitectura de la propuesta de solución 42
Figura 2.5. Diagrama de clases UML 43
Figura 2.6. Implementación del patrón Experto 45
Figura 2.7. Implementación del patrón Creador 45
Figura 2.8. Implementación del patrón Alta Cohesión 46
Figura 2.9. Implementación del patrón Singleton 47
Figura 2.10. Implementación del patrón Scheduler 51
Figura 2.11. Tiempo de corrida del algoritmo 53
Figura 3.1. Pruebas unitarias 66
Figura 3.2. Pruebas de aceptación 69

ÍNDICE DE TABLAS

Tabla 2.1. HU-1. Construir vector de término 36
Tabla 2.2. HU-3. Construir diccionario de términos 36
Tabla 2.3. HU-5. Construir matriz de semejanza 37
Tabla 2.4. Estimación por historia de usuario 39
Tabla 2.5. Plan estimado de duración de las iteraciones 39
Tabla 2.6. Plan de entrega de las iteraciones 40
Tabla 2.7. Pseudocódigo. Algoritmo para la representación de corpus textual 49
Tabla 2.8. Pseudocódigo. Procedimiento para actualizar pesos 50
Tabla 2.9. Documentos-Términos 54
Tabla 2.10 . Frecuencia del término en el corpus 55
Tabla 2.11. TF*IDF 55
Tabla 2.12. Consulta, términos y frecuencia 56
Tabla 2.13. Similitud de los documentos 56
Tabla 3.1. HU-1 Construir vector de término. Clase Vector 61
Tabla 3.2. HU-2 Cálculo de la relevancia de los términos. Clase Term 62
Tabla 3.3. HU-3 Construir diccionario de términos. Clase TrieImpl 63
Tabla 3.4. HU-4 Almacenar diccionario de términos. Clase ReadWriteFile 64
Tabla 3.5. HU-6 Almacenar la matriz de semejanza en un archivo binario. Clase ReadWriteFile 65
Tabla 3.6. Pruebas de aceptación. Iteración 1 67
Tabla 3.7. Pruebas de aceptación. Iteración 2 69

INTRODUCCIÓN

Históricamente el hombre ha necesitado de medios sobre los cuales representar todo acerca del mundo que lo rodea y de reflejar de alguna manera su evolución. La escritura ha sido el mecanismo tradicional y fundamental que soporta su conocimiento en el tiempo. Dicha evolución ha facilitado la existencia de diferentes medios de representación de la escritura llegando a nuestros días donde la información se representa digitalmente y es posible su almacenamiento y distribución masiva de forma simple y rápida, a través de la red de computadoras. La digitalización abrió nuevos horizontes en las formas que el hombre pude tratar con la información que produce (1).

La información es un conjunto de datos procesados, que constituye un mensaje, esta permite resolver problemas y ayudar en la toma de decisiones, ya que su aprovechamiento racional es la base del conocimiento (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.

Los archivos existen desde que el hombre decidió fijar por escrito sus relaciones como ser social y constituyen la memoria de las instituciones y de las personas. A lo largo de los años estos se han encargado de evidenciar la evolución del hombre en la historia. En un primer momento fueron utilizados con el fin de guardar actas públicas tanto de carácter político-administrativo como de carácter notarial (edad media) (4).

Con la evolución del hombre, estos fueron tomando auge hasta llegar a la actualidad, donde juega un papel imprescindible, en la toma de decisiones, la solución de problemas y el conocimiento que genera en el hombre. Los archivos constituyen un eslabón fundamental en la actualidad, juegan un papel importante en la sociedad promoviendo el conocimiento, pero fundamentalmente por su carácter cultural, económico, político e histórico. Por todo ello, se hace necesario efectuarle un tratamiento adecuado con el fin de lograr una mejor comprensión, organización, recuperación de la información que contiene e identificar su importancia real para la organización.

El desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC) ha propiciado no sólo la producción de una cantidad cada vez mayor de documentos, sino también se ha acrecentado el papel fundamental de los procesos de digitalización y recuperación de la información para su preservación. Se

hace inminente que se extienda el uso de los archivos en formato digital, debido a su importancia en la obtención de información para la organización. Un archivo digital o fichero es una unidad de datos o información almacenada en algún medio que puede ser utilizada por aplicaciones de computadora (5).Dichos ficheros facilitan la interpretación de su contenido, pueden ser almacenados durante largos periodos de tiempo y consultados simultáneamente a través de la red de computadoras.

Luego que los archivos son digitalizados una de las tareas principales es la recuperación de la información que poseen, un problema que va en aumento debido a la variedad de estructuras que existen. La Recuperación de Información (RI) abarca el conjunto de acciones, métodos y procedimientos para la representación, almacenamiento y organización; su objetivo fundamental es obtener los documentos ordenados en función del grado de relevancia, para responder a las necesidades del usuario. Un Sistema de RI (SRI) es un programa que implementa un modelo de RI, posee tres componentes principales: la base de datos documental, el subsistema de consultas y el mecanismo de recuperación (6).

Existen diversos modelos para la recuperación de la información: el modelo booleano, basado en la teoría de conjuntos del álgebra de Boole, el modelo Fuzzy que intenta solucionar los problemas del modelo booleano (la relevancia es binaria) basándose en la lógica difusa, el modelo probabilístico, que parte de la presencia o ausencia de los términos de la consulta en los documentos de la colección y el modelo vectorial, que se basa en el álgebra de vectores para el filtrado, recuperación, indexado y cálculo de relevancia de información. 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 (1) (7) (8).

El problema de la RI se estudia desde dos puntos de vista distintos: una es el computacional, que tiene que ver con la construcción de estructuras de datos y algoritmos eficientes que mejoran la calidad de la respuesta, y la otra perspectiva es el humano, que se corresponde al estudio del comportamiento y de las necesidades de los usuarios. Dicho proceso se encuentra determinado por el tipo de información y el medio de almacenamiento que se emplee (1). En el presente trabajo la forma de representar la información es la documental digital tal como: escritos, libros, revistas, documentos bibliográficos, cartas y todos aquellos archivos digitales de carácter histórico que su contenido sea texto.

Debido al cúmulo de información que manejan las organizaciones, es inminente que exceda las capacidades de tratamiento con las que suelen contar los recursos humanos, por lo que se hace evidente la aplicación de una técnica de recuperación de información para tratar los archivos almacenados y obtener resultados relevantes del análisis de los eventos históricos registrados en ellos (9).

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 (10). 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 (10). 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.

Los sistemas de recuperación de información 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 (11). 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.

En Cuba se avanza en la informatización del país con planes que sea sostenible a largo plazo y garantice un acceso pleno de todos sus ciudadanos a las Tecnologías de la Información y las Comunicaciones. La Universidad de las Ciencias Informáticas (UCI), centro educacional encargado de formar profesionales altamente calificados en la rama de la informática, está inmerso en esta tarea vital. Uno de sus centros de desarrollo es el Centro de la 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, Gestión bibliotecaria y Centros de Documentación (12). Dentro de los productos desarrollados se encuentran el Sistema de Gestión de Archivos Históricos Xabal-Arkheia, evolucionado hasta su versión 3.0 el cual contempla la informatización de los procesos incorporación, consulta y conservación de documentos de archivo.

El sistema informático Xabal-Arkheia 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, estos pueden ser 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.

Para la organización según el contenido de los archivos digitales textuales un 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.

La tarea de extracción del contenido de un archivo digital textual 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.

Por tanto, contar con un componente informático capaz de realizar la clasificación automática de archivos digitales textuales es necesario para la etapa de organización y clasificación de archivos digitales textuales en contribución a la recuperación de información.

De acuerdo con lo anteriormente expuesto se plantea en siguiente **problema a resolver:** ¿Cómo representar en un modelo matemático los archivos digitales textuales en contribución a la clasificación automática de documentos y 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 se define como **objetivo general:** Desarrollar una solución informática que permita mediante un modelo matemático la representación de archivos digitales textuales en contribución a la clasificación automática de documentos y a la recuperación de información

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 limpieza de datos en el procesamiento del corpus textual de archivos digitales textuales.
- ✓ Implementar un componente informático para la representación de estructuras 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 fundamentales de la recuperación de información en archivos digitales textuales?
- ✓ ¿Cuáles son los modelos de recuperación de información más utilizados en la actualidad, ventajas y desventajas?
- ✓ ¿Cuál es el modelo de recuperación de información más adecuado para la implementación de la propuesta de solución?
- ✓ ¿Cuál estrategia utilizar para lograr baja complejidad algorítmica en problemas de alta dimensionalidad en la recuperación de información de archivos digitales textuales?
- ✓ ¿Qué estrategia es necesaria definir para comprobar el funcionamiento de la solución desarrollada?

Para dar cumplimiento al objetivo general se definieron las siguientes **tareas de la investigación:**

- Elaboración del marco teórico de la investigación para garantizar el basamento y fundamento científico de la misma.
- Análisis de diferentes modelos de recuperación de información para la definición de los antecedentes de la investigación.
- Identificación de las principales soluciones informáticas que utilicen modelos 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.
- Implementación de un algoritmo basado en el modelo de recuperación de información seleccionado para la representación del corpus textual de archivos digitales textuales.
- Descripción de las estructuras de datos utilizadas para la representación del corpus textual de los documentos digitales.
- Selección de la metodología, las herramientas y las tecnologías a utilizar para el desarrollo de la solución propuesta.
- Selección de técnicas para disminuir el tiempo de ejecución 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.

- Realización de las pruebas de software para validar el correcto funcionamiento de la propuesta de solución.

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 desarrollo está compuesto por tres capítulos, los cuales se encuentran estructurados de la siguiente forma:

Capítulo 1. Fundamentación teórica: Se describen 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. Análisis y diseño de la solución propuesta: 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 del componente desarrollado.

Capítulo 1. Fundamentación teórica

El presente capítulo tiene como objetivo plantear los aspectos teóricos que sirven de soporte para la elaboración del componente. Se exponen conceptos relevantes para lograr una mejor comprensión sobre 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. Por último, se presentan la metodología, herramientas, tecnologías y lenguajes a utilizar en la implementación del componente.

1.1 Conceptos fundamentales

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 (13). 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 (13). 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, 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 tanto 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 (14). 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 (15). Incluye la representación, el almacenamiento, la organización y el acceso a elementos de información (16). Facilita la localización y presentación a un sujeto de información relevante a una necesidad de información expresada como una pregunta (17). 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 (18).

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 el comienzo y fin de una palabra, además de reconocer símbolos no alfabéticos como números o caracteres especiales. También estandariza todas las palabras ya sean mayúsculas o minúsculas (1). 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 remplazar las palabras por su raíz, 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 (1). 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 (1). Algunos estudios determinan que se llegan a reducir los archivos de índices en un 40 por ciento (19).

1.2 Estudio de los antecedentes de la investigación

Luego de esclarecer los conceptos fundamentales involucrados en la investigación, el siguiente punto consiste en entender cómo se realiza la recuperación de información. La situación relacionada con el alcance de la investigación, consiste en la representación matemática de los archivos digitales textuales. La forma de representar los documentos y de medir su similitud son características que definen los modelos de recuperación de información.

1.2.1 Modelos de recuperación de información

Los modelos de recuperación se dividen en dos grupos: clásicos y estructurados (20). En el primero se incluyen el modelo booleano, el modelo probabilístico y el modelo vectorial. 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. En el segundo grupo se encuentran modelos estructurados correspondientes a listas de términos sin solapamiento y a nodos próximos (son modelos escasamente difundidos).

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 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 (8).

También existen otros modelos alternativos como el modelo Fuzzy, que tiene como principal función solucionar los problemas del modelo booleano y los modelos basados en la navegación entre páginas web que son de tres tipos: estructura plana, estructura guiada e hipertexto. El primero es una simple lectura de un documento aislado del contexto, el segundo incorpora la posibilidad de facilitar la exploración organizando los documentos en una estructura tipo directorio con jerarquía de clases y subclases y el tercero se basa en la idea de un sistema de información que de la posibilidad de adquirir información de forma no estrictamente secuencial sino a través de nodos y enlaces (16).

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 (1) (8).

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 (1).

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 orden 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 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) (8).

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 (8).

Las respuestas que emite la aplicación de ese modelo son menos eficientes que las del modelo booleano, además de que todos los documentos seleccionados no son realmente relevantes. Por lo que se puede considerar la posibilidad de que un documento sea relevante o no, dado que ya haya sido 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 (1) (8) (21). Por lo 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 (1) (8).

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 orden 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 (7).

Modelo Fuzzy

El Modelo Fuzzy intenta solucionar los problemas del modelo booleano (la relevancia es binaria) basándose en la lógica difusa. De manera general para cada término se define un conjunto difuso donde cada documento tendrá un determinado grado de pertenencia.

En este modelo al igual que en el Booleano las consultas están compuestas por términos indexados y por los operadores AND, OR y NOT, por lo cual son expresiones booleanas. Para facilitar el manejo de dicha consulta esta puede llevarse a Forma Normal Disyuntiva y luego a Forma Normal Disyuntiva Completa respecto a todos los términos indexados del sistema. Las ventajas del modelo radican en que es capaz de

recuperar documentos que no incluyan exactamente los mismos términos de la consulta, a diferencia del modelo Booleano tradicional define un orden entre los documentos recuperados a través de una función de pertenencia y tiene en cuenta la relación existente entre los índices, almacenada en la matriz de correlación. En cuanto a sus desventajas, su implementación puede ser un poco más costosa que la de otros modelos, entre otras cosas por la gran cantidad de cálculos aritméticos que deben realizarse para evaluar una consulta y no tiene en cuenta la frecuencia de ocurrencias de un término en un documento. O sea la función de comparación no prioriza documentos que tengan 100 veces la palabra casa, sobre documentos que contengan una sola ocurrencia de esta palabra (22).

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 modelo booleano no aporta una adecuada representación del contenido digital textual, no permite establecer medidas de similitud entre documentos lo que implica que cada término dentro de la colección tiene el mismo nivel de importancia. El modelo probabilístico únicamente permite la clasificación independiente de un documento, determina la importancia de un documento independientemente de la colección. El modelo fuzzy no tiene en cuenta la frecuencia de ocurrencias de un término en un documento y su implementación puede ser un poco más costosa que la de otros modelos. En cuanto al modelo vectorial permite representar un archivo digital textual sin afectar el contenido semántico y 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, por lo que el equipo de desarrollo lo considera una buena alternativa y una fuerte estrategia para representar colecciones de documentos.

1.2.2 Estado del arte

Desde los orígenes de la recuperación de información uno de los temas a los que más atención se le ha prestado es a la evaluación de los sistemas desarrollados dentro del campo. Para concluir el estudio de los antecedentes de la investigación se analizan algunos sistemas que implementen modelos de recuperación de información bajo la tarea de representación del corpus textual de documentos. Esta actividad se realiza para determinar si estos sistemas funcionan correctamente y si un cambio determinado mejora su funcionamiento y resultados.

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 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 claves, 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 (23). 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 (24) (25). 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 (26).

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. Su principal objetivo 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 (27).

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 pero gráficamente de interés 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 (27).

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 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 (28). 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 estado del arte

Luego del análisis realizado sobre los sistemas que realizan la recuperación de información y representación del corpus textual de documentos se arriba a las siguientes conclusiones:

- **WordStat, Alceste y T-LAB** muestran un conjunto de características importantes 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. Sin embargo son soluciones informáticas de plataforma Windows, esto imposibilita su integración con varios tipos de software y limita su capacidad multiplataforma, además son softwares privativos 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.

  El objetivo de T-LAB es el análisis lingüístico de textos y no permite comparativa de documentos u obtención del corpus representado.
- **Lucene** cuenta con un conjunto de funcionalidades para tratar el corpus textual de los documentos digitales, es multiplataforma y permite la integración de sus componentes con otros sistemas. Pero 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.
- **IRAMUTEQ** se caracteriza por su incapacidad de realizar los diferentes tipos de análisis de datos textuales a una colección de archivos de textos, por tanto, 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.

Por lo que se hace necesario implementar una solución capaz de integrarse a sistemas multiplataforma, que permitan el procesamiento de datos textuales así como su representación matemática en una determinada estructura.

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 la 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 (29). Para guiar el desarrollo del software a implementar se decide el empleo de una metodología ágil. Dicha selección se basa en la existencia de una estrecha relación con el cliente, llegando este a formar parte del equipo, se necesitan constantes entregas del avance en el proceso de desarrollo y que permita estandarizar el desarrollo de software que se propone para la actividad productiva de la UCI. También se tiene en cuenta que durante este proceso el software se puede ver sometido a constantes cambios. En la figura 1.1 se exponen varias características de ambos tipos de metodologías.

Figura 1.1. Modelo de dominio *(30)*

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 (31).

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 del 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 (29). Esta metodología se basa más en la administración del proyecto que en la propia programación o creación del producto.

AUP

El Proceso Unificado Ágil de Scott Ambler o Agile Unified Process (AUP) en inglés es una versión simplificada del Proceso Unificado de Rational (RUP). Este describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. El AUP aplica técnicas ágiles incluyendo:

- Desarrollo Dirigido por Pruebas (test driven development-TDD en inglés).
- Modelado ágil.
- Gestión de Cambios ágil
- Refactorización de Base de Datos para mejorar la productividad.

Al igual que en RUP, en AUP se establecen cuatro fases que transcurren de manera consecutiva.

Fases de AUP

1. Inicio: El objetivo de esta fase es obtener una comprensión común cliente-equipo de desarrollo del alcance del nuevo sistema y definir una o varias arquitecturas candidatas para el mismo.
2. Elaboración: El objetivo es que el equipo de desarrollo profundice en la comprensión de los requisitos del sistema y en validar la arquitectura.
3. Construcción: Durante la fase de construcción el sistema es desarrollado y probado al completo en el ambiente de desarrollo.
4. Transición: El sistema se lleva a los entornos de preproducción donde se somete a pruebas de validación y aceptación y finalmente se despliega en los sistemas de producción (32).

AUP-UCI

Al no existir una metodología de software universal, ya que toda metodología debe ser adaptada a las características de cada proyecto (equipo de desarrollo, recursos, etc.) exigiéndose así que el proceso sea configurable. Se decide hacer una variación de la metodología AUP, de forma tal que se adapte al ciclo de vida definido para la actividad productiva de la UCI. Esta metodología tiene entre sus objetivos aumentar la calidad del software que se produce, de ahí la importancia de aplicar buenas prácticas, para ello se apoya en el Modelo CMMI-DEV v1.3. El cual constituye una guía para aplicar las mejores prácticas en una entidad desarrolladora. Estas prácticas se centran en el desarrollo de productos y servicios de calidad.

De las 4 fases que propone AUP (Inicio, Elaboración, Construcción, Transición) se decide para el ciclo de vida de los proyectos de la UCI mantener la fase de Inicio, pero modificando el objetivo de la misma, se unifican las restantes 3 fases de AUP en una sola, a la que llamaremos Ejecución y se agrega la fase de Cierre. (33)

A partir de la selección de la metodología y siendo consecuente con el proceso productivo de la UCI. Se define que el escenario seleccionado para la presente investigación es el 4. Para esto se tiene en cuenta que el modelado del negocio se encuentra bien definido y el cliente estará siempre acompañando al equipo de desarrollo.

Por tanto, se mantiene el estándar del proceso de desarrollo de software propuesto en la UCI, dando cumplimiento a las líneas definidas por CMMI-DEV v1.3.

1.3 Herramientas y lenguajes de desarrollo

Para la realización de la propuesta de solución se realizó un estudio, para la selección de aquellas herramientas que respondieran a las necesidades del sistema a implementar. A continuación se mostrará el resultado del estudio realizado.

1.3.1 Lenguaje 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 y robusto, (34) (35). 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 8.1.

1.3.2 Herramientas de modelado

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 (36). 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.

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 (37). 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.

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.

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 los sistemas WordStat, Alceste, T-LAB, Iramuteq y Lucene.

Por tanto, se concluyó que es necesario implementar un componente informático 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, AUP y AUP-UCI. A partir de este análisis se seleccionó el escenario número cuatro de la metodología AUP en su variante UCI para guiar el desarrollo de software teniendo en cuenta que permite estandarizar el proceso de desarrollo del software que se propone para la actividad productiva de la UCI. En apoyo a la metodología se seleccionó la herramienta CASE Visual Paradigm 8.0 que permite el modelado de artefactos 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. Análisis y diseño de la solución propuesta

El presente capitulo tiene como objetivo describir los principales artefactos relacionados con el análisis, diseño e implementación de la propuesta de solución al problema identificado en la actual investigación. Dicho 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 y el diagrama de clases. 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

Luego del estudio de los antecedentes y consecuentemente con el objetivo de la investigación, de diseñar e implementar un componente informático que permita la representación matemática del corpus textual de archivos digitales, se realiza la propuesta de solución. El componente debe implementar el modelo de espacio vectorial y proporcionar la arquitectura necesaria para la utilización de la estructura de dato Trie la cual permite representar y organizar la colección de documentos.

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 la propuesta de solución cuenta con tres momentos fundamentales: entrada, procesamiento y representación. El flujo de entrada consiste en una colección de

archivos digitales previamente filtrados y codificados. Luego el procesamiento de la colección está dividido en dos etapas fundamentales, la primera se basa en la construcción del diccionario de datos del corpus textual utilizando la estructura Trie, y la segunda consiste en la aplicación del modelo de espacio vectorial, donde se calculan la importancia de cada término con respecto al documento, para así construir la matriz de documentos. Finalmente se obtiene la matriz de semejanza de los documentos, constituyendo la representación textual del corpus de tales archivos.

2.1 Planeación

La planeación es la etapa inicial de todo desarrollo de software. 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 (29). En esta sección se describen dos elementos fundamentales, los cuales son: Historias de Usuario y Ejecución del Proyecto. Dichos elementos recogen las características principales del componente 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 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 (38).La figura 2.2 muestra una representación estática del entorno real del proyecto mediante el modelo de dominio.

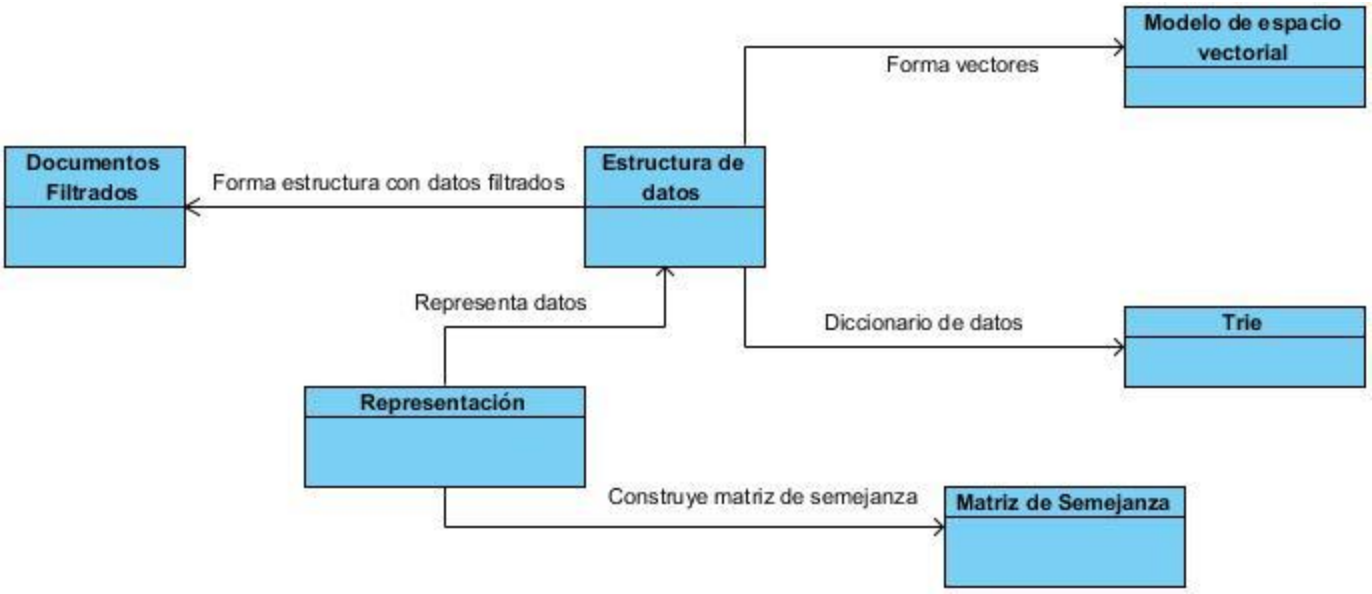


Figura 2.2. Modelo de dominio

En el modelo de dominio mostrado se aprecia que los documentos a procesar ya se encuentran filtrados y estandarizados a través de un decodificador. 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.

2.1.2 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 donde 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 (39). 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 qué es lo importante que debe quedar escrito y cómo llevar el control de cada una. A partir de un estudio realizado de los elementos más importantes recomendados por varios autores (40) (41) (42) 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.

A continuación se muestran las principales Historias de usuario, las restantes se encuentran en el anexo1.

Tabla 2.1. HU-1. Construir vector de término

Historia de Usuario					
Numero de Historia:	HU-1	Usuario:	Programadores	Fecha:	18/01/2018
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 del documento.				
Observaciones	El texto de entrada debe estar previamente filtrado por Stop Filter, Spanish Character Filter y Stem Filter.				
Puntos Estimados:	3		Puntos Reales: 3		3

Tabla 2.2. HU-3. Construir diccionario de términos

Historia de Usuario					
Numero de Historia:	HU-3	Usuario:	Programadores	Fecha:	19/01/2018
Título:	Construir el diccionario de términos.				
Descripción:	El componente debe construir el diccionario de términos asociado a la colección de entrada.				
Observaciones:	Para la construcción del diccionario debe haber culminado el proceso de construcción de todos los vectores de la colección.				
Puntos Estimados:	3		Puntos Reales: 3		3

Tabla 2.3. HU-5. Construir matriz de semejanza

Historia de Usuario					
Numero de Historia:	HU-5	Usuario:	Programadores	Fecha:	17/01/2018
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	3		

2.1.3 Matriz de trazabilidad

La matriz de trazabilidad de requisitos del proyecto es una herramienta que permite vincular los requisitos del producto, desde su concepción, hasta los entregables de Proyecto que los satisfacen (43).

En esta investigación se realizaron 2 tipos de matrices de trazabilidad: Matriz de trazabilidad RF-RF y Matriz de trazabilidad RF-HU.

La figura 2.3 muestra la matriz de trazabilidad RF-RF. La matriz de RF-HU puede ser consultada en el anexo 1.

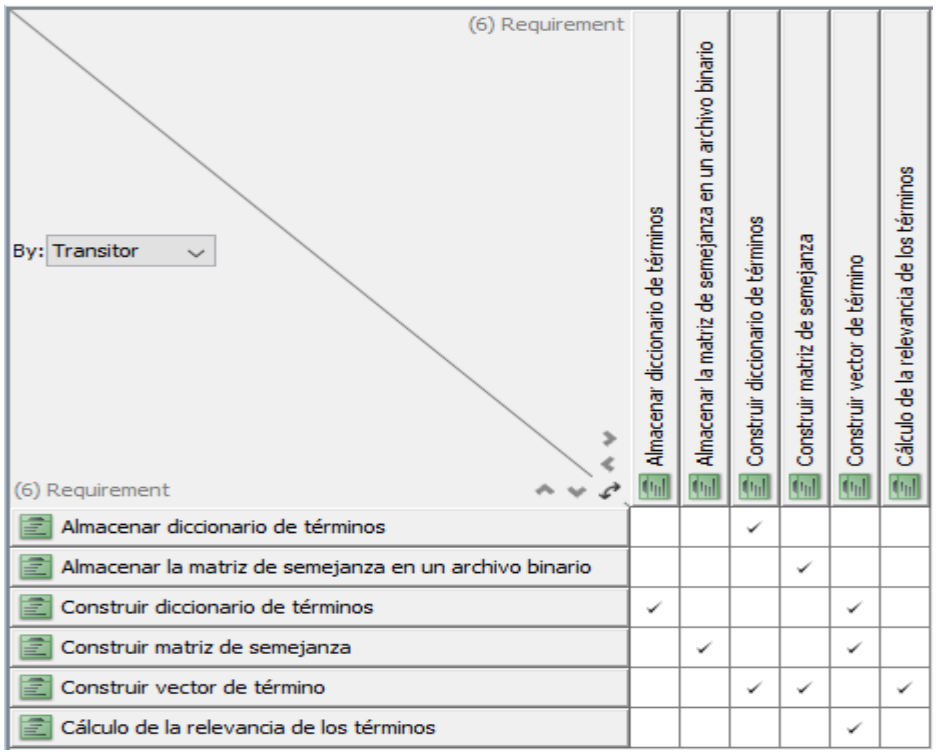


Figura 2.3. Matriz de trazabilidad RF-RF

La figura 2.3 muestra las relaciones que existe entra cada uno de los requisitos funcionales del componente.

2.1.4 Tiempo de ejecución del proyecto

El tiempo de ejecución del proyecto es un tipo de medida en la que se basa un equipo de desarrollo para determinar el tiempo necesario 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. 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	Construir vector de términos.	3
HU-2	Relevancia de cada término.	3
HU-3	Construir diccionario de términos.	3
HU-4	Almacenar diccionario de términos.	3
HU-5	Construir matriz de semejanza.	2
HU-6	Almacenar matriz de semejanza en un archivo binario.	3

2.1.5 Iteraciones

En la metodología seleccionada AUP-UCI, el desarrollo del software se divide en etapas para facilitar su realización. Por 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 (29). 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	Construir vector de términos, Calculo de relevancia de los términos, Construir diccionario de términos, Construir matriz de semejanza.	12
2	Almacenar diccionario de términos.	2
3	Almacenar la matriz de semejanza en un archivo binario.	2
Total		16

El plan 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 consecuente con la iteración contigua. Como se observa en la figura del modelo de la

propuesta de solución cuenta con 3 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.

2.1.6 Plan de entrega

En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. 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 (29). 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 2.9 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 2018
Iteración 2	24 de febrero del 2018
Iteración 3	20 de abril del 2018

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 contar con los documentos estandarizados y filtrados. 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

Luego que se analizan y especifican los requisitos del software, esta es la primera de las tres actividades técnicas que se requieren para la construir y verificar el software (44). 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 del componente a desarrollar.

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

La Arquitectura de software es una disciplina de especial relevancia en la Ingeniería de Software que constituye un aspecto de vital importancia en el momento del diseño de la estructura del sistema o componente a desarrollar. La arquitectura de software es la representación de alto nivel de la estructura del sistema o aplicación, que describe las partes que la integran, las interacciones entre ellas, los patrones que supervisan su composición, y las restricciones a la hora de aplicar esos patrones (45). Por lo cual implementar una arquitectura de software que represente adecuadamente el componente a desarrollar sirve como guía al proceso de desarrollo del 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 (36).

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 2.4 se muestra la arquitectura de la propuesta de solución.

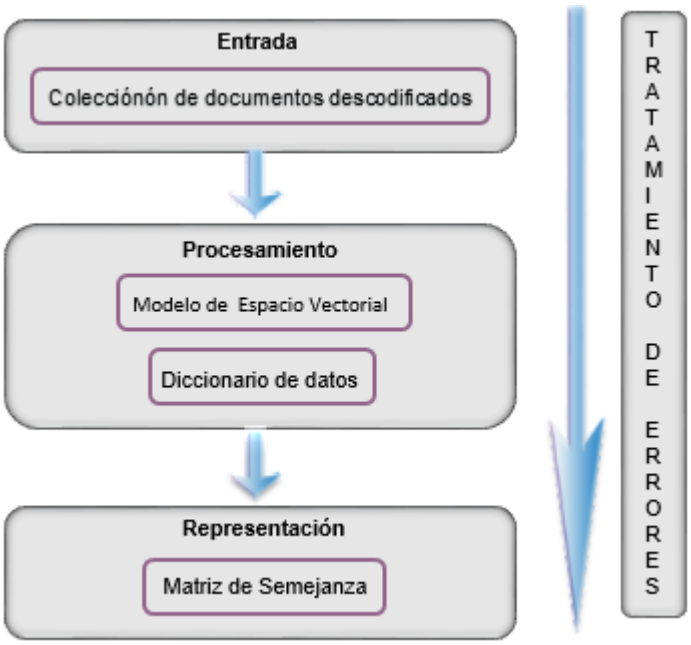


Figura 2.4. Arquitectura de la propuesta de solución

Como se muestra en la figura 2.4, el diseño está dividido en tres capas: entrada, procesamiento y representación. En la primera capa se entran la colección de documentos ya codificados de acuerdo a los estándares permitidos y luego estos son enviados a la capa intermedia. La capa de procesamiento es la responsable de las operaciones de la aplicación, construir los vectores de términos, construir el diccionario y de conformar 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 (46). 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 2.5 se muestra 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.

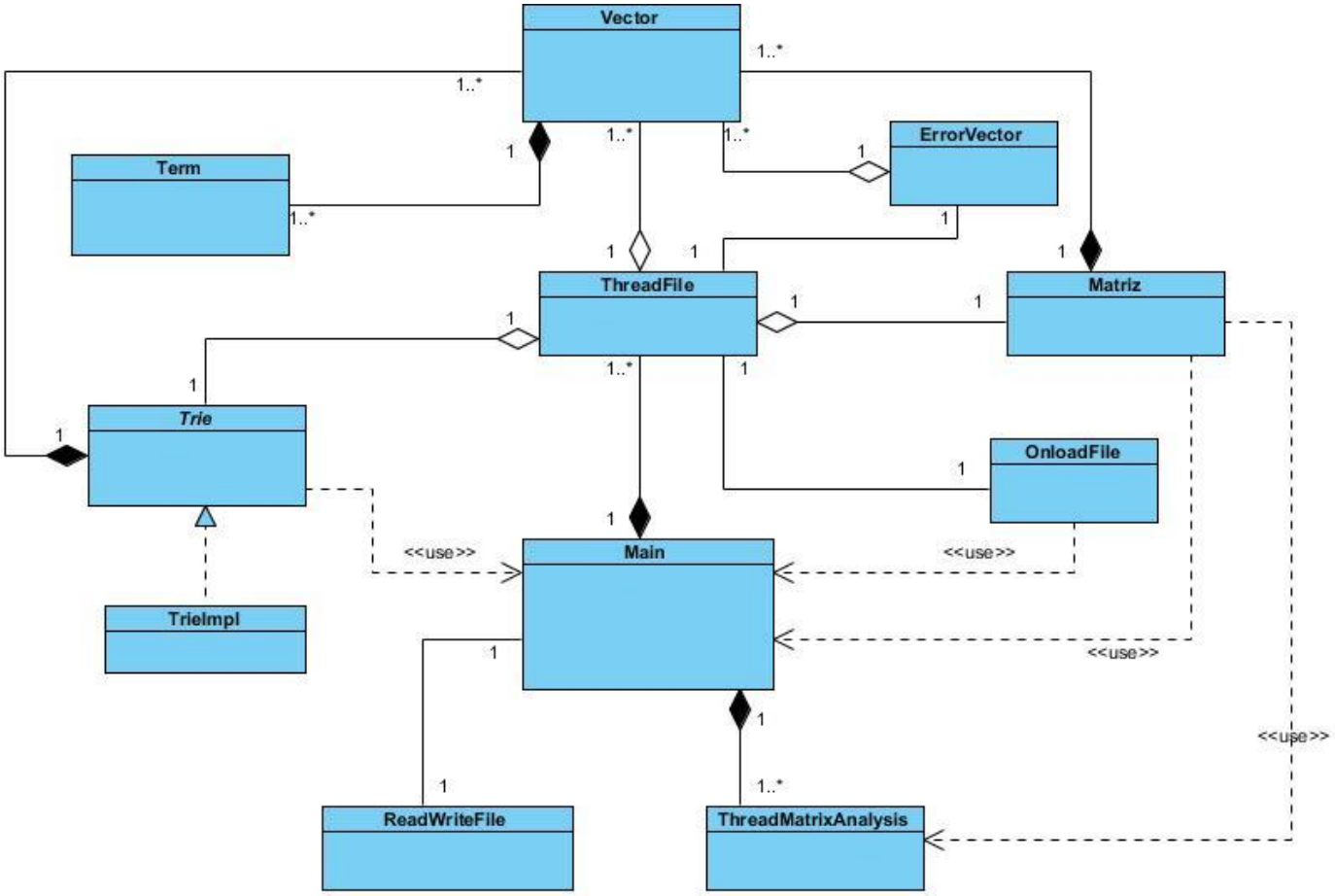


Figura 2.5. Diagrama de clases UML

El sistema a implementar cuenta con una jerarquía de clases asociada a las tareas necesarias para la representación matemática 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 se relaciona con la clase ReadWriteFile, posee una asociación por composición con las clases ThreadFile y ThreadMatrixAnalysis, así como relaciones de uso con las clases Trie, Matriz y OnloadFile, esta última se encarga de establecer el fichero con todos los documentos a procesar.

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 de la colección de entrada, la misma se relaciona con OnloadFile y ErrorVector, y posee asociaciones por agregación con las clases Trie, Matriz y Vector. Para la colección de entrada es necesario construir un diccionario de términos,

de lo cual se responsabiliza la clase interfaz Trie, que es implementada por la clase TrieImpl y posee una asociación por composición con la clase Vector.

Se muestra también la relaciona de la clase Matriz con Vector mediante una asociación por composición y contiene la matriz de todos los términos de la colección de documentos. Vector es la clase encargada de construir el vector para cada documento de la colección de entrada y tiene además una asociación por composición con Term, que contiene la información de un término específico. 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. El anexo número 2 muestra 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 describen un problema que ocurre reiteradas veces y el núcleo de la solución al problema, de forma que pueden utilizarse en ilimitadas ocasiones sin tener que hacer dos veces lo mismo. Un patrón de diseño no es más que una solución estándar para un problema común de programación. Una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios (36). Por lo que constituyen una guía para el rápido desarrollo de software (47). 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 (36). 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 (36). 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 2.6 muestra el uso de este patrón.

```
public class Term implements Serializable {

    private int tf;
    private String stem;
    private BigDecimal weigh;


    public String getStem() {...3 lines }

    public BigDecimal getWeigh() {...3 lines }

    public void calcWeigh(int ni, int D) {

        BigDecimal n = new BigDecimal(D).divide(new BigDecimal(ni), 4, RoundingMode.UP);
        BigDecimal LogN = BigDecimalMath.log(n);
        BigDecimal LogBase = BigDecimalMath.log(new BigDecimal(10));
        BigDecimal Log = LogN.divide(LogBase, 4, RoundingMode.UP);

        BigDecimal Wij = Log.multiply(new BigDecimal(tf));

        weigh = Wij;

    }
```

Figura 2.6. Implementación del patrón Experto

Como se observa en la figura 2.6 la clase Term es la encargada de formar los términos con sus pesos respectivamente a partir de la colección de entrada. Esta cuenta con toda la información necesaria para la implementación del método calcWeigh () encargado de calcular el peso de cada término de la colección.

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 (36). 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.

```
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 2.7. Implementación del patrón Creador

Como se observa en la figura 2.7 la clase Vector es la encargada de la creación de una lista de los objetos tipo términos mediante una asociación por agregación con la clase Term.

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 (36). Este patrón permite la obtención de clases que pueden mantenerse fácilmente y ser reutilizables durante el proceso de desarrollo de software. La figura 2.8 muestra el uso del mismo

```
public class Matriz implements Serializable {

    private Map<String, Vector> map;
    private static Matriz matriz;

    public Matriz() {
    }

public static Matriz getMatriz() {
     if (matriz == null) {
     matriz = new Matriz();
     }
     return matriz;
     }
```

Figura 2.8. Implementación del patrón Alta Cohesión

Como se observa en la figura 2.8 la clase Matriz es la encargada de crear el objeto matriz, lo que garantiza la alta cohesión e incrementa la claridad, reutilización y la facilidad de comprensión del diseño.

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 (36). 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 acrecienten la oportunidad de una mayor productividad. El ejemplo analizado en la figura 2.7 soporta tanto el patrón creador 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 (48). La valía de los patrones de creación ilustra como estructurar y encapsular estas decisiones.

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 (48). 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.

```
public class Matriz implements Serializable {

    private Map<String, Vector> map;
    private static Matriz matriz;

    private Matriz() {
    }

public static Matriz getMatriz() {
     if (matriz == null) {
     matriz = new Matriz();
     }
     return matriz;
     }
```

Figura 2.9. Implementación del patrón Singleton

Como se observa en la figura 2.9 en la clase Matriz se crea el constructor de forma privada y se implementa el método publico getMatriz, el cual en caso en que no esté creado, crea el objeto matriz, si ya existe una instancia del objeto lo devuelve, esto permite que durante la ejecución del programa se compile con una sola instancia del objeto.

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 (48).

- 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 (48).

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 analizará el patrón Scheduler, la aplicación del mismo se describe en la sección 2.4.2.

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 (48).

2.3 Implementación de la propuesta de Solución

En esta sección se describe la 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 almacenamiento de términos. 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 2.7 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 previamente filtrados y codificados que contengan texto plano escrito en español. Dicha colección constituye la entrada al algoritmo propuesto.

Tabla 2.7. Pseudocódigo. Algoritmo para la representación de corpus textual

Algoritmo para la representación de corpus textual	
1	Inicio
2	D ← **Entrada**(Colección de documentos)
3	V ← {Vector del documento, inicia en vacío}
4	Q ← {Diccionario de términos, inicia en vacío}
5	M ← {Matriz de Semejanza, inicia en vacío}
6	Para cada T de D hacer
7	Si T pertenece V
8	entonces **Recalcular**(T, V)
9	Sino V ← **Vector**(T)
10	Si T pertenece Q
11	entonces **Recalcular**(T, Q)
12	Sino Q ← **Dictionary**(T)
13	M ← **Matriz**(V)
14	M ← **Weigh**(M)
15	C ← {Vector consulta, inicia en vacío }
16	Para cada C de M hacer
17	Para cada V de M
18	Si V distinto C
19	entonces **Semejanza**(C,V)
20	**Salida**(M)
21	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 D recibe la colección de documentos de entrada filtrados y codificados a procesar. En la línea 3, 4 y 5 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 6 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 10 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 13 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).** Luego en la línea 15 se crea la estructura de datos que almacena el resultado de una consulta a la matriz de semejanza C. A partir de la línea 16 termina la construcción de la matriz de semejanza y puede ocurrir una de dos opciones, para cada C de M y para cada V de M:

a) Si V distinto de C, entonces **Semejanza**(C, V)

b) Sino **Salida** (M) y termina la ejecución del programa.

La tabla 2.8 muestra el pseudocódigo del procedimiento para la actualización de los pesos de los términos de los vectores en la matriz de semejanza.

Tabla 2.8. 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
10	Fin

En la línea 3, se representan 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 ← **Frecuencia**(T) es la frecuencia de aparición del término T en el documento y NI ← **Frecuencia**(T,D) es la frecuencia de aparición del término T en la colección. Cada peso se calcula como la frecuencia de aparición de T en el documento por la inversa de la cantidad de documentos de la colección donde aparece el término.

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 (49). 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 (49).

Para el control de la concurrencia se emplean el patrón: Scheduler. La figura 2.9 muestra su implementación.

```
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 trie (por su nombre en inglés) que crea un árbol de prefijos. Su propósito es llevar el recuento de la cantidad de apariciones de cada término, así como construir el vocabulario de las palabras de la colección. Para ello al insertar un término se busca primero el mayor prefijo de esa palabra que ya esté presente en el diccionario y a partir de ahí se continua insertando hasta donde termine el término; en el subepígrafe 2.3.3 se describe esta estructura. El objetivo de esta técnica es mejorar la capacidad de respuesta al realizar la búsqueda, al no realizar una búsqueda exhaustiva para reconocer si un término existe o no en el diccionario.

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 (16).

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] (16) (50).

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 (16):

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

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 de la sumatoria de los pesos P de los documentos al cuadrado, donde n es un término del diccionario de datos. 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.

2.3.3 Corrida del componente implementado

La implementación de un trie como diccionario de términos de una colección de documentos mejora el tiempo de búsqueda de un término específico, siendo una constante determinada por la cantidad de caracteres de la consulta. Lo expuesto se demuestra dada una colección de términos, verificar si uno en específico se encuentra en la colección. Si se utiliza una estructura de datos lineal el tiempo de búsqueda de cada término es O(n) ya que necesita al menos n comparaciones, donde n es la cantidad de términos del corpus. Cuando crece la cantidad de término esta solución no es viable porque implica un costo computacional muy alto.

Otra solución es ordenar todos los términos alfabéticamente y luego usar búsqueda binaria para encontrar el término consulta. Esta solución introduce una complejidad temporal de O(n logn).

El trie mejora la complejidad en O (|s|) donde |s| es la cantidad de caracteres de la palabra s (51).

La figura 2.11 muestra las ventajas en las corridas del componente implementando utilizando el diccionario indexado (trie) para un total de 700 000 palabras.

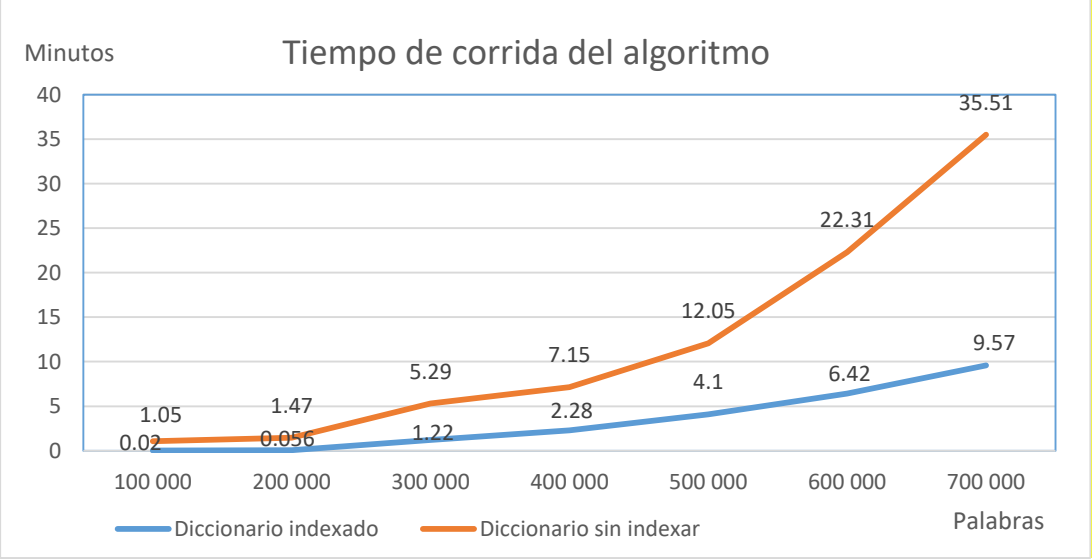


Figura 2.11. Tiempo de corrida del algoritmo

Como se muestra en la figura 2.10 para cada par de corridas del componente 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.

2.3.4 Ejemplo del algoritmo que implementa el Modelo Espacio Vectorial

El corpus consta de 5 documentos acerca de animales y una consulta

Doc1:{Perro, Elefante, Caiman}

Doc2:{Perro, Perro, Sapo}

Doc3:{ Sapo, Elefante, Caiman}

Doc4:{ Sapo, Mono, Caiman}

Doc5:{ Abeja, Caiman, Caiman}

Cons1:{Perro, Sapo, Abeja, Abeja}

Como paso inicial se obtiene la matriz de frecuencias documentos-términos, que constituye las repeticiones de cada término en los respectivos documentos en que se encuentra. La tabla 2.9 muestra la relación documento-término.

Tabla 2.9. Documentos-Términos

	Mono	**Abeja**	**Elefante**	**Caiman**	**Perro**	**Sapo**
Doc1			1	1	1	
Doc2					2	1
Doc3			1	1		1
Doc4	1			1		1
Doc5		1		2		

Luego se obtiene la tabla de IDF (peso del término respecto al corpus)

IDF=$\log_2 \frac{N}{n}$

N se corresponde con la cantidad de documentos del corpus.

n se corresponde con la cantidad de documentos en que el término aparece.

La columna Frecuencia documentos (df) representa las n veces que se repite el término en correspondencia con el corpus. La tabla 2.10 muestra la frecuencia del término en el corpus.

Tabla 2.10 . Frecuencia del término en el corpus

Término	Frecuencia documentos (df)	IDF (N=5)
Mono	1	2.32
Abeja	1	2.32
Elefante	2	1.32
Caiman	4	0.32
Perro	2	1.32
Sapo	3	0.74

Finalmente se calculan los pesos en cada documento mediante el producto TF*IDF, combinando las tablas anteriores.

TF constituye la cantidad de repeticiones del término en el documento. La tabla 2.11 muestra la relación TF*IDF.

Tabla 2.11. TF*IDF

	Mono	**Abeja**	**Elefante**	**Caiman**	**Perro**	**Sapo**	**Norma***
Doc1			1.32	0.32	1.32		1.89
Doc2					2.64	0.74	2.74
Doc3			1.32	0.32		0.74	1.55
Doc4	2.32			0.32		0.74	2.46
Doc5		2.32		0.64			2.41

En esta tabla se agregó la columna norma, que corresponde a la longitud de los vectores y permite normalizar. Ejemplo

Norma (Doc1)=$\sqrt{(1.32)^2 + (0.32)^2 + (1.32)^2}$=1.89

La tabla 2.12 muestra la consulta suministrada por el usuario, que incluye términos y frecuencias, se computan sus pesos.

Tabla 2.12. Consulta, términos y frecuencia

	Mono	Abeja	Elefante	Caiman	Perro	Sapo	Norma
Cons1 Frecuencias	0	2	0	0	1	1	-
Cons1 Ponderada	0	4.64	0	0	1.32	0.74	4.88

A continuación, se calculan las semejanzas existentes entre la consulta y los cinco documentos

$$\text{SIM (Doc1, Cons1)} = \frac{(0*0)+(0*4.64)+(1.32*0)+(0.32*0)+(1.32*1.32)+(0*0.74)}{1.89*4.88} = 0.18$$

$$\text{SIM (Doc2, Cons1)} = \frac{(0*0)+(0*4.64)+(0*0)+(0*0)+(2.64*1.32)+(074*0.74)}{2.74*4.88} = 0.30$$

$$\text{SIM (Doc3, Cons1)} = \frac{(0*0)+(0*4.64)+(1.32*0)+(0.32*0)+(0*1.32)+(0.74*0.74)}{1.55*4.88} = 0.07$$

$$\text{SIM (Doc4, Cons1)} = \frac{(2.32*0)+(0*4.64)+(0*0)+(0.32*0)+(0*1.32)+(0.74*0.74)}{2.46*4.88} = 0.04$$

$$\text{SIM (Doc5, Cons1)} = \frac{(0*0)+(2.32*4.64)+(0*0)+(0.64*0)+(0*1.32)+(0*0.74)}{2.41*4.88} = 0.91$$

En la tabla 2.13 se ordenan las respuestas por similitud a la consulta 1.

Tabla 2.13. Similitud de los documentos

Similitud (d, q)	0.91	0.30	0.18	0.07	0.04
Documento	Doc5	Doc2	Doc1	Doc3	Doc4
Orden de Relevancia	1	2	3	4	5

Del análisis de la tabla anterior surge que el documento 5 (Doc5) es el más similar a la consulta 1 (Cons1).

Conclusiones del capítulo

Una vez realizada la propuesta de solución de la investigación se arribó a las siguientes conclusiones:

- La definición del modelo de dominio, facilitó el entendimiento del negocio a informatizar.
- La definición de las historias de usuario permitió identificar las funcionalidades a desarrollar en la solución desarrollada.
- La arquitectura de software empleada y los patrones de diseño sentaron las bases para la siguiente etapa de construcción del software.

Capítulo 3. Comprobación del funcionamiento de la propuesta de solución

En el presente capítulo se desarrolla lo referente a la fase de Implementación y Pruebas, los cuales son determinantes en el proceso de desarrollo de software. Se definen los estándares de codificación utilizados, así como las pruebas realizadas: pruebas unitarias y de aceptación, encargadas de comprobar el funcionamiento del software para garantizar el correcto funcionamiento del producto final.

3.1 Comprobación del funcionamiento del componente desarrollado

Uno de los pilares de la metodología AUP-UCI es el proceso de pruebas (32). 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.

Las pruebas a realizar se dividen en dos grupos: pruebas unitarias, encargadas de verificar el código y diseñadas por los programadores y las pruebas de aceptación destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida por el cliente (52). 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 (53); 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. 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 (54). 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.

Las pruebas de caja negra abarcan todo lo referente a la interfaz del software. Los casos de pruebas 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 la integridad de la información externa se mantiene. Por otro lado las pruebas de caja blanca se enmarcan en la comprobación de 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.

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 puede invocar al código que queremos probar y determina si el resultado obtenido es el esperado. Por lo general son simples y rápidas de codificar, buscan aislar cada parte del programa y mostrar que las partes individuales son correctas, facilitan la reestructuración del código, separa la interfaz y la implementación y permiten llegar a la fase de integración asegurando que las partes individuales funcionan correctamente (55). 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 la automatización de las pruebas unitarias, que permite realizar la ejecución de clases Java de manera controlada en la fase de desarrollo, para evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. El framework provee al usuario de herramientas, clases y métodos que le facilitan la tarea de realizar pruebas en su sistema y así asegurar su consistencia y funcionalidad (56)**.** 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 3.

Pruebas unitarias. Iteración 1

La primera iteración de pruebas unitarias fue realizada sobre las historias de usuarios siguientes: HU-1 Construir vector de término y HU-2 Cálculo de la relevancia de los términos.

La tabla 3.1 y 3.2 muestran el resultado de la ejecución de las pruebas realizadas a las clases Vector y Term generadas para la resolución de las historias de usuarios HU-1 Construir vector de término y HU-2 Cálculo de relevancia de los términos. En ellas se muestra 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 3.1. HU-1 Construir vector de término. Clase Vector

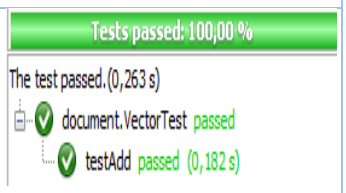
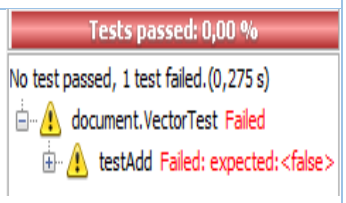
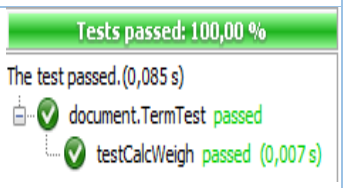
Pruebas unitarias					
Método a probar	**Entrada**	**Resultado esperado**	**Resultado obtenido**	**Funciona correcta mente**	**Observaciones**
add()	Recibe la raíz semántica de una palabra.	Debe añadir un nuevo término al vector y retornar el valor booleano true.	El término fue añadido correctamente al vector y se obtuvo el valor booleano true.	Si	
	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 true.	La frecuencia del término no fue actualizada. En su lugar el mismo fue adicionado nuevamente y se obtuvo el valor booleano false.	No	

Tabla 3.2. HU-2 Cálculo de la relevancia de los términos. Clase Term

Pruebas unitarias					
Método a probar	**Entrada**	**Resultado esperado**	**Resultado obtenido**	**Funciona correcta mente**	**Observaciones**
calcWeigh()	Recibe la cantidad de repeticiones del término en el documento y la inversa de la cantidad de documentos de la colección donde aparece el término.	Debe calcular la importancia del término en el documento y retornar el valor para cada uno de los términos del documento.	LA relevancia de los términos fue calculada correctamente retornando el valor numérico.	Si	

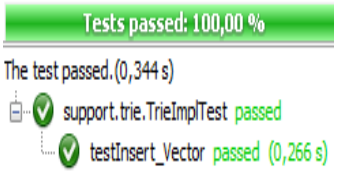
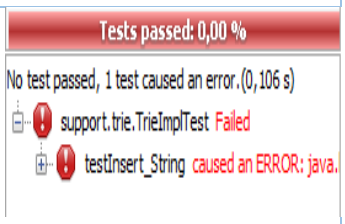
Durante la primera iteración se detectaron errores de codificación en el método add de la clase Vector al verificar la existencia del término que pretende adicionar al vector, evitando de esta forma valores duplicados. En este caso el término fue añadido al vector, en lugar de actualizar su frecuencia como se esperaba. 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 realiza sobre las historias de usuario siguientes: HU-3 Construir diccionario.

La tabla 3.3 muestra el resultado de la ejecución de las pruebas realizadas a la clase TrieImpl las cuales fueron generadas para la resolución de la historia de usuario HU-3 Construir diccionario de términos.

Tabla 3.3. HU-3 Construir diccionario de términos. Clase TrieImpl

Pruebas Unitarias					
Método a probar	**Entrada**	**Resultado esperado**	**Resultado obtenido**	**Funciona correcta mente**	**Observaciones**
Insert_Vector()	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.	Si	
Insert_String()	Se recibe un término extraído del vector.	Cada término extraído del vector debe ser añadido al diccionario o actualizada su frecuencia de aparición.	Los términos del vector no fueron correctamente añadidos o actualizados en el diccionario	No	

En la segunda iteración se detectaron errores de codificación en el método Insert_String () de la clase TrieImpl, al verificar que el término extraído del vector no fue añadido o actualizado su frecuencia correctamente. 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-4 Almacenar diccionario de términos y HU-6 Almacenar la matriz de semejanza en un archivo binario.

Las tablas 3.4 y 3.5 muestran el resultado de la ejecución de las pruebas realizadas a la clase ReadWriteFile la cual fue generada para solucionar las historias de usuarios HU-4 Almacenar diccionario de términos y HU-6 Almacenar la matriz de semejanza en un archivo binario. Las mismas mantienen los mismos campos que las tablas anteriores.

Tabla 3.4. HU-4 Almacenar diccionario de términos. Clase ReadWriteFile

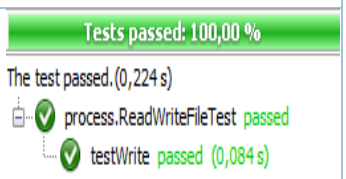
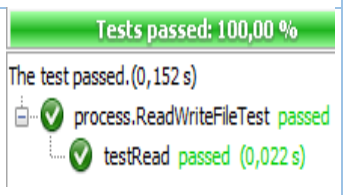
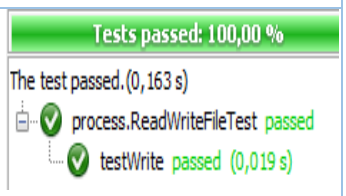
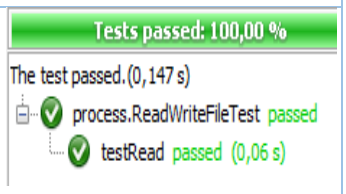
Pruebas Unitarias					
Método a probar	**Entrada**	**Resultado esperado**	**Resultado obtenido**	**Funciona correcta mente**	**Observaciones**
Write()	Recibe como datos el objeto a guardar (para este caso es el objeto diccionario) y la dirección del lugar donde se va a almacenar.	Debe crear un fichero en la dirección indicada que contenga el objeto de entrada.	El fichero fue creado correctamente en la dirección indicada.	Si	
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	

Tabla 3.5. HU-6 Almacenar la matriz de semejanza en un archivo binario. Clase ReadWriteFile

Pruebas Unitarias					
Método a probar	**Entrada**	**Resultado esperado**	**Resultado obtenido**	**Funciona correcta mente**	**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	
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	

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 3.1 durante el proceso de pruebas unitarias se verificaron un total de 6 operaciones del componente implementado. Durante la primera iteración se encontraron errores en una de las dos operaciones comprobadas y en la segunda de dos operaciones comprobadas una presento

problemas. En la tercera iteración se comprobaron dos 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.

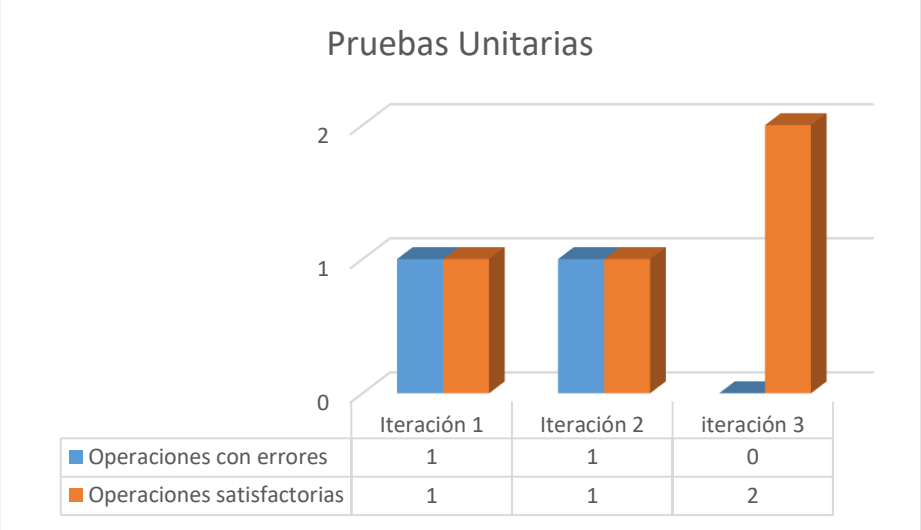


Figura 3.1. Pruebas unitarias

Pruebas de aceptación

Las pruebas de aceptación son una parte integral del desarrollo incremental definido por la metodología AUP-UCI. 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 (57). Las mismas obligan al cliente a un profundo conocimiento del estado de la aplicación y lo que debe hacer en circunstancias específicas (52).

En el proceso de desarrollo de software de las pruebas de aceptación se divide en iteraciones, que están asociadas cada historia de usuario. Para ello se realiza un modelo que involucra: acción a probar, datos de pruebas, resultado esperado, resultado obtenido y evaluación de la prueba.

Pruebas de aceptación. Iteración 1

La tabla 3.6 muestra los datos obtenidos a partir de la aplicación de las pruebas de aceptación a las historias de usuario HU-1, HU-3 y HU-5 (por su número de historia).

Tabla 3.6. Pruebas de aceptación. Iteración 1

Pruebas de Aceptación				
Acción a Probar	**Datos de Prueba**	**Resultado Esperado**	**Resultado Observado**	**Pasa la prueba**
HU-1: cada documento de la colección debe ser presentado en un vector de términos donde cada componente del vector constituye el lexema de la palabra correspondiente en la colección.	La prueba da inicio con 2000 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 1 minutos.	**No conformidad:** Tiempo de ejecución demasiado alto. El vector solo muestra los términos, se necesita manejar más información.
HU-3: el motor de representación debe construir un diccionario de términos con el vocabulario de la colección	La prueba da inicio con 2000 documentos	El diccionario debe construirse a partir de la colección de entrada y si se encuentra el término actualizar su frecuencia.	El diccionario fue construido correctamente pero la frecuencia de los términos no fue actualizada.	**No conformidad:** No se actualizo correctamente la frecuencia de los términos en el diccionario.

HU-5: el componente debe construir la matriz de términos a partir de los vectores de términos y actualizar los pesos asociados a cada término del vector.	La prueba da inicio con 2000 documentos	Los pesos asociados a cada término deben estar debidamente calculados.	El peso de los términos no fue actualizado correctamente y el tiempo de representación demoro un total de 3 minutos.	**No conformidad:** Los pesos de los términos no fueron actualizados y el tiempo de ejecución demasiado alto. **Pedido de cambio:** Matriz de semejanza.
HU-5: el componente debe realizar la representación textual de los documentos en un tiempo prudencial.	La prueba da inicio con 500 documentos con dimensión máxima de 4000 palabras, para un total de 736 336 palabras.	El tiempo estimado de respuesta debe ser como máximo 12 minutos.	El componente demoró en procesar los documentos un total de 35 minutos.	**No conformidad:** Tiempo de ejecución demasiado alto.

En la primera iteración se detectaron cuatro no conformidades asociadas a las historias de usuarios: HU-1, HU-3, y HU-5 respectivamente, además de un pedido de cambio en la HU-5. 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 como un árbol de prefijos (Trie) que al insertar una palabra se busca primero el mayor prefijo de esa palabra que ya esté presente y a partir de ahí se continua insertando hasta donde termine la palabra 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 2

La tabla 3.7 muestra el resultado de la ejecución de las pruebas de aceptación a las historias de usuario HU-4 Almacenar diccionario de términos y HU-6 Almacenar la matriz de semejanza en un archivo binario.

Tabla 3.7. Pruebas de aceptación. Iteración 2

Pruebas de Aceptación				
Acción a Probar	**Datos de Prueba**	**Resultado Esperado**	**Resultado Observado**	**Pasa la prueba**
HU-4, HU-6: 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 segunda y última iteración pasa satisfactoriamente la prueba de aceptación y culmina el proceso de desarrollo del software. Como se muestra en la figura 3.2 el proceso de pruebas de aceptación se realizó en dos iteraciones. Durante la primera iteración se obtuvo cuatro no conformidades y un pedido de cambio que fueron corregidas e incluido de forma correcta. Luego se realizó la segunda iteración de forma satisfactoria. En total se detectaron cuatro no conformidades y un pedido de cambio durante el proceso de pruebas.

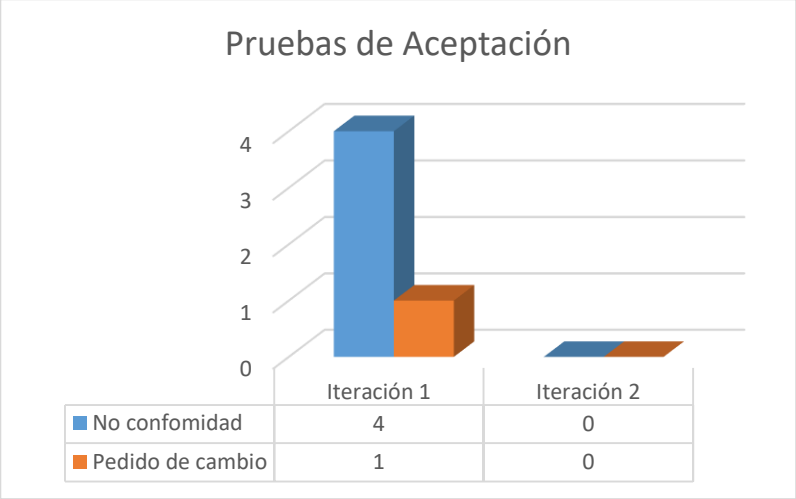


Figura 3.2. Pruebas de aceptación

Conclusiones del capítulo

En este capítulo 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, permitiendo comprobar cada uno de los métodos involucrados en la ejecución del componente desarrollado. Luego las iteraciones de las pruebas de caja negra permitieron garantizar el correcto funcionamiento del componente, corrigiendo las no conformidades encontradas. 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 la representación matemática de 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 matemática de archivos digitales textuales.

El análisis realizado en el epígrafe Estudio de los antecedentes de la investigación, permitió conocer herramientas existentes que utilizan modelos de recuperación de información y se basan en la representación del corpus textual de archivos digitales. Los resultados del análisis determinaron que ninguna de las herramientas analizadas constituía 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.

El análisis y diseño realizado de la solución propuesta permitió obtener un mejor entendimiento y organización del funcionamiento del componente, que ayudó a su vez a una mayor comprensión de las tareas a desarrollar por el equipo de desarrollo.

Fue comprobado el funcionamiento del componente apoyado en los niveles de pruebas: pruebas unitarias y pruebas de aceptación. Estas pruebas permitieron detectar y corregir errores de codificación en las funcionalidades durante la implementación, contribuyendo así al desarrollo de un componente más robusto y menos propenso al mal funcionamiento.

Por lo expuesto anteriormente, se puede concluir que se desarrolló satisfactoriamente un componente informático que permite la representación matemática del corpus textual de una colección de archivos digitales textuales contribuyendo de este modo a la recuperación de información.

RECOMENDACIONES

A partir de las conclusiones abordadas se listan las recomendaciones en vistas de posibles mejoras:

Proporcionar al componente desarrollado una salida que muestre la semejanza punto a punto de los documentos procesados.

Agregar nuevas medidas de semejanza al sistema para mejorar los criterios comparativos entre los documentos.

En dependencia de las propiedades de hardware existentes y en correspondencia con el número de documentos a procesar dividirlos en lotes, ya sea de 500, de 1000, entre otras cantidades a procesar.

REFERENCIAS

1. **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.

2. **CHIAVENATO, I.** *Introducción a la Teoría General de la Administración.* Septima. 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. **Marín Agudelo, S. A.** *Estado de la archivística en América Latina 2000-2009: Perspectivas teóricas y aproximaciones conceptuales.* (2012).

5. **DUQUE, D.** Los Archivos Digitales. *Los archivos y algo más.* [En línea] 2009. [Citado el: 7 de Diciembre de 2017.] http://doraduque.wordpress.com/tag/que-es-un-archivo-digital/.

6. *Revista Cubana de Ciencias Informáticas versión On-line ISSN 2227-1899 Rev cuba cienc informat.* **Grossman and Frieder, 2012, Croft.** no.2, La Habana : s.n., 2016, Vol. 10.

7. **ZAZO RODRÍGUEZ, A. F., FIGUEROA PIANAGUA, C. G. y ALONSO BERROCAL, J. L.** *Recuperación de información utilizando el modelo vectorial.* Salamanca : s.n., 2002.

8. **SECO NAVEIRAS, D.** *Técnicas de indexación y recuperación de documentos utilizando referencias geográficas y textuales.* 2009.

9. **YERBABUENA.** DOKUMENTALISTAS. *Yerbabuena Software, innovacion tecnologica en gestion documental.* [En línea] 3 de 12 de 2012. [Citado el: 8 de Diciembre de 2017.] http://www.dokumentalistas.com/articulos/yerbabuena-software-innovacion-tecnologica-en-gestion-documental/.

10. **LILIANA, M.** La estructura del texto. *Estructuras Textuales.* [En línea] 19 de abril de 2010. [Citado el: 15 de Diciembre de 2017.] http://comunicacionlinguisticai.blogspot.com/p/la-estructura-del-texto.html.

11. **CHOWDHURY, Gobinda G.** *Introduction to modern information retrieval.* ilustrada. Londres : Library Association, 1999.

12. **UCI.** *gespro. Suite de Gestión de Proyectos.* [En línea] [Citado el: 1 de Diciembre de 2017.] http://gespro.ciged.prod.uci.cu.

13. **ORTEGA MENDOZA, R. M.** *Descubrimiento Automático de Hipónimos a partir de textos no estructurados.* [Documento] Tonantzintla, Puebla : s.n., 2007.

14. *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.

15. *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.

16. **BAEZA YATES, R. y RIBEIRO NETO, B.** *Modern Information Retrieval.* New York : ACM Press, 1999.

17. **KORFHAGE, R. R.** *Information Storage and Retrieval.* New York : Wiley Computer Publishing, 1997.

18. **AHO, A. V., SETHI, R. y ULLMAN, J. D.** *Compiladores: Principios, técnicas y herramientas.* s.l. : Pearson Educación, 1998. ISBN 9684443331.

19. **FRANCIS, W. y KUCERA, H.** Frequency analysis of english usage. *Lexicon and Grammar.* Boston : Houghton Mifflin, 1982.

20. **Martínez Méndez, F. J.** *Recuperación de información: modelos, sistemas y evaluación.* Murcia: Kiosko : s.n., 2004.

21. **SALTON, G., [ed.].** *The SMART Retrieval System - Experiments in Automatic Document Processing.* s.l. : Prentice Hall In, 1971.

22. **SÁNCHEZ, Maité Torres y MARTÍNEZ, Lourdes Garrido.** *Modelo de Recuperación de Información Fuzzy.*

23. **RESEARCH, P.** *Una herramienta avanzada para el análisis de contenido y minería de texto.* [En línea] [Citado el: 7 de Enero de 2018.] https://www.software-shop.com/producto/wordstat.

24. **IMAGE.** Alceste. A textual statistics software. [En línea] 2012. [Citado el: 7 de Enero de 2018.] http://www.image-zafar.com/sites/default/files/telechargements/alceste2012plusen.pdf.

25. **BICQUELET, A.** *Introduction to Alceste.* [Documento] London : LSE, Methodology Institute, 2012.

26. **FRANCO, L.** Herramientas para el Análisis de Textos. [En línea] [Citado el: 8 de Enero de 2018.] http://tlab.it/es/presentation.php.

27. **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: 9 de

Enero de 2018.] http://pepsic.bvsalud.org/scielo.php?script=sci_arttext&pid=S1413-389X2013000200016. ISSN 1413-389X.

28. **APACHE.** Lucene. [En línea] [Citado el: 9 de Enero de 2018.] https://lucene.apache.org/core/.

29. **LETELIER, P.** Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP). [En línea] 15 de enero de 2006. [Citado el: 8 de Enero de 2018.] http://www.cyta.com.ar/ta0502/v5n2a1.htm. ISSN 1666-1680.

30. **J.C, Fernadez.** Metodologías Ágiles para el Desarrollo de Software y Metodologías Para el desarrollo Web. [En línea] [Citado el: 6 de Febrero de 2018.] https://es.slideshare.net/jofese/metodologas-giles-y-metodologias-web-apra-el-desarrollo-de-software.

31. **AMBLER, S. W.** Feature Driven Development (FDD) and Agile Modeling. *Agile Modeling.* [En línea] Ambysoft Inc, 2014. [Citado el: 9 de Enero de 2018.] http://www.agilemodeling.com/essays/fdd.htm.

32. **Tamara Rodríguez, Sánchez.** *PROGRAMA DE MEJORA: Metodología de desarrollo para la Actividad productiva de la UCI.*

33. **Sánchez, T.R.** *PROGRAMA DE MEJORA: Metodología de desarrollo para la Actividad productiva de la UCI.* Universidad de las Ciencias Informáticas. 2015.

34. **MEDIAWIKI.** Programación en Java/Características del lenguaje. *WIKILIBROS.* [En línea] 9 de abril de 2014. [Citado el: 8 de Enero de 2018.] http://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Java/Caracter%C3%ADsticas_del_lenguaje.

35. **EXES.** Curso de Introducción a Java. *El lenguaje java.* [En línea] [Citado el: 8 de Enero de 2018.] http://www.mundojava.net/caracteristicas-del-lenguaje.html?Pg=java_inicial_4_1.html.

36. **LARMAN, C.** *UML y Patrones.* New York : Prentice Hall, 1999.

37. **VÁZQUEZ ORIHUELA, A.** Visual Paradigm. *Scribd.* [En línea] [Citado el: 9 de Enero de 2018.] http://es.scribd.com/doc/166415572/Visual-Paradigm#scribd.

38. **Carolina Martínez, Ivette.** *Modelo Conceptual/Modelo de Dominio.* [Documento] Caracas : Universidad Simón Bolívar, 2010.

39. **PMO.** ¿Qué son las historias de usuario? *PMOinformatica.com.* [En línea] 24 de abril de 2013. [Citado el: 14 de febrero de 2018.] http://www.pmoinformatica.com/.

40. **A. SÁNCHEZ, E., LETELIER, P. y CANÓS, J. H.** *Mejorando la gestión de historias de usuario en eXtreme Programming.* [Documento] Valencia : s.n., 2004.

41. **VILLAMIZAR SUAZA, K.** *Definición de equivalencias entre historias de usuario y especificaciones en UN-LENCEP para el desarrollo ágil de software.* [Documento] Medellín : s.n., 2013.

42. **ECHEVERRY TOBÓN, L. M. y DELGADO CARMONA, L. E.** *Caso práctico de la metodología ágil XP al desarrollo de software.* [Documento] Pereira : s.n., 2007.

43. **Sevilla.J.** Matriz de Trazabilidad de Requisitos . [En línea] 7 de Julio de 2016. [Citado el: 7 de Abril de 2018.] http://www.overti.es/tecnologia/301-matrices-de-trazabilidad-de-requisitos.

44. **PRESSMAN, R. S.** *Ingeniería de Software. Un enfoque práctico.* Quinta. s.l. : Mc Graw Hill, 2002.

45. **L. Bass, P. y CLEMENTS, R. K.** *Software Architecture in Practice.* New York : Addison Wesley, 2003.

46. **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 2018.] http://es.scribd.com/doc/31096724/Diagrama-de-Clases-en-UML#scribd.

47. **INNOCUO.** Patrones de diseño, la importancia de aprenderlos. *innocuo.* [En línea] 2 de Diciembre de 2006. [Citado el: 15 de 12 de 2017.] http://blog.innocuo.com/2006/09/patrones-de-diseno-la-importancia-de-aprenderlos/.

48. **MARTÍNEZ JUAN, F. J.** *Guía de construcción de software en java con patrones de diseño.* Oviedo : s.n.

49. **DRAKE, J.M.** Programación Concurrente. *CTR - Computadores y Tiempo Real.* [En línea] [Citado el: 2 de abril de 2018.] http://www.ctr.unican.es/asignaturas/procodis_3_ii/doc/procodis_1_01.pdf.

50. **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.

51. **Fulgueiro, L.A.** CProg. *Estructura de datos Trie – arbol de prefijos.* [En línea] 25 de Marzo de 2018. [Citado el: 3 de Abril de 2018.] http://cprog.cubava.cu/2018/03/25/trie-arbol-de-prefijos/.

52. **BECK, K.** *Extreme Programming Explained.* New York : Addison-Wesley Professional, 1999.

53. **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.

54. **UNAD.** CAP. 9 BINDER - PROCESO DE PRUEBAS. *Tipos de Estrategia de pruebas.* [En línea] [Citado el: 2 de abril de 2018.] https://ldc.usb.ve/~teruel/southpark/DP/pruebas/documentos/capitulo9.htm.

55. **SOMMARIVA, A.** Pruebas Unitarias, Parte 1: Introducción y utilización de objetos simulados (Mock). *MicroGestion.* [En línea] 24 de febrero de 2014. [Citado el: 22 de Abril de 2018.] http://www.microgestion.com/index.php/mg-developers/articulos/74-unit-test-part1-mock.

56. **MAVEN.** Curso de Especialista en Aplicaiones y Servicios Web conJava Enterprise. *Casos de prueba: JUnit.* [En línea] 26 de Junio de 2014. [Citado el: 24 de Abril de 2018.] http://www.jtech.ua.es/j2ee/publico/lja-2012-13/sesion04-apuntes.html.

57. **KEN AUER, R. M.** *Extreme Programming Applied.* New York : Addison-Wesley Professional, 2001.

58. **VILLAYANDRE LLAMAZARES, M.** Lingüística computacional II. Curso monográfico sobre Lingüística de corpus. *Departamento de Filología Hispánica y Clásica.* [En línea] [Citado el: 10 de Enero de 2018.] https://reflexionesdecoloniales.files.wordpress.com/2014/09/tipos-de-corpus.pdf.

59. **TORUELLA, J. y JOAQUIM, L.** Diseño de Corpus Textuales y Orales. *Laboratorio de Tecnologías Lingüísticas.* [En línea] 1999. [Citado el: 27 de marzo de 2018.] http://latel.upf.edu/traductica/lc/material/torruella_llisterri_99.pdf.

60. **GÓMEZ DÍAZ, R.** La lematización en español: Una aplicación para la recuperación de información. *Revista española de Documentación Científica.* [En línea] 2005. [Citado el: 9 de Enero de 2018.] http://redc.revistas.csic.es/index.php/redc/article/viewFile/301/348. ISBN 84-9704-186-0.

61. **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 2018.] http://www.alegsa.com.ar/Dic/api.php.

62. **NETBEANS.** *Oracle Corporation and/or its affiliates.* [En línea] [Citado el: 9 de Enero de 2018.] http://netbeans.org/.

63. **DEUTSCH, M.** Verification and Validation. [ed.] Jensen R. y Tonies C. *Software Engineering.* New York : Prentice Hall, 1979, págs. 329-408.

64. **PROGRAMACIÓNDESARROLLO.** Entorno de desarrollo integrado(IDE). [En línea] 25 de enero de 2013. [Citado el: 10 de Enero de 2018.] https://fergarciac.wordpress.com/2013/01/25/entorno-de-desarrollo-integrado-ide/.

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 Minning 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-

ANEXOS

Anexo 1. Historias de usuario

Anexo 1. HU-2 Cálculo de la relevancia de los términos

Historia de Usuario					
Numero de Historia:	HU-6	Usuario:	Programadores	Fecha:	22/01/2018
Título:	Almacenar la matriz de semejanza en un archivo binario.				
Descripción:	El objeto matriz se serializa y se almacena con extensión .bin en un espacio físico en el disco				
Observaciones:	Debe haber culminada la construcción de la matriz de semejanza.				
Puntos Estimados:	2	Puntos Reales: 2	2		

Anexo 1. HU-4. Almacenar diccionario de términos

Historia de Usuario					
Numero de Historia:	HU-4	Usuario:	Programadores	Fecha:	19/01/2018
Título:	Almacenar diccionario de términos.				
Descripción:	El objeto diccionario se serializa y se almacena con extensión .bin en un espacio físico en el disco				
Observaciones:	Debe haber culminada la construcción del diccionario				
Puntos Estimados:	2	Puntos Reales: 2	2		

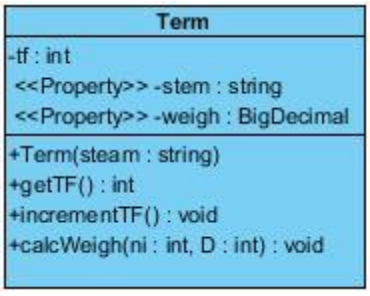
Anexo 1. HU-6. Almacenar la matriz de semejanza en un archivo binario

Historia de Usuario					
Numero de Historia:	HU-2	Usuario:	Programadores	Fecha:	20/01/2018
Título:	Cálculo de la relevancia de los términos.				
Descripción:	El componente debe calcular para cada término del documento la relevancia o peso. El peso es la importancia que posee el término en el documento.				
Observaciones:	Para el cálculo de la relevancia debe recibir la cantidad de repeticiones del término en el documento analizado.				
Puntos Estimados:	3	Puntos Reales: 3		3	

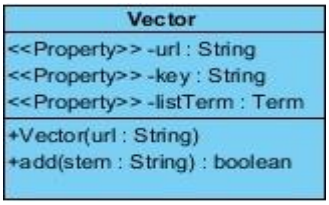
Requirement(6) \ Historia de usuario(6)	Construir vector de término	Cálculo de relevancia de los términos	Construir diccionario de términos	Almacenar diccionario de término	Constuir matriz de semejanza	Almacenar matriz de semejanza
RF1	Y					
RF2		Y				
RF3			Y			
RF4				Y		
RF5					Y	
RF6						Y

Anexo 1. Matriz de trazabilidad RF-HU

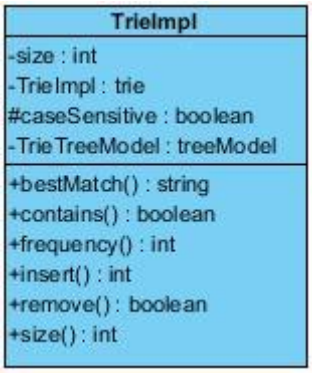
Anexo 2. Entidades del diagrama de clases del sistema



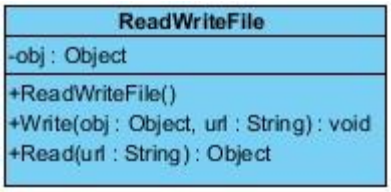
Anexo 2. Clase Term



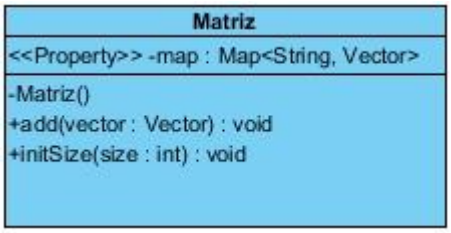
Anexo 2. Clase Vector



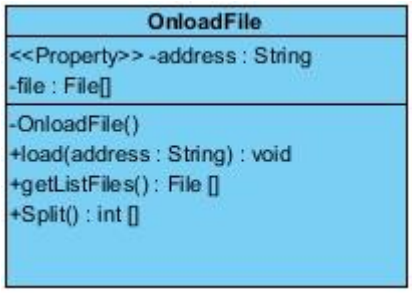
Anexo 2. Clase TrieImpl



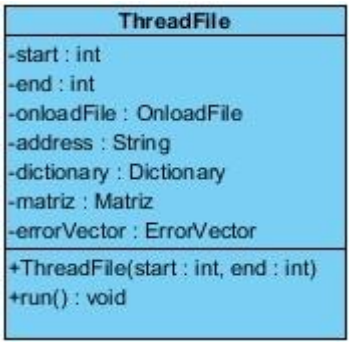
Anexo 2. Clase ReadWriteFile



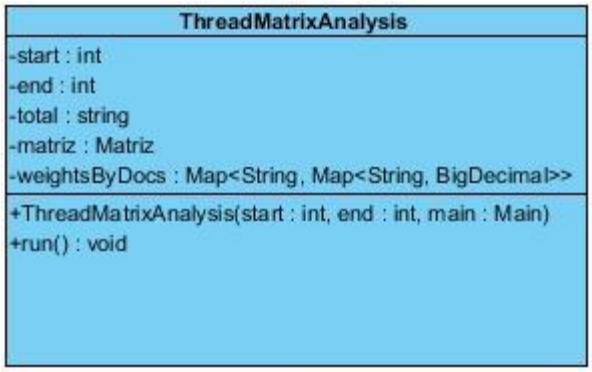
Anexo 2. Clase Matriz



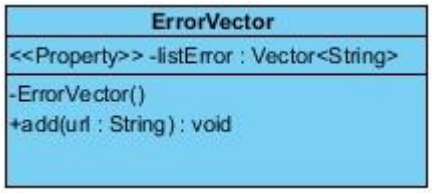
Anexo 2. Clase OnloadFile



Anexo 2. Clase ThreadFile

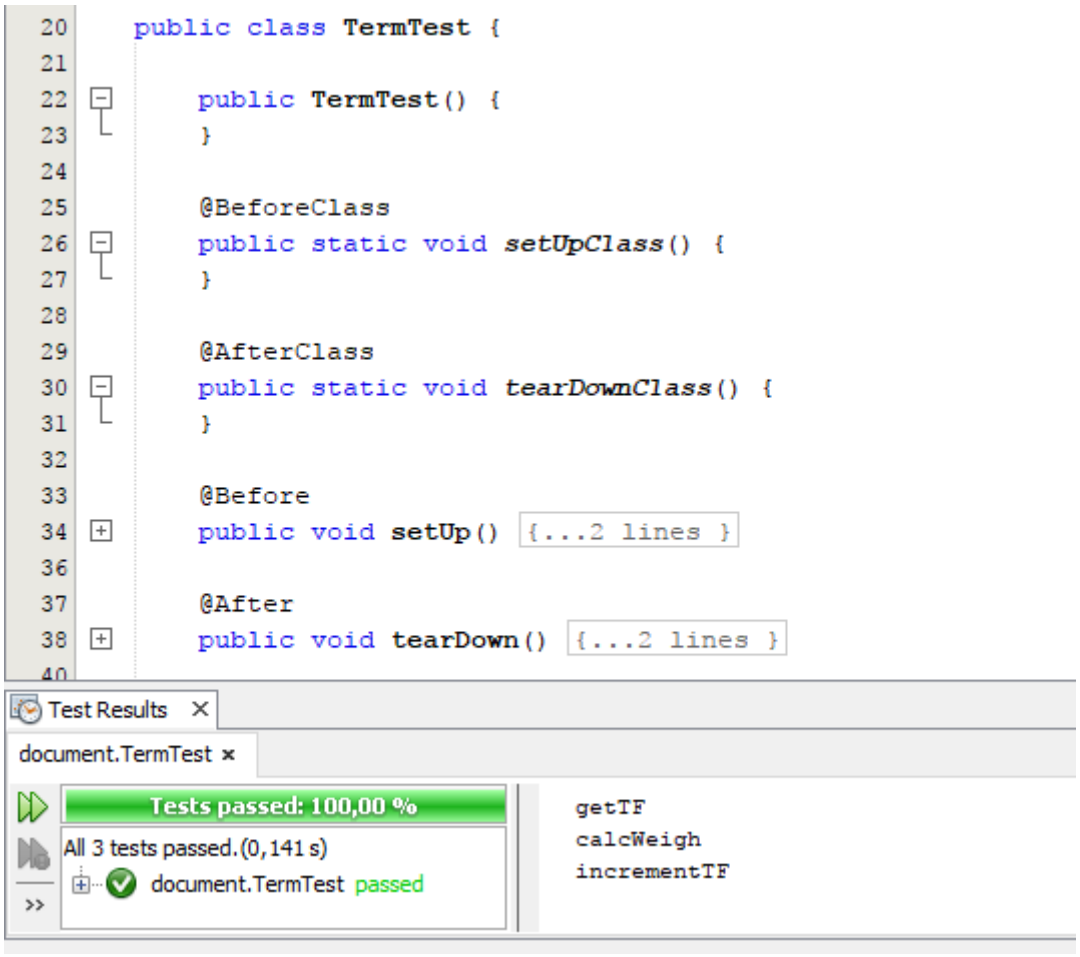


Anexo 2. Clase ThreadMatrixAnalysis

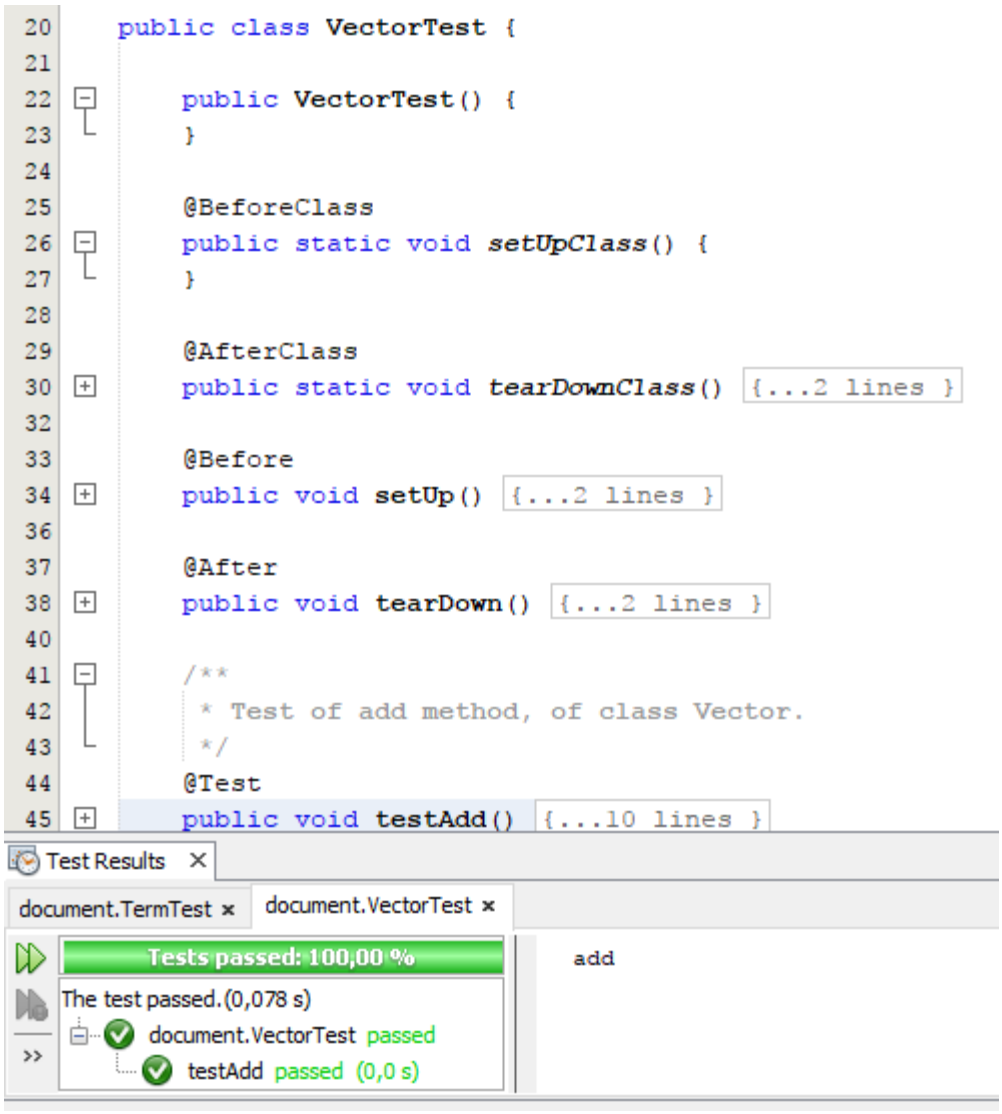


Anexo 2. Clase ErrorVector

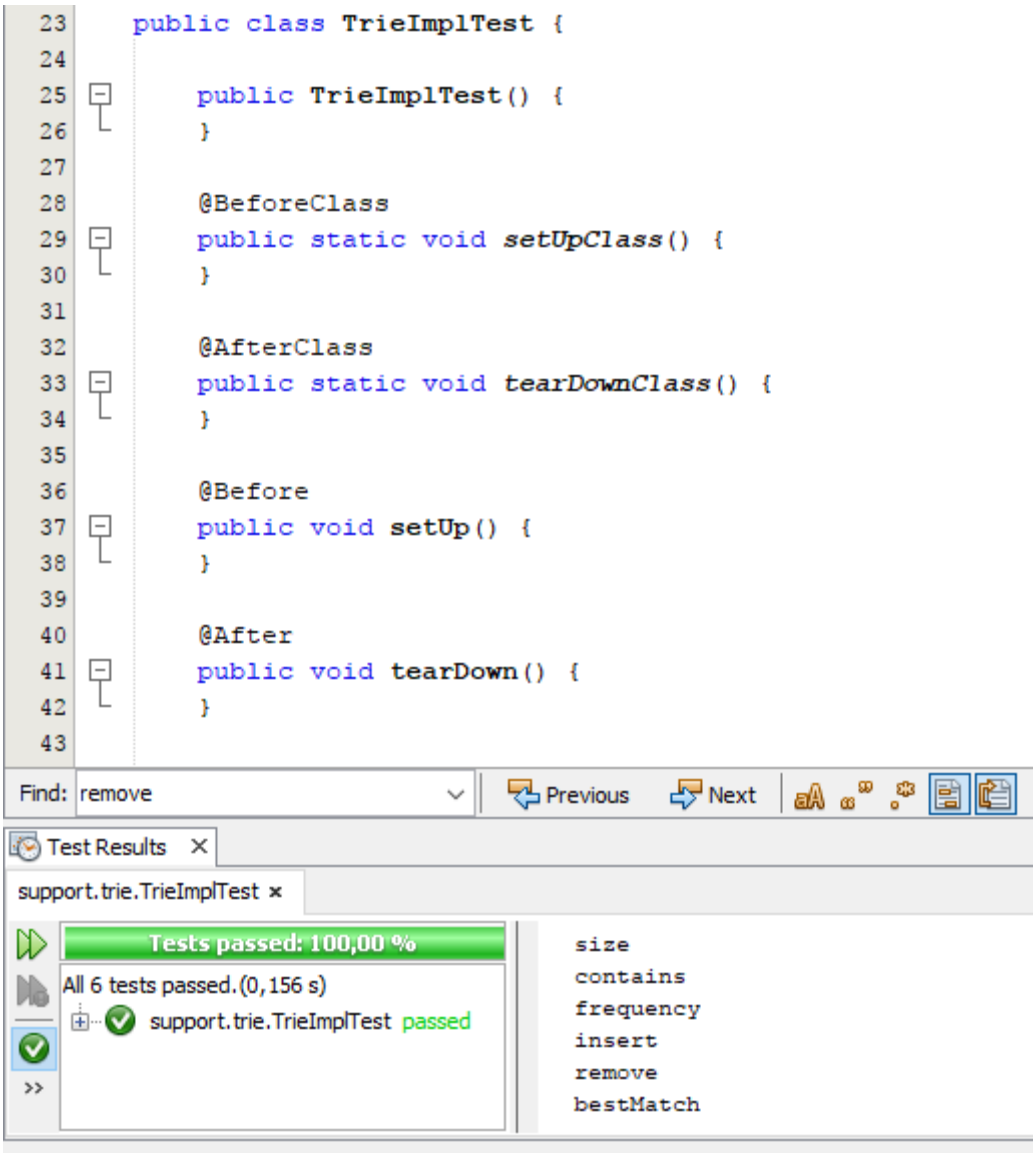
Anexo 3. Pruebas unitarias



Anexo 3. Pruebas Unitarias. TermTest



Anexo 3. Pruebas Unitarias. VectorTest



Anexo 13 Pruebas Unitarias. TrieImplTest

```
22      public MatrizTest() {
23      }
24
25      @BeforeClass
26      public static void setUpClass() {...2 lines }
28
29      @AfterClass
30      public static void tearDownClass() {...2 lines }
32
33      @Before
34      public void setUp() {...2 lines }
36
37      @After
38      public void tearDown() {...2 lines }
40
41      /** Test of add method, of class Matriz ...3 lines */
44      @Test
45      public void testAdd() {
46          System.out.println("add");
47          Vector vector = new Vector("Doc");
48          Matriz instance = new Matriz();
49          instance.add(vector);
50      }
```

Test Results

document.MatrizTest

Tests passed: 100,00 %

All 3 tests passed. (0,276 s)

document.MatrizTest passed

add
initSize
getMap

Anexo 3. Pruebas Unitarias. MatrizTest

```java
public class ReadWriteFileTest {

    public ReadWriteFileTest() {
    }
        /** Test of Write method, of class ReadWriteFile ...3 lines */
    @Test
    public void testWrite() {
        System.out.println("Write");
        Object obj = null;
        String url = "";
        ReadWriteFile instance = new ReadWriteFile();
        instance.Write(obj, url);
    }

    /** Test of Read method, of class ReadWriteFile ...3 lines */
    @Test
    public void testRead() {
        System.out.println("Read");
        String url = "";
        ReadWriteFile instance = new ReadWriteFile();
        Object expResult = null;
        Object result = instance.Read(url);
        assertEquals(expResult, result);
    }
```

Test Results

document.MatrizTest | process.ReadWriteFileTest

Tests passed: 100,00 %

Both tests passed. (0,104 s)

process.ReadWriteFileTest passed

```
Write
Read
```

Anexo 3. Pruebas Unitarias. ReadWriteFileTest

GLOSARIO DE TÉRMINOS

1. **corpus textual:** representación escrita de la lengua (58) (59).
2. **lematización:** es una etiqueta informática que en español coincidirá generalmente con el lexema o raíz de las palabras, pero no necesariamente han de ser equivalentes (60)
3. **legomena:** dicho una sola vez.
4. **API:** (Application Programming Interface - Interfaz de Programación de Aplicaciones). Grupo de rutinas (conformando una interfaz) que provee un sistema operativo, una aplicación o una biblioteca, que definen cómo invocar desde un programa un servicio que estos prestan (61).
5. ***token*:** es una cadena de caracteres que tiene un significado coherente en cierto lenguaje. Ejemplos de *tokens* podrían ser palabras clave, identificadores, números, signos, o un conjunto de varios caracteres.