Universidad de las Ciencias Informáticas

Facultad 2

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

"Solución informática para la limpieza de datos en archivos digitales textuales."

Autor: Alejandro Cuellar Monteagudo

Tutores: Ing. Andis Eloy Yero Guevara

Ing. Pável Reyes Estévez

La Habana, Cuba.

2018

DECLARACIÓN DE AUTORÍA

Se declara que Alejandro Cuellar Monteagudo es el único autor del trabajo de diploma “Solución informática para la limpieza de datos en archivos digitales textuales” y autorizamos a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los ___ días del mes de junio del año 2018.

_____

Alejandro Cuellar Monteagudo

Firma Autor

_____ _____

Ing. Pável Reyes Estévez

Firma Tutor

Ing. Andis Eloy Yero Guevara

Firma Tutor

Resumen

La recuperación de información es el área del conocimiento mediante la cual se localiza y accede a los recursos de información que son pertinentes para la resolución de un problema determinado. Incluye la representación, el almacenamiento, la organización y el acceso a elementos de información. Uno de los problemas actuales en la recuperación de información es la representación del contenido de archivos digitales, debido a la necesidad de realizar actividades de clasificación de archivos digitales para la identificación y el agrupamiento de documentos semejantes con características comunes. En esta investigación se tiene como objetivo el desarrollo de una solución informática para la representación del corpus textual de archivos digitales textuales para la clasificación automática de documentos. Para el logro del objetivo se realiza un estudio de los antecedentes de los sistemas informáticos de recuperación de información y se prepara el entorno de desarrollo de software para resolver el problema en cuestión. Se utiliza el algoritmo de Porter que permite lematizar las palabras de los archivos digitales.

Palabras clave: archivo digital textual, corpus textual, lematizar, recuperación de información.

Abstract

Information retrieval is the area of knowledge through which it locates and accesses the information resources that are relevant to the resolution of a given problem. It includes representation, storage, organization and access to information elements. One of the current problems in the retrieval of information is the representation of the content of digital files, due to the need to perform activities of classification of digital files for the identification and grouping of similar documents with common characteristics. The objective of this research is to develop a computer solution for the representation of the textual corpus of textual digital files for the automatic classification of documents. To achieve the objective, a background study of the information retrieval computer systems is carried out and the software development environment is prepared to solve the problem in question. The algorithm of Porter is used that allows to lematizar the words of the digital archives.

Keywords: digital textual file, textual corpus, lemmatize, information retrieval.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Introducción

En la actualidad es creciente la necesidad que tienen las organizaciones de velar por la calidad de sus datos como fuente fundamental para la recuperación de información, por lo que es de vital importancia contar con procedimientos que ayuden en el proceso de limpieza de datos.

La información es un conjunto de datos con un significado que constituye un mensaje sobre un determinado contexto, disponible para uso inmediato y que proporciona orientación en la toma de decisiones.(Julián Pérez Porto 2012) 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.(Julián Pérez Porto 2009)

Un archivo es un conjunto de documentos producidos o recibidos por toda persona, servicio u organismo, en el ejercicio de su actividad. Su importancia viene determinada por su uso; por lo que un buen manejo de la información para cualquier empresa, entidad, institución, gobierno o persona, fortalece la toma de decisiones y permite el logro de sus objetivos.

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

La informática más que una herramienta, es una ciencia que se ha desarrollado a lo largo de los años. Este avance ha logrado resolver diversos problemas a los que se enfrenta el hombre en su vida cotidiana, lo que ha captado la atención de muchas empresas e instituciones que buscan una mejora en los procesos de toma de decisiones. El uso y desarrollo de esta ciencia se hace cada vez más necesario para poder analizar las grandes cantidades de información generadas en los procesos empresariales.

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

La recuperación de información es el área del conocimiento mediante la cual se localiza y accede a los recursos de información que son pertinentes para la resolución de un problema determinado. Incluye la representación, el almacenamiento, la organización y el acceso a elementos de información.(Pino 2004)

Uno de los problemas actuales en la recuperación de información es la representación del contenido de archivos digitales textuales, debido a la necesidad de realizar actividades de clasificación de archivos digitales textuales para la identificación y el agrupamiento de documentos semejantes con características comunes.

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, se dificultan ante la necesidad de diccionarios, expertos humanos y disponibilidad de espacio físico y digital para el almacenamiento de las colecciones de archivos.

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

Por lo anteriormente planteado se define como **problema de investigación:** ¿Cómo realizar la limpieza de datos en el procesamiento de los archivos digitales para la clasificación automática de documentos y la recuperación de la información?

Definiéndose como **objeto de estudio:** El Procesamiento de archivos digitales textuales.

Definiéndose como **objetivo general:** Desarrollar un componente informático que permita la limpieza de datos en el procesamiento del corpus textual de los archivos digitales textuales.

Enmarcado en el **campo de acción:** Limpieza de datos en archivos digitales textuales

Objetivos específicos:

1. 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.
2. Implementar un componente informático para la limpieza de datos del corpus textual de archivos digitales textuales.
3. Comprobar el funcionamiento del algoritmo y componente desarrollado a través de la realización de pruebas funcionales y de aceptación.

Para dar cumplimiento a los objetivos específicos se propone ejecutar las siguientes Tareas:

Objetivo 1:

- ✓ Descripción de los conceptos fundamentales para la limpieza de datos en el procesamiento del corpus textual de archivos digitales textuales.
- ✓ Identificación de las principales soluciones informáticas que utilicen modelos o algoritmos de recuperación de información para la definición de los antecedentes de la investigación.
- ✓ Selección de la metodología de desarrollo de software, herramientas, lenguajes y tecnologías a utilizar para el desarrollo de la solución informática que permita la representación del corpus textual de archivos digitales.

Objetivo 2:

- ✓ Selección de las estructuras de datos para la representación del corpus textual de los documentos digitales.
- ✓ Selección de técnicas que mejoren el rendimiento computacional para disminuir el tiempo de corrida y respuesta del algoritmo seleccionado en la representación del corpus textual de documentos digitales.

Objetivo 3:

- ✓ Comprobación del funcionamiento del algoritmo para demostrar la correcta representación del corpus textual.
- ✓ Definición de una estrategia de prueba para comprobar el funcionamiento del componente desarrollado.

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:** Se empleó para crear abstracciones con vistas a explicar la realidad, operando de forma práctica mediante el uso de una herramienta de modelación llamada Visual Paradigm en su versión 5.0.

Métodos empíricos

- **Observación**: Se aplicó fundamentalmente para evaluar la dimensión comportamental, en todas las diferentes etapas del proceso, permitiendo identificar las causas de la problemática anteriormente descrita y detallada y de esta manera proyectar su posible solución.

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

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

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

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

Capítulo 1: Fundamentación teórica

1.1 Introducción.

En este capítulo se abordan los conceptos principales más asociados al dominio del problema principal en cuestión, las principales definiciones de procesos y software basados en la limpieza de datos, principalmente las soluciones existentes actualmente en el mundo que dan un punto de partida al desarrollo del software. Además de analizar la metodología, herramientas, tecnologías y lenguajes seleccionados para el desarrollo de la solución.

1.2 Aspectos teóricos de la investigación

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

Limpieza de datos

La limpieza de datos o depuración de datos es el proceso de detectar, corregir o eliminar registros corruptos o imprecisos de un conjunto de registros, tablas o bases de datos con información incorrecta, incompleta, mal formateada o duplicada.(Kyocera 2017)

Lematización

La lematización es un proceso lingüístico que consiste en, dada una forma flexionada, hallar el lema correspondiente. El lema es la forma que por convenio se acepta como representante de todas las formas flexionadas de una misma palabra. Es decir, el lema de una palabra es la palabra que nos encontraríamos como entrada en un diccionario tradicional: singular para sustantivos, masculino singular para adjetivos, infinitivo para verbos.(LEXICOON 2018)

Corpus Textual

Los corpus o corpora, su plural en latín se definen como aquel conjunto de datos o textos de un mismo tipo que sirve de base a una investigación. Para ser más concretos, un corpus lingüístico recoge todo un vastísimo conjunto de textos, desde guiones de cine a obras de teatro, transcripciones radiofónicas o ensayos, donde quedan almacenados un conjunto aún más amplio de términos.(Conde 2005)

Concurrencia

Dos o más procesos decimos que son concurrentes, paralelos, o que se ejecutan concurrentemente, cuando son procesados al mismo tiempo, es decir, que para ejecutar uno de ellos, no hace falta que se haya ejecutado otro.(Drake 2015)

Archivo Digital

Un archivo digital, también denominado Fichero, es una unidad de datos o información almacenada en algún medio que puede ser utilizada por aplicaciones de la computadora. Cada archivo se diferencia del resto debido a que tiene un nombre propio y una extensión que lo identifica. Esta extensión sería como el apellido y es lo que permite diferenciar el formato del archivo y, asimismo, interpretar los caracteres que conforman el contenido del archivo. De esta manera, un archivo de texto, podrá tener la extensión .txt (el nombre completo sería: ARCHIVO.txt); uno de documento enriquecido, .doc, .pdf; uno de imágenes, .jpg, .gif; y lo mismo ocurre con cada formato.(Wordpress 2009)

1.3 Soluciones Existentes

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

DAC

DAC es un software de recuperación de información multiplataforma creado en la Universidad de las Ciencias Informáticas por el Ing. Andis Eloy Yero Guevara y el Ing. Pável Reyes Estévez. Está compuesto por un conjunto de herramientas que permite la estandarización de documentos Word a formato de texto, filtra los documentos eliminando de los mismos un conjunto de palabras no relevantes, realiza la lematización de los documentos, construye vectores de términos con el lema de las palabras y calcula el peso de todas las palabras.

A partir del estudio realizado acerca de este software se determinó que contiene un conjunto de herramientas a tener en cuenta a la hora de realizar el software. Entre sus principales deficiencias se encontró que a la hora de analizar un gran número de documentos el tiempo de ejecución aumenta considerablemente.

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.(APACHE 2016) 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.

WordStat

WordStat es un software privativo de análisis cualitativo de datos desarrollado por la compañía Provalis Research con sede en Montreal, Canadá. Este analiza el texto y relaciona su contenido a información estructurada, que incluya datos numéricos y categoriales. Sus principales funciones son el análisis de contenido en colección de documentos ANSI o RTF y variables alfanuméricas cortas (hasta de 255 caracteres). También permite la exclusión opcional de pronombres y conjunciones, mediante el uso de listas de exclusión definidas por el usuario, así como el análisis de frecuencia de palabras clave, frases, categorías, conceptos derivados o códigos definidos por el usuario e ingresados manualmente dentro de un texto. Actualmente su distribución solo funciona en sistemas operativos Windows, para su uso en Mac y Linux es necesario un emulador.(RESEARCH 2017) 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.

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.(Software-shop 2018)

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.

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

A pesar de que las soluciones informáticas analizadas anteriormente abarcan un gran número de funcionalidades que les permiten a los usuarios utilizarlas como herramientas para la recuperación de información y representación del corpus textual de documentos, presentan deficiencias que son necesarias resolver. WordStat, y T-LAB muestran un conjunto de características que se pueden tener en cuenta para el desarrollo de la solución, entre las cuales se encuentra la exclusión de pronombres y conjunciones del contenido de los documentos digitales a través del uso de listas de exclusión.

Además, conforman diccionarios con las raíces de los términos y sus frecuencias de aparición. Sin embargo, son soluciones informáticas de plataforma Windows, esto imposibilita su integración con varios tipos de software y limita su capacidad multiplataforma, además de ser software propietario lo cual no se corresponde con las proyecciones de la Universidad de las Ciencias Informáticas y Cuba con respecto al uso del software libre y la soberanía tecnológica.

Lucene y DAC presume ser la solución informática más cercana para resolver el objetivo de la investigación. Cuenta con un conjunto de funcionalidades para tratar el corpus textual de los documentos digitales. Alberga todas las características encontradas en las otras dos soluciones informáticas explicadas, es multiplataforma y permite la integración de sus componentes con otros sistemas. Sin embargo, su principal deficiencia es que al crecer el volumen de datos a procesar, el tiempo de respuesta es elevado, lo que disminuye su utilización por potenciales deficiencias de rendimiento.

Lucene a pesar de que se puede integrar con diferentes sistemas su incapacidad para la comparación entre documentos resulta ser un problema para su selección, debido a que su procesamiento se basa en la comparación de una consulta con una colección de textos y no entre colecciones de textos. Por otro lado, necesita una indexación previa para realizar búsquedas sobre el corpus textual de los documentos digitales.

1.4 Metodología, Lenguajes y Herramientas de Desarrollo

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

1.4.1 Metodología de Desarrollo

El Proceso Unificado Ágil *de Scott Ambler o Agile Unified Process* (AUP) 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 que aún se mantienen válidos en RUP. AUP se preocupa especialmente de la gestión de riesgos. Propone que aquellos elementos con alto riesgo obtengan prioridad en el proceso de desarrollo y sean abordados en etapas tempranas del mismo.
En la Universidad de la Ciencias Informáticas (UCI) se desarrolló una versión de esta metodología ágil con el objetivo de crear un estándar en el proceso productivo de manera que se adapte al ciclo de vida de los proyectos en la UCI. En esta variación de AUP para la UCI se mantiene la primera fase de inicio, la segunda fase de ejecución en esta variación para la UCI resume las fases restantes de elaboración, construcción y transición.

La metodología a utilizar es AUP-UCI enfocándose en el escenario número cuatro Historia de usuario. Dicha selección se basa en que el equipo de desarrollo involucrado es pequeño, existe una estrecha relación con el cliente, llegando este a formar parte del equipo y se necesitan constantes entregas del avance en el proceso de desarrollo. También se tiene en cuenta que durante este proceso el software se puede ver sometido a constantes cambios.

Finalmente se encuentra la fase de cierre donde se analiza los resultados del proyecto y se realizan las actividades formales de cierre de proyecto.

Las historias de usuario son un instrumento propuesto por la metodología seleccionada para el levantamiento de requerimientos en el desarrollo de software. Constituyen tarjetas dónde el cliente describe brevemente (con el fin de que sean dinámicas y flexibles) las características que el sistema debe poseer, sean requisitos funcionales o no funcionales.

A partir de un estudio realizado de los elementos más importantes recomendados por varios autores 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.

1.4.2 Proyecto a desarrollar

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

1.4.3 Cliente

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

1.4.4 Tecnologías

Lenguaje de programación

Un lenguaje de programación es un lenguaje diseñado para describir el conjunto de acciones consecutivas que un equipo debe ejecutar. Por lo tanto, un lenguaje de programación es un modo práctico para que los seres humanos puedan dar instrucciones a un equipo.(IsraelCCM. 2018)

Java

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo.(ICTEA 2018)

IDE de programación

Un entorno de desarrollo integrado, es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI).(Wordpress 2013)

Apache POI

La librería Apache POI permite acceder a los documentos del producto ofimático Microsoft Office utilizando el lenguaje de programación Java y otros del ecosistema de la Máquina virtual de Java.(Pico.dev 2016)

1.4.5 Herramientas

Herramienta CASE Visual Paradigm (v8.0)

Visual Paradigm es una herramienta *CASE*: Ingeniería de Software Asistida por Computación. La misma propicia un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación.(PRESSMAN 2002b)

Netbeans(v8.2)

Netbeans es un entorno de desarrollo gratuito y de código abierto. Permite el uso de un amplio rango de tecnologías de desarrollo tanto para escritorio, como aplicaciones Web, o para dispositivos móviles. Da soporte a las siguientes tecnologías, entre otras: **Java**, **PHP**, **Groovy**, **C/C++**, **HTML5** entre otros. Además, puede instalarse en varios sistemas operativos como Windows, Linux, Mac OS.(Calendamaia 2014)

1.5 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 las soluciones WordStat, T-LAB, Lucene y DAC. 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 la metodología AUP. A partir de este estudio se seleccionó el escenario número cuatro para guiar el desarrollo de software teniendo

en cuenta las condiciones y características del equipo de desarrollo. En apoyo a la metodología se seleccionó la herramienta CASE Visual Paradigm 8.0 que permiten 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: ELEMENTOS FUNDAMENTALES DE LA PROPUESTA DE SOLUCIÓN

2.1 Introducción

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

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

2.2 Modelo de la Propuesta de solución

A partir del estudio de los antecedentes y consecuentemente con el objetivo de la investigación, se concluye que es necesario diseñar e implementar un componente informático que permita la limpieza de datos en archivos digitales textuales. La Figura 1 presenta un esquema general que aclara el proceso de limpieza de datos 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.

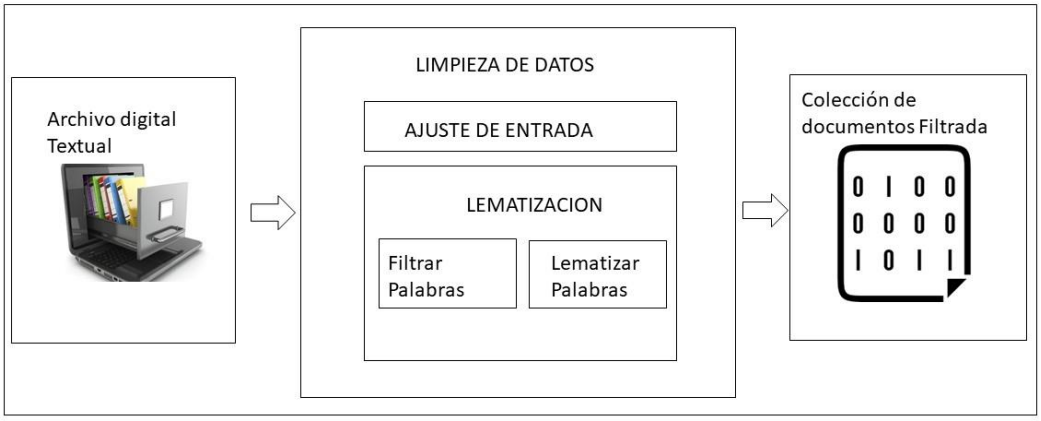


Fig. 1 Modelo de la propuesta de solución

Como se observa en la figura 2.1 el modelo de propuesta de solución cuenta con tres partes fundamentales: entrada, limpieza de datos y salida. El flujo de entrada consiste en una colección de archivos digitales a procesar por el filtrado de la colección. Este primeramente realiza un ajuste de la entrada para luego aplicar la limpieza de datos. Dicho modelo está dividido en dos etapas fundamentales: la primera etapa cuenta con el lematizador, que se encarga de filtrar y reducir cada palabra a su raíz semántica. Finalmente se obtiene un archivo binario con la información de los archivos digitales luego de hallar la raíz semántica a cada una de las palabras.

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

2.3 Modelo de dominio

Un Modelo de Dominio es un artefacto de la disciplina de análisis, construido con las reglas de UML durante la fase de concepción, en la tarea construcción del modelo de dominio, presentado como uno o más diagramas de clases y que contiene, no conceptos propios de un sistema de software sino de la propia realidad física. Los modelos de dominio pueden utilizarse para capturar y expresar el entendimiento ganado en un área bajo análisis como paso previo al diseño de un sistema, ya sea de software o de otro tipo. Similares a los mapas mentales utilizados en el aprendizaje, el modelo de dominio es utilizado por el analista como un medio para comprender el sector industrial o de negocios al cual el sistema va a servir.(Garcerant 2008) La figura 2.2 muestra una representación estática del entorno real del proyecto mediante el modelo de dominio.

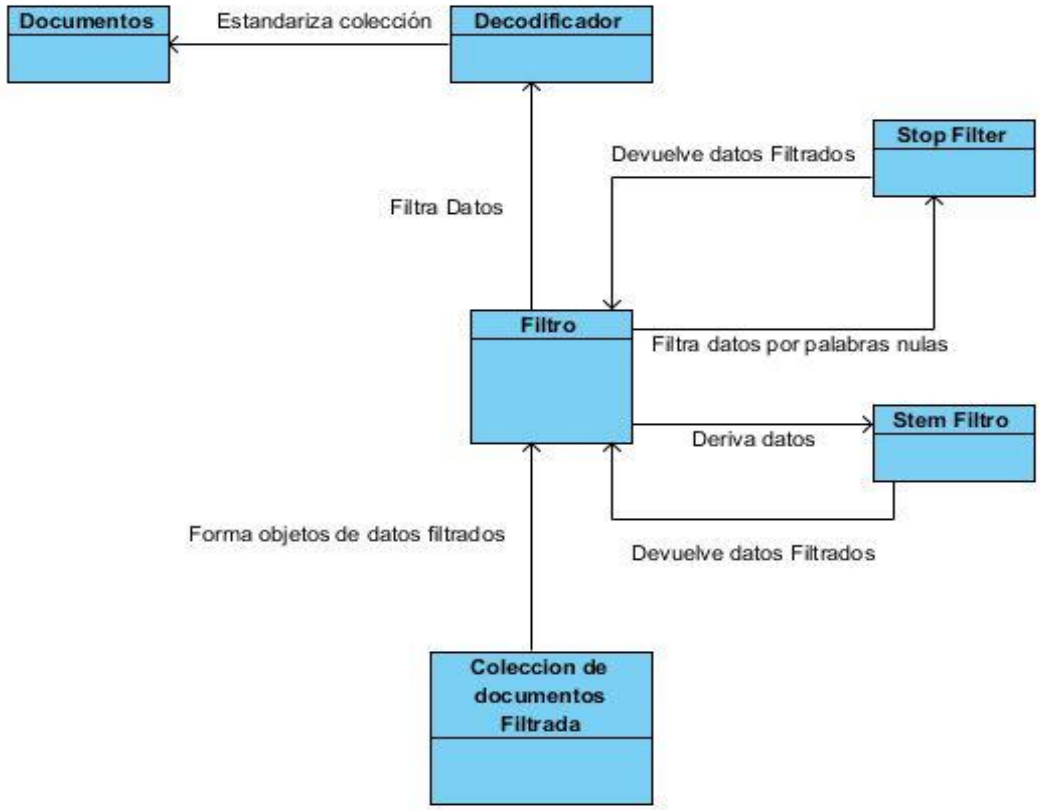


Fig. 2 Modelo de dominio

En el modelo de dominio mostrado se aprecia que los documentos a procesar necesitan ser estandarizados, tarea que se realiza por medio de un decodificador. Esto se debe a que existen múltiples extensiones para la escritura de textos digitales como: doc, odt, pdf, docx, txt, rtf, entre otras. Independiente de la extensión, la codificación de los caracteres que conforman los textos es otro parámetro a tener en cuenta, la misma

puede ser: ISO-8859-1, UTF-8, UTF-16, ANSI, entre otras. Por tal motivo estandarizar la colección de entrada a un formato y codificación predeterminada garantiza el funcionamiento y acoplamiento del proceso de lectura de los documentos.

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

2.4 Requerimientos del Sistema

Una vez implementado el sistema que se tiene como propuesta de solución deberá cumplir con ciertos requisitos definidos previamente por el equipo de desarrollo y el cliente.

Los requerimientos son condiciones o capacidades que el sistema tiene que tener para satisfacer algún documento formal, estos describen a su vez todo lo que el sistema debe hacer o tener, también deben ser especificados por escrito como acuerdo del contrato.

Una de sus definiciones los describe como “especificación de lo que debe ser implementado. Estos son descripciones de cómo el sistema se debe comportar, de las propiedades y atributos del mismo. Deben ser una restricción del proceso de desarrollo del sistema”.

2.4.1 Requisitos Funcionales

Los requisitos funcionales definen qué hace el sistema, describen entradas y salidas, es decir, las funciones del sistema. A continuación, se presentan los requisitos funcionales identificados:

Tabla 1 Requisitos Funcionales

Requisitos	Descripción
RF.1 Estandarizar mediante UTf-8 la colección de documentos de entrada.	El componente debe estandarizar la colección de documentos a un formato y codificación específica. Mediante un proceso de conversión de los textos produce como salida una colección de textos en formato txt con codificación UTF-8.
RF.2 Leer la colección de documentos codificados en UTF-8.	El Componente debe la colección de documentos que se encuentra en formato txt y codificación UTF-8
RF.3 Definir del listado de palabras no relevantes para la clasificación automática de documentos en el lenguaje español.	Definir un conjunto de palabras que no aportan significado por sí solas y por tanto los no se tienen en cuenta.
RF.4 Aplicar el filtrado de palabras no relevantes a la colección de entrada.	El componente debe eliminar las palabras que sean de poca importancia en dicho documento una vez leído.
RF.5 Lematizar mediante el algoritmo de Porter todas las palabras de la colección de documentos.	El componente debe hallar el lema de las palabras de la colección de entrada.
RF.6 Construcción de las estructuras de datos definidas en la representación del corpus textual en contribución al Modelo de Espacio Vectorial.	El componente debe construir las estructuras de datos una vez lematizado el contenido de los documentos.

2.4.2 Historias de usuario

Las historias de usuario son un instrumento para el levantamiento de requerimientos para el desarrollo de un software, que ha emergido con la aparición de los nuevos marcos de trabajo de desarrollo ágil.(PMOinformatica 2013)

Cada historia de usuario debe ser lo suficientemente comprensible y no muy extensa en cuanto a la cantidad de requisitos a incluir en ella, con el fin de que se puedan implementar en poco tiempo. No existe un estándar determinado para redactar una historia de usuario, según la metodología seleccionada, el equipo de desarrollo determina que es lo importante que debe quedar escrito y como llevar el control de cada una. A partir de un estudio realizado de los elementos más importantes recomendados por varios autores 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.
- Nombre del Requisito: 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.
- Tiempo Estimado: tiempo en días que se asigna o se supone al desarrollo de la historia de usuario.
- Tiempo Real: tiempo real en semanas que demora el desarrollo de la historia de usuario.

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

Las tablas 2, 3, 4, 5 y 6 muestran las historias de usuario Estandarizar mediante UTf-8 la colección de documentos de entrada, Leer la colección de documentos codificados, Aplicar el filtrado de palabras no relevantes a la colección de entrada, Lematizar mediante el algoritmo de Porter todas las palabras de la colección de documentos de entrada, Construcción de las estructuras de datos definidas en la representación del corpus textual en contribución al Modelo de Espacio Vectorial. Los campos que contienen son los definidos anteriormente.

Tabla 2 HU-1. Estandarizar mediante UTf-8 la colección de documentos de entrada

Historia de Usuario	
Número: HU-1	**Nombre del requisito:** Estandarizar mediante UTf-8 la colección de documentos de entrada
Programador: Alejandro Cuellar Monteagudo	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado** 31
Riesgo en Desarrollo: Ninguno	**Tiempo Real:** 4
Descripción: El componente debe estandarizar la colección de documentos a un formato y codificación específica. Inicialmente recibe una colección de documentos en distintos formatos y codificaciones. Mediante un proceso de conversión de los textos produce como salida una colección de textos en formato txt con codificación UTF-8.	
Observaciones: El proceso de estandarización debe aceptar varios tipos de formatos y codificación: ISO-2022-CN, BIG5, EUC-TW, GB18030, HZ-GB-23121, ISO-8859-5, KOI8-R, WINDOWS-1251, IBM866, IBM855, ISO-8859-7, WINDOWS-1253, ISO-8859-8, WINDOWS-1255, ISO-2022-JP, SHIFT_JIS, EUC-JP, ISO-2022-KR, EUC-KR, Unicode, UTF-8, UTF-16BE / UTF-16LE, UTF-32BE / UTF-32LE.	

Tabla 3 HU-2. Leer la colección de documentos codificados en UTF-8

Historia de Usuario	
Número: HU-2	**Nombre del requisito:** Leer la colección de documentos codificados en UTF-8.
Programador: Alejandro Cuellar Monteagudo	**Iteración Asignada:** 2
Prioridad: Alta	**Tiempo Estimado** 10
Riesgo en Desarrollo: Ninguno	**Tiempo Real:** 2
Descripción: El componente debe leer la colección de documentos una vez estandarizados con la codificación UTF-8. Mediante un proceso que permite la lectura de documentos txt.	
Observaciones:	

El proceso de lectura de documentos debe leer los documentos txt con codificación UTF-8.

Tabla 4 HU-3. Aplicar el filtrado de palabras no relevantes a la colección de entrada

Historia de Usuario	
Número: HU-3	**Nombre del requisito:** Aplicar el filtrado de palabras no relevantes a la colección de entrada.
Programador: Alejandro Cuellar Monteagudo	**Iteración Asignada:** 2
Prioridad: Medio	**Tiempo Estimado** 5
Riesgo en Desarrollo: Ninguno	**Tiempo Real:** 1
Descripción: El componente debe eliminar las palabras que sean de poca importancia en dicho documento una vez leído después de su estandarización. Mediante un StopWord que permite la eliminación de dichas palabras.	
Observaciones: El proceso de estandarización debe eliminar las palabras: él, http, www, ésta, éstas, éste, éstos, última, últimas, último, últimos, a, añadió, aún, aun, actualmente, adelante, además, afirmó, agregó, ahí, ahora, al, algún, algo, alguna, algunas, alguno, algunos, alrededor, ambos, ante, anterior, antes, apenas, aproximadamente, aquí, así, aseguró, aunque, ayer, bajo, bien, buen, buena, buenas, bueno, buenos, cómo, cada, casi, cerca, cierto, cinco, comentó, como, cómo, con, conocer, consideró, considera, contra, cosas, creo, cual, cuales, cualquier, cuando, cuanto, cuatro, cuenta, entre otras.	

Tabla 5 HU-4. Lematizar mediante el algoritmo de Porter todas las palabras de la colección de documentos de entrada.

Historia de Usuario	
Número: HU-4	**Nombre del requisito:** Lematizar mediante el algoritmo de Porter todas las palabras de la colección de documentos de entrada.
Programador: Alejandro Cuellar Monteagudo	**Iteración Asignada:** 2
Prioridad: Alta	**Tiempo Estimado** 15
Riesgo en Desarrollo: Ninguno	**Tiempo Real:** 3
Descripción: El Componente debe hallar la raíz semántica de todas las palabras de los documentos	

Observaciones: Las palabras luego de la lematización deberían quedar de la siguiente forma: abandonadas abandon, abaratar abarat, abordaban abord, abarroteros abarroter

Tabla 6 HU-5. Construcción de las estructuras de datos definidas en la representación del corpus textual en contribución al Modelo de Espacio Vectorial.

Historia de Usuario		
Número: HU-5	**Nombre del requisito** Construcción de las estructuras de datos definidas en la representación del corpus textual en contribución al Modelo de Espacio Vectorial.	
Programador: Alejandro Cuellar Monteagudo		**Iteración Asignada:** 2
Prioridad: Alta		**Tiempo Estimado** 12
Riesgo en Desarrollo: Ninguno		**Tiempo Real:** 3
Descripción: El componente debe crear un objeto de datos con la colección de documentos filtrada		
Observaciones: Luego del proceso de filtrado y lematización de las palabras el sistema debe construir un objeto binario capaz de almacenar los documentos filtrados y almacenados.		

2.4.3 Tiempo de ejecución del proyecto

El tiempo de ejecución del proyecto es una medida que tiene el equipo de desarrollo para completar las historias de usuario en una determinada iteración. Dicha medida se calcula totalizando los puntos estimados de las historias de usuario realizadas en una iteración. En este sentido un punto, equivale a una semana ideal de programación donde se trabaje sin interrupciones. 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 7 presenta la estimación de esfuerzo por historia de usuario propuesta para el desarrollo de la aplicación:

Tabla 7 Estimación por historia de usuario

Número	Título de Historia de Usuario	Estimación en Puntos
HU-1	Estandarizar mediante UTf-8 la colección de documentos de entrada	4
HU-2	Leer la colección de documentos codificados en UTF-8	2
HU-3	Aplicar el filtrado de palabras no relevantes a la colección de entrada	1
HU-4	Lematizar mediante el algoritmo de Porter todas las palabras de la colección de documentos de entrada	3
HU-5	Construcción de las estructuras de datos definidas en la representación del corpus textual en contribución al Modelo de Espacio Vectorial.	3

2.4.4 Iteraciones

El desarrollo del software se divide en etapas para facilitar su realización, dichas etapas se nombran iteraciones. Para cada iteración se define un número determinado de historias de usuario a implementar y al final de cada iteración se obtiene como resultado la entrega de las funcionalidades correspondientes. 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 8 muestra el plan de iteraciones propuesto por el equipo de desarrollo.

Tabla 8 Plan estimado de duración de las iteraciones

Iteración	Título de Historia de Usuario	Duración Total
1	Estandarizar colección de documentos.	31
2	Leer la colección de entrada, Filtrar colección de entrada Realizar stemming a la colección de entrada, Almacenar la colección de documentos	42
Total		1

El plan de iteraciones mostrado en la tabla 8 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 componente informático a desarrollar, posee una entrada y salida determinada consecuente con la iteración contigua. Como se observa en la figura 2.1 el modelo de la propuesta de solución cuenta con 2 etapas principales que culminan en una versión operativa del mismo.

2.4.5 Plan de entrega

El plan de entrega consiste en establecer conjuntamente el equipo de desarrollo y el cliente la duración y fecha de entrega de cada iteración hasta lograr el producto final. En este plan se detalla la fecha fin de cada iteración, mostrando una versión desarrollada del producto en ese momento, hasta lograr el producto final en la fecha establecida. 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 9 muestra el plan de entrega definido por el equipo de desarrollo.

Tabla 9 Plan de entrega de las iteraciones

Iteraciones	Fecha de entrega
Iteración 1	15 de marzo del 2018
Iteración 2	15 de mayo del 2018

El plan de entrega mostrado en la tabla 9 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 componente debe ser capaz de realizar la estandarización de la colección de entrada. Una vez culminado este proceso, para la segunda entrega el componente debe filtrar la colección de entrada. Para la tercera entrega el componente debe realizar el stemming a dicha colección luego del filtrado de palabras. En la entrega final el producto debe ser completamente funcional, permitiendo el almacenamiento de la colección.

2.5 Diseño de la propuesta de solución

El diseño se encuentra en el núcleo técnico de la ingeniería de software. Una vez que se analizan y especifican los requisitos del software, esta es la primera de las tres actividades técnicas que se requieren para construir y verificar el software.(PRESSMAN 2002a) En esta sección se realiza la descripción de la arquitectura de software y los patrones de diseño aplicados a la propuesta de solución. Se presenta el diagrama de clases persistentes del componente a desarrollar y se expone brevemente en qué consisten las tarjetas clase-responsabilidad-creador, así como la descripción mediante el uso de dichas tarjetas de las principales clases del sistema.

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

2.5.1 Arquitectura de software

La Arquitectura de Software se refiere a las estructuras de un sistema, compuestas de elementos con propiedades visibles de forma externa y las relaciones que existen entre ellos. Los elementos pueden ser entidades que existen en tiempo de ejecución (objetos, hilos), entidades lógicas que existen en tiempo de desarrollo (clases, componentes) y entidades físicas (nodos, directorios). Por otro lado, las relaciones entre elementos dependen de propiedades visibles (o públicas) de los elementos, quedando ocultos los detalles de implementación. Finalmente, cada conjunto de elementos relacionados de un tipo particular corresponde a una estructura distinta, de ahí que la arquitectura está compuesta por distintas estructuras.(Ingesoft2 2014)

2.5.2 Arquitectura N capas

La arquitectura de N-Capas es un estilo más que existe en la programación, es importante en muchos aspectos principalmente porque nos permite a los desarrolladores separar la lógica de Acceso a Datos (Capa de Datos) de la lógica del Negocio (Capa de Negocio) y a su vez de la lógica de Diseño (Capa de Presentación). Cualquier capa puede ser modificada o reemplazada y el sistema debe seguir funcionando normalmente. Todo esto tiene su origen del refrán “Divide y Vencerás” en donde logramos encontrar una solución óptima a un problema complejo dividiendo este en partes más simples.(Nicholls 2013)

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 Fig. 3 se ilustra la arquitectura de la propuesta de solución.

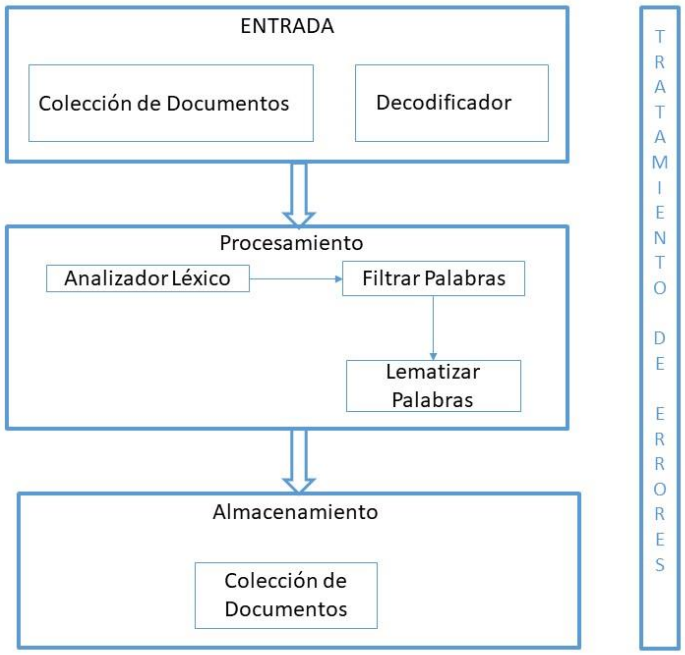


Fig. 3 Arquitectura de la propuesta de solución

Como se muestra en la Figura 3, el diseño de la arquitectura está divido en tres capas: entrada, procesamiento y representación. En la primera capa se prepara la colección de documentos atendiendo a la codificación de los textos. El decodificador selecciona los documentos siguiendo los estándares de codificación permitidos y estos son enviados a la capa intermedia. La capa de procesamiento es la responsable de las operaciones de la aplicación, en ella se realizan los distintos tipos de filtros que son entregadas a la capa de almacenamiento. Dicha capa es la encargada de almacenarlos datos de que devuelve la capa intermedia 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.

2.5.3 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.(PRESSMAN 2002a) 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 Fig. 4 ilustra el diagrama de clases que representa la estructura estática del componente, 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.

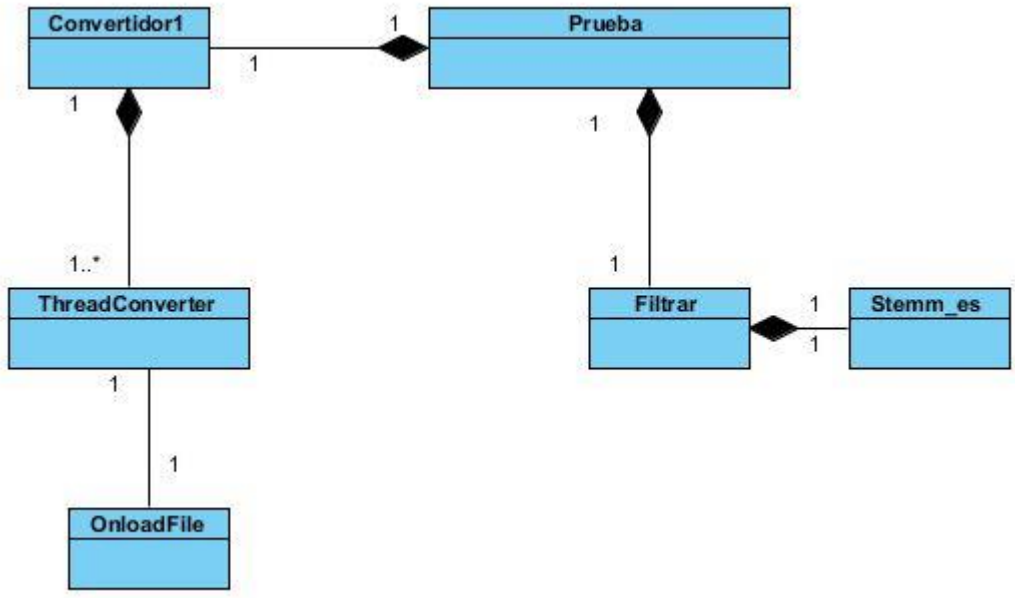


Fig. 4 Diagrama de clases UML

El componente a implementar cuenta con una jerarquía de clases asociada a las tareas necesarias para la representación del corpus textual. Dicha jerarquía está formada por la clase principal Prueba que se encarga de crear y controlar la ejecución del flujo principal del componente. La misma tiene una relación de asociación con la clase Filtrar, una asociación por composición con la clase Convertidor1. La clase Filtrar tiene la responsabilidad de la lectura y escritura de los ficheros.

La clase ThreadConverter se encarga del proceso de lectura y filtrado de la colección de entrada. Esta se relaciona con Convertidor1 a través de una asociación por composición y con OnloadFile mediante una relación de uso.

A cada documento se le aplican una serie de filtros para depurar su contenido, eliminando caracteres extraños, palabras nulas y estandarizando el mismo a minúsculas. De estas tareas se encarga la clase Filtrar de conjunto con Stemm_es con las que posee relaciones de asociación por composición.

2.5.4 Patrones de Diseño

Los patrones de diseño son soluciones para problemas típicos y recurrentes que enfrentan los programadores al desarrollar una aplicación. Son soluciones probadas y documentadas que explican cómo resolver un determinado problema bajo determinadas circunstancias. Por lo que constituyen una guía para el rápido desarrollo de software.(Rubenfa 2014) 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)

Los patrones de asignación de responsabilidades 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.(Gutierrez 2007) 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.

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.(Gutierrez 2007) 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 Fig. 5 muestra el uso de este patrón.

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

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

Fig. 5 Implementación del patrón Experto

Como se observa en la Figura 5 la clase Decoder es la encargada de la decodificación de la colección de entrada. Esta cuenta con toda la información necesaria para la implementación del método getEncoding () encargado de obtener la codificación de los documentos de entrada.

Creador

El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos. El propósito fundamental es encontrar un creador que se debe conectar con el objeto producido en cualquier evento.(Gutierrez 2007) Por tanto, este patrón pretende lograr un bajo acoplamiento (patrón que se describe más adelante) lo que supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización. La Fig. 6 muestra el uso de este patrón.

```
private void analyze() throws FileNotFoundException  {
File file=new File("E:\\Escuela\\TESIS\\version de la tesis 2\\JavaApplication5\\salida\\"+archivos[e].getName());
PrintWriter pw = new PrintWriter(file);
  String word =jTextAreal.getText();
String [] palabras={"él", "http", "www", "ésta", "éstas", "éste", "éstos", "última", "últimas", "último", "últimos",
String [] mama=word.split(" ");
String [] papa=new String[mama.length];
int a=mama.length;

for (int i =0;i<a;i++){
     for(int l=0;l<palabras.length;l++)
     {
         if (mama[a-1].equals(palabras[l]))
               a=a-1;
         if(mama[i].equals(palabras[l]))

             mama[i]="" ;
     }
    if (mama[i].length()>=1) {
  mama[i]=mama[i].replaceAll("[^A-Za-z0-9]", "");
 mama[i]= mama[i].trim();

 String raiz= es.stemm(mama[i]);
 papa[i]=raiz;

 pw.print(" "+papa[i]);
```

Fig. 6 Implementación del patrón Creador

Como se observa en la Figura 6 la clase analyze es la encargada de la creación de la colección de documentos una vez filtrada la colección de entrada.

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.(Gutierrez 2007) Este patrón permite la obtención de clases fácilmente mantenibles y reutilizables durante el proceso de desarrollo de software. La Fig. 7 muestra el uso del mismo.

```
public class Stemm_es {

    public Stemm_es() {
    }

    /** Determina si el caracter es una vocal ...3 lines */
    private boolean is_vowel(char c) {...4 lines }

    /** Retorna la posicion de la primera vocal encontrada ...3 lines */
    private int getNextVowelPos(String word, int start) {
        int len = strlen(word);
        for (int i = start; i < len; i++) {
            if (is_vowel(word.charAt(i))) {
                return i;
            }
        }
        return len;
    }
```

Fig. 7 Implementación del patrón Alta Cohesión

En la Figura 7 se muestra el método getNextVowelPos de la clase Stemm_es. Este método se encarga de retornar la posición de la primera vocal encontrada mediante el método is_vowel el cual determina si un caracter es una vocal y strlen () el cual devuelve el tamaño de la palabra.

Patrones de Concurrencia

Son los patrones destinados a resolver problemas que surgen en la sincronización de recursos compartidos y tareas que interactúan concurrentemente. En estos ambientes se debe asegurar que la integridad de los datos de los recursos compartidos se garantice y que las tareas que acceden o interactúan entre si no entren en deadlock o starvation. El primero de los problemas ocurre cuando todas las tareas están esperando por algún evento sin que ninguna de las partes pueda proseguir. El segundo inconveniente se produce cuando una tarea espera por un recurso que, si bien es liberado, nunca le es asignado.(ERDÖDY 2006)

Por tal motivo aplicar patrones de concurrencia dentro de la programación permite que el componente 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 analiza el patrón Scheduler, la aplicación del mismo se describe en la sección 2.6.1.

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.(ERDÖDY 2006)

2.6 Tarjetas Clase-Responsabilidad-Creador

La utilización de tarjetas clase-responsabilidad-creador (CRC) es una técnica de diseño que consiste en hacer, mediante determinadas tarjetas, un inventario de las clases que se necesitan para implementar el sistema y la forma en que interactúan.(LETELIER 2006) De este modo se pretende facilitar el análisis y discusión de las mismas por parte del equipo de proyecto con el objetivo que el diseño sea lo más simple posible verificando las especificaciones del sistema.

Las tarjetas CRC contienen una serie de campos que deben ser llenados por el equipo de desarrollo, ellos son(ECHEVERRY TOBÓN, L. M.; DELGADO CARMONA 2007):

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

Las tablas 10 y 11 describen mediante tarjetas CRC las clases más importantes que intervienen en el desarrollo de la aplicación:

Tabla 10 Tarjeta CRC . Clase convertidor1

Nombre de la Clase: Convertidor1
Responsabilidades de la Clase: Convertidor1 se encarga de estandarizar la colección de entrada atendiendo a su formato y codificación. También invoca a las clases encargadas de leer la colección de entrada y aplicar la concurrencia.
Colaboradores de Clase
OnloadFile, ThreadConverter

Tabla 11 Tarjeta CRC. Clase Filtrar

Nombre de la Clase: Filtrar

Responsabilidades de la Clase:
Filtrar se encarga de aplicar varios filtros a la colección de entrada. También invoca a la clase encargada de lematizar la todas las palabras de la colección de entrada.
Colaboradores de Clase
Stemm_es

2.7 Implementación de la Propuesta de Solución

En esta sección se describe implementación de la propuesta de solución. Para ello se detalla el algoritmo implementado para la representación del corpus textual de los documentos digitales, así como las estructuras de datos y los patrones de diseño utilizados durante la codificación.

El algoritmo desarrollado inicialmente realiza una estandarización de los documentos de la colección de entra convirtiendo los documentos a formato texto plano y codificación UTF-8. Este proceso se realiza mediante las librerías Apache POI las cuales permiten leer documentos del paquete ofimático Microsoft Office a través del lenguaje de programación Java. Posterior a este proceso el algoritmo se encarga de leer la colección de texto que se encuentra en formato texto plano codificación UTF-8 donde el texto de un documento se guarda en una variable de tipo String, luego se separan las palabras de ese texto a través del método Split(" ") donde el método recorre el String y cada vez que detecte un espacio lo que esta antes del espacio lo elimina del String y lo agrega a una posición de un arreglo. Después de este proceso se procede a eliminar todos los caracteres que no sean letras y números de cada una de las palabras de la colección. A continuación, se eliminan un conjunto de palabras que pertenecen a StopWord que son palabras que por sí solas no significan nada o sea luego de que se eliminen esas palabras las oraciones y los párrafos siguen manteniendo su idea central. Después se procede a lematizar con el algoritmo de Porter las palabras que no pertenezcan a StopWord. El algoritmo de Porter se utiliza para realizar Stemming. El algoritmo lee un archivo, toma una serie de caracteres y de esa serie, una palabra; luego la valida verificando que todos los caracteres involucrados sean letras, de ser así, aplica Stemming sobre ella. A continuación del proceso de lematizar las palabras el lema resultante se guarda en un archivo binario.

La tabla 12 muestra el pseudocódigo del algoritmo desarrollado para el almacenamiento de documentos digitales textuales. Se asume que comienza su ejecución a partir de una colección de documentos que contengan texto plano escrito en español. Dicha colección constituye la entrada al algoritmo propuesto.

Tabla 12 Pseudocódigo. Algoritmo para la representación de corpus textual

Algoritmo para la representación de corpus textual		
1	Inicio	
2		CD ← **Entrada**(Colección de documentos)
3		SW ← {Stop Word del idioma, definido para el algoritmo}

```
4           O ← {Objeto de Datos, inicia en vacío}
5           Para cada D de CD hacer
6                   D ← StopWord(SW,D)
7                   D ← Stemming(D)
8           O ← Palabra(D)
9           Salida(O)
10      Fin
```

Entre las líneas 2 y 4 del pseudocódigo mostrado se inicializan las variables y estructuras de datos necesarias para la corrida del algoritmo. Inicialmente CD recibe la colección de documentos de entrada a procesar y SW contiene el Stop Word (palabras nulas) con los términos definidos para la corrida del algoritmo. En la línea 4 se crea el objeto para el almacenamiento de los documentos de entrada. A partir de la línea 5 comienza la corrida del algoritmo, donde para cada D "documento" de la colección se realizan las siguientes operaciones:

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

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

En la línea 8 el mecanismo encargado de la construcción del objeto se encarga de insertar todas las palabras nuevas en el objeto O**.**

2.7.1 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. Esto permite que arquitecturas multiprocesadores ejecuten al menos tantas tareas como cantidad de procesadores posean.(Drake 2015)

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.

Implementación de la concurrencia

En el componente desarrollado se utilizó la concurrencia con el objetivo de aprovechar la mayor cantidad de recursos disponibles que posee un ordenador. Logrando así una mejora en los tiempos de respuesta del algoritmo desarrollado. Para esto se detectan aquellas situaciones algorítmicas que se puedan ejecutar de manera simultánea sin comprometer la efectividad de la salida esperada. A partir de este punto la utilización de múltiples hilos permite acceder a la colección de documentos a procesar desde varios procesos. Los hilos en el componente desarrollado se crearon de dos maneras la primera en el convertidor de documentos utilizando implements runneable de esta forma se evita tener que especificar qué acción debe realizar cada hilo mediante el método run() con la acción que va a realizar todos los hilos. El proceso de filtrar y lematizar las colecciones de texto se implementa utilizando extends thread de esta forma se puede especificar qué acción debe realizar cada hilo de manera independiente evitando así las secciones críticas. Las secciones críticas son regiones de código a las que no puede acceder más de un proceso o hilo al mismo tiempo.(Aguilera 2015) En el componente desarrollado las principales zonas críticas son: la colección de documentos a procesar y la escritura de archivos. Si más de un proceso accede a estas zonas los datos se duplicarían y se corromperían. Para casos como los expuesto anteriormente existen soluciones tales como semáforos o bloqueo de procesos, en el algoritmo desarrollado no se utilizaron porque esto provocaría inanición de procesos que no es más que cuando un proceso muy lento bloquea un recurso compartido y hace esperar a otro proceso que se ejecuta más rápido para acceder a este recurso, lo que provocaría que hubiesen hilos o procesos detenidos hasta que el resto terminaran lo que traería como consecuencia que el tiempo de respuesta del algoritmo implementado aumente.

El algoritmo descrito en la sección 2.8 evalúa el factor tiempo para la representación del corpus textual de la colección de entrada. La figura 8 muestra dos corridas del algoritmo en cada instante. La colección de entrada varía desde 500 hasta 2000 documentos para cada par de corridas.

Fig. 8 Tiempo de corrida del algoritmo

Como se observa en la gráfica, el tiempo de corrida secuencial es mayor en todos los casos que la corrida concurrente. Esto se debe a que a medida que aumenta el tamaño de la colección aumenta el número de operaciones secuenciales que debe realizar el componente. Para el algoritmo implementado se fija la concurrencia en la construcción de las estructuras de datos. Esto se realiza de manera que una arquitectura multiprocesador permite construir simultáneamente un máximo de estructuras por vez.

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

```
public class Main extends Thread {

    private int conv = 0;
    private int move = 0;

    @Override
    public void run() {
        //Se crea un objeto con la referencia de todos los documentos a procesar
        OnloadFile onloadFile = OnloadFile.getOnloadFile();
        onloadFile.load("D:\\Tesis\\Aplicación\\documentos\\.doc.docx\\");

        int[] split = onloadFile.Split();

        int s0 = split[0];
        int s1 = s0 + split[1];
        int s2 = s1 + split[2];
        int s3 = s2 + split[3];

        ThreadConverter threadConverter0 = new ThreadConverter(0, s0);
        ThreadConverter threadConverter1 = new ThreadConverter(s0, s1);
        ThreadConverter threadConverter2 = new ThreadConverter(s1, s2);
        ThreadConverter threadConverter3 = new ThreadConverter(s2, s3);

        Thread thread0 = new Thread(threadConverter0);
        Thread thread1 = new Thread(threadConverter1);
        Thread thread2 = new Thread(threadConverter2);
        Thread thread3 = new Thread(threadConverter3);

        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);
        }
```

Fig. 9 Implementación del patrón Scheduler

La clase Main que se muestra en la Figura 9 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.

2.8 Conclusiones del Capítulo

En este capítulo se elabora la propuesta de solución que aclara el proceso de representación de la información de acuerdo a las especificaciones analizadas. A partir de esto se describieron los principales artefactos relacionados con el análisis, diseño e implementación de la propuesta de solución. Para el cumplimiento de este objetivo se elaboraron los instrumentos correspondientes a la etapa de planeación, tales como: historias de usuario, tiempo de ejecución, plan de iteraciones y el plan de entrega al cliente. En la segunda sección se especificó la arquitectura de software, el modelado UML, patrones de diseño y tarjetas clase-responsabilidad-creador. Finalmente se expuso el algoritmo implementado, así como las estructuras de datos utilizadas para el apoyo a la representación de la información.

En la etapa de planeación se describieron las historias de usuario, se calculó y planificó el tiempo de ejecución en semanas de trabajo y se confeccionó el plan de entrega por iteraciones, quedando establecido un total de 13 semanas de trabajo con dos iteraciones.
Para el diseño de la propuesta de solución se empleó una arquitectura N capas, la misma quedó estructurada en tres capas: Entrada, Procesamiento y Almacenamiento. Se confeccionó además el diagrama de dominio, diagrama de clases y se describieron los patrones de diseño, así como las tarjetas clases-responsabilidad-creador.

En la implementación de la propuesta de solución se presentó el algoritmo a partir de su pseudocódigo y se describieron las principales características del mismo. Se describió y analizó el uso de la concurrencia en dicho algoritmo.

CAPÍTULO 3: COMPROBACIÓN DEL FUNCIONAMIENTO DE LA PROPUESTA DE SOLUCIÓN

Introducción

La fase e implementación de un sistema es de las más importantes para el desarrollo de un software. Esta fase materializa en forma de código la arquitectura, todos los artefactos y descripciones definidos en la anterior etapa de análisis y diseño con el objetivo de realizar el producto final que desea obtener el cliente. Al realizar pruebas a un software se garantiza que el mismo cumpla con los requerimientos y funcionalidades que pide el cliente. Estas pruebas se llevan a cabo en la etapa de validación donde se realizan un conjunto de pruebas, cada una con un objetivo específico.

3.1 Comprobación del funcionamiento del algoritmo implementado

El proceso de filtrado de la colección de entrada en el algoritmo implementado constituye uno de los procesos de mayor importancia ya que si el algoritmo no realiza una correcta estandarización convirtiendo los documentos de la colección de entrada a formato TXT y codificación UTF-8 el componente ni tendría documentos sobre los cuales trabajar. La correcta estandarización de la colección de entrada permite al componente funcionar de forma correcta, por tal motivo es necesario comprobar si la estandarización de la colección de entrada realizada por el componente implementado se realiza de forma correcta. Para ello es necesario confeccionar una colección de documentos a estandarizar, a partir de esto es posible determinar si el resultado emitido por el algoritmo se corresponde con el resultado esperado.

La colección de documentos utilizada cuenta con 3560 documentos donde los documentos están en formato en diferentes formatos doc y docx. El tamaño de los documentos va desde 1Kb hasta 1.5Mb y el tamaño total de la colección de entrada es de 188Mb.

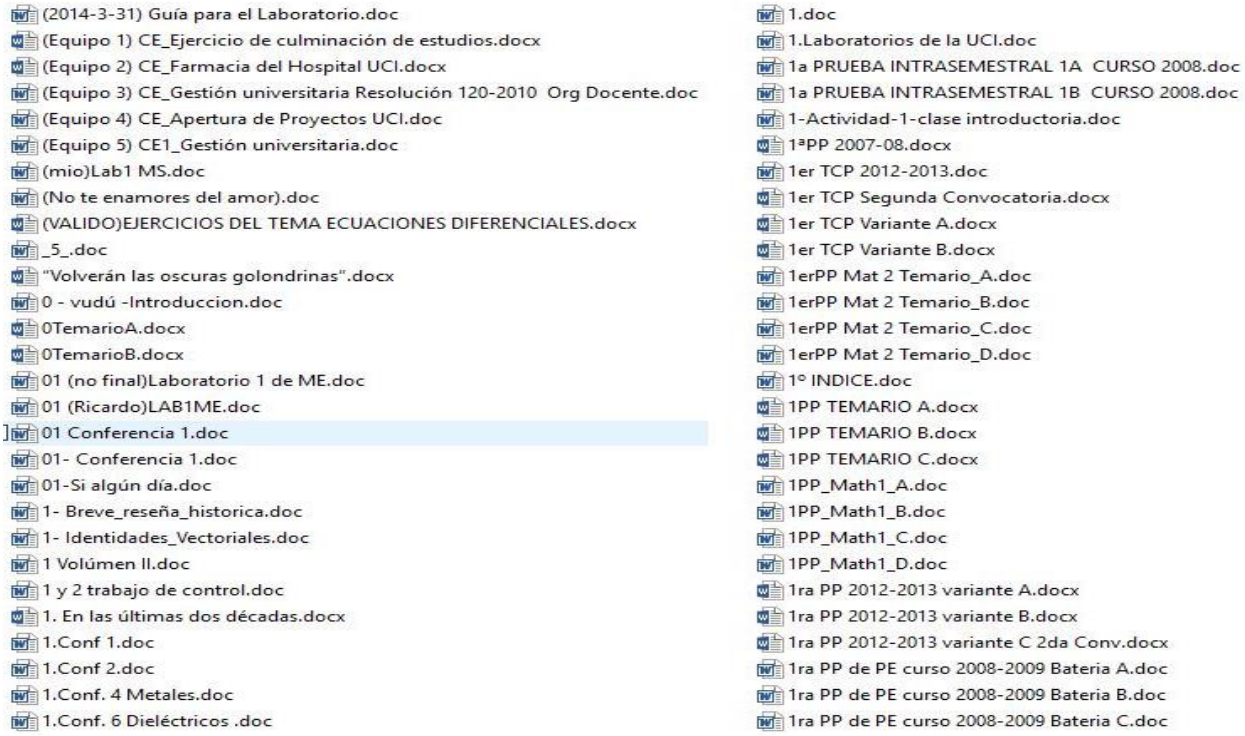


Fig. 10 Colección de Prueba

La figura 10 muestra la colección de prueba a la cual se le va a realizar el proceso de estandarización.

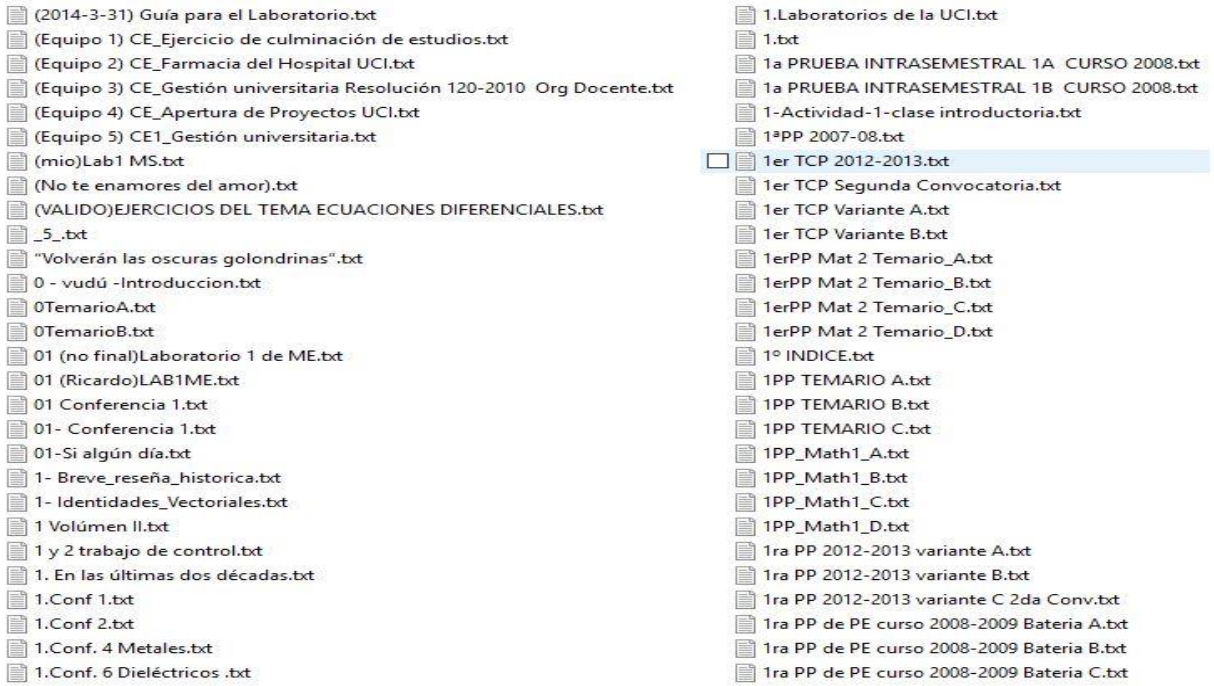


Fig. 11 Salida del Algoritmo

La figura 11 muestra la colección de entra luego del proceso de estandarización donde todos los documentos que el componente convirtió se encuentran en formato TXT y codificación UTF-8

```
Total convertido: 3546
Total no convertido: 14
BUILD SUCCESSFUL (total time: 34 seconds)
```

Fig. 12 Respuesta del sistema

La figura 12 muestra la respuesta del sistema donde de 3560 documentos a estandarizar solo se estandarizaron 3546 y no se pudieron estandarizar 14 documentos el algoritmo arrojo una efectividad del 99.60%. Los documentos que no se estandarizaron tenían la extensión doc, pero su codificación era RTF. El tamaño de la colección luego del proceso de estandarización fue de 30.4Mb.

El proceso de filtrado de la colección de documento constituye la base en la representación del corpus textual de una colección de archivos digitales de entrada. El correcto filtrado de los documentos permite disminuir el tamaño de las colecciones texto reduciendo el costo computacional de la búsqueda sobre la colección de entrada. Por tal motivo es necesario comprobar si el filtrado de la colección realizado por el componente se realiza de forma correcta. Para esto es necesario desarrollar un texto prueba para que el algoritmo lo filtre, a partir de esto es posible comparar si el resultado emitido por el algoritmo se corresponde con el resultado esperado.

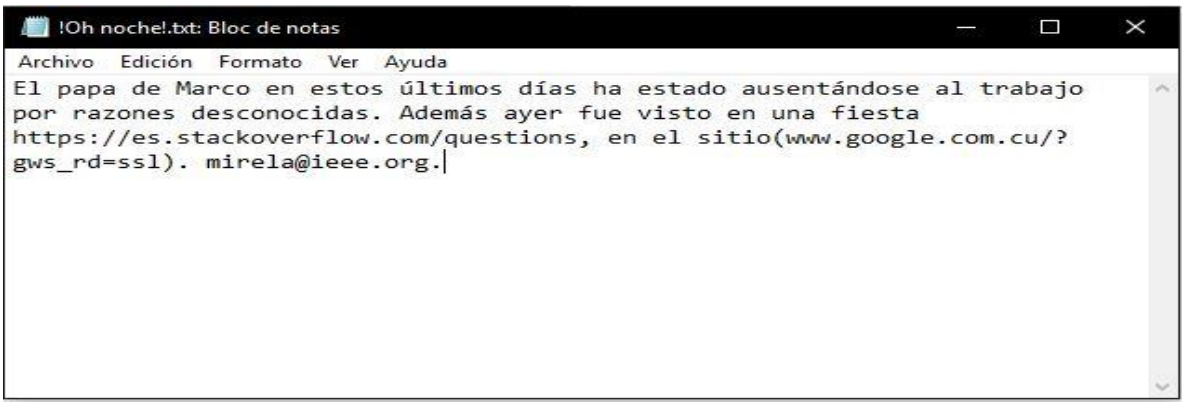


Fig. 13 Texto de Prueba

La figura 13 muestra el texto al cual se le va a realizar el proceso de filtrado donde el componente debe eliminar un conjunto de palabras pertenecientes al StopWord, además debe eliminar todos los caracteres que no sean letras o números, el algoritmo de también tiene que eliminar las direcciones de los sitios web y

las direcciones de correo electrónico. Este proceso debe reducir las colecciones de texto en un 30% o superior.

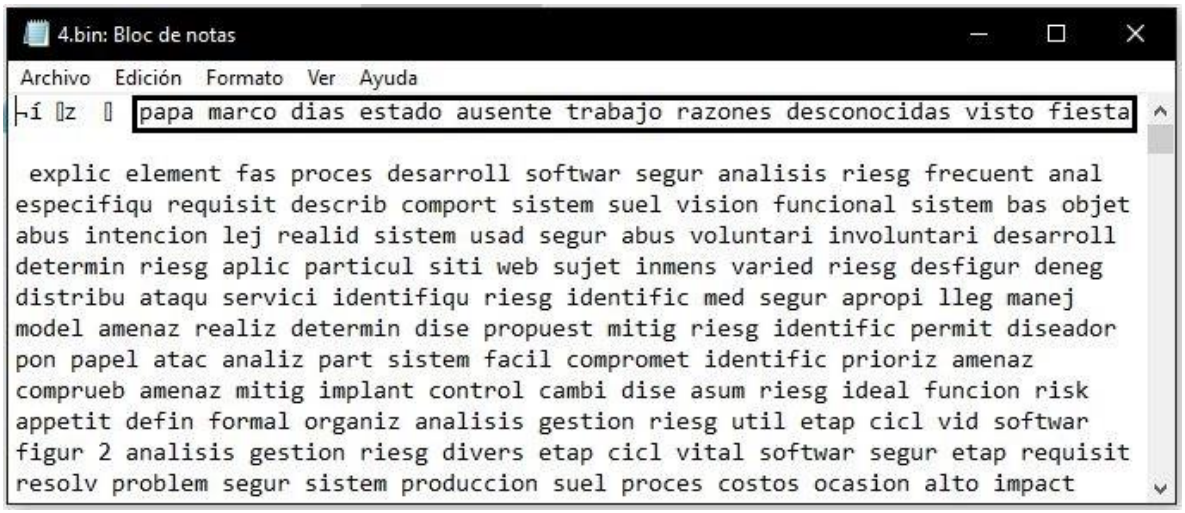


Fig. 14 Resultado de la prueba

La Figura 14 muestra el resultado de la prueba donde el algoritmo es capaz de realizar el filtrado de la colección de manera satisfactoria eliminando todos los elementos que debía eliminar arrojando una efectividad del 100%. En el texto de prueba de 28 palabras solo quedó 10 palabras reduciendo el tamaño del texto en un 64.28%.

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

El listado de lexemas utilizado se encuentra en el diccionario de lexicografía práctica del Ministerio de Educación, Cultura y Deporte del gobierno de España. La colección asciende a un total de 28 389 palabras con sus correspondientes lexemas. Con el objetivo de sintetizar en este documento la representación completa de los 28 389 vocablos involucrados en el proceso de prueba, solo se ejemplifican 108. La tabla 11 muestra la colección de prueba formada por la dupla palabra-lexema.

Tabla 13 Colección de Prueba

Nro	Palabras	Raices	Nro	Palabras	Raices	Nro	Palabras	Raices
1	a	a	**37**	abatida	abat	**73**	abogada	abog
2	aarón	aaron	**38**	abatido	abat	**74**	abogadas	abog
3	abaco	abac	**39**	abatidos	abat	**75**	abogadete	abogadet
4	abajo	abaj	**40**	abatimiento	abat	**76**	abogadillo	abogadill
5	abandera	abander	**41**	abatió	abat	**77**	abogado	abog
6	abandona	abandon	**42**	abatir	abat	**78**	abogados	abog
7	abandonada	abandon	**43**	abatirnos	abat	**79**	abolición	abolicion
8	abandonadas	abandon	**44**	abatirse	abat	**80**	abolirá	abol
9	abandonado	abandon	**45**	abba	abba	**81**	abominan	abomin
10	abandonados	abandon	**46**	abbud	abbud	**82**	abono	abon
11	abandonamos	abandon	**47**	abc	abc	**83**	abonos	abon
12	abandonan	abandon	**48**	abdicación	abdic	**84**	aborda	abord
13	abandonar	abandon	**49**	abdicar	abdic	**85**	abordaban	abord
14	abandonarlo	abandon	**50**	abeba	abeb	**86**	abordada	abord
15	abandonaron	abandon	**51**	abedrop	abedrop	**87**	abordadas	abord
16	abandono	abandon	**52**	abel	abel	**88**	abordan	abord
17	abandonó	abandon	**53**	abelardo	abelard	**89**	abordando	abord
18	abaratar	abarat	**54**	aberración	aberr	**90**	abordar	abord
19	abarca	abarc	**55**	aberrante	aberr	**91**	abordará	abord
20	abarcamos	abarc	**56**	abidjan	abidj	**92**	abordarán	abord
21	abarcan	abarc	**57**	abiel	abiel	**93**	abordaron	abord
22	abarcar	abarc	**58**	abierta	abiert	**94**	aborde	abord
23	abarcará	abarc	**59**	abiertamente	abiert	**95**	abordo	abord
24	abarcarán	abarc	**60**	abiertas	abiert	**96**	abordó	abord
25	abarcaría	abarc	**61**	abierto	abiert	**97**	abortados	abort
26	abarcó	abarc	**62**	abiertos	abiert	**98**	abortiva	abort
27	abaroa	abaro	**63**	abigeo	abige	**99**	aborto	abort
28	abarroteros	abarroter	**64**	abimael	abimael	**100**	abortos	abort
29	abarrotó	abarrot	**65**	abióticos	abiot	**101**	abra	abra
30	abastece	abastec	**66**	abismo	abism	**102**	abraham	abraham
31	abastecedora	abastecedor	**67**	ablandar	abland	**103**	abrasivo	abras
32	abastecer	abastec	**68**	ablandó	abland	**104**	abrazarlo	abraz
33	abastecimiento	abastec	**69**	abluciones	ablucion	**105**	abrazarse	abraz
34	abastecimientos	abastec	**70**	abnegación	abneg	**106**	abrazo	abraz
35	abasto	abast	**71**	aboagay	aboagay	**107**	abre	abre
36	abatares	abatar	**72**	abocó	aboc	**108**	abrego	abreg

El resultado obtenido luego de ejecutar el *stemming* en el software desarrollado para la colección de prueba se muestra en la Figura 10.

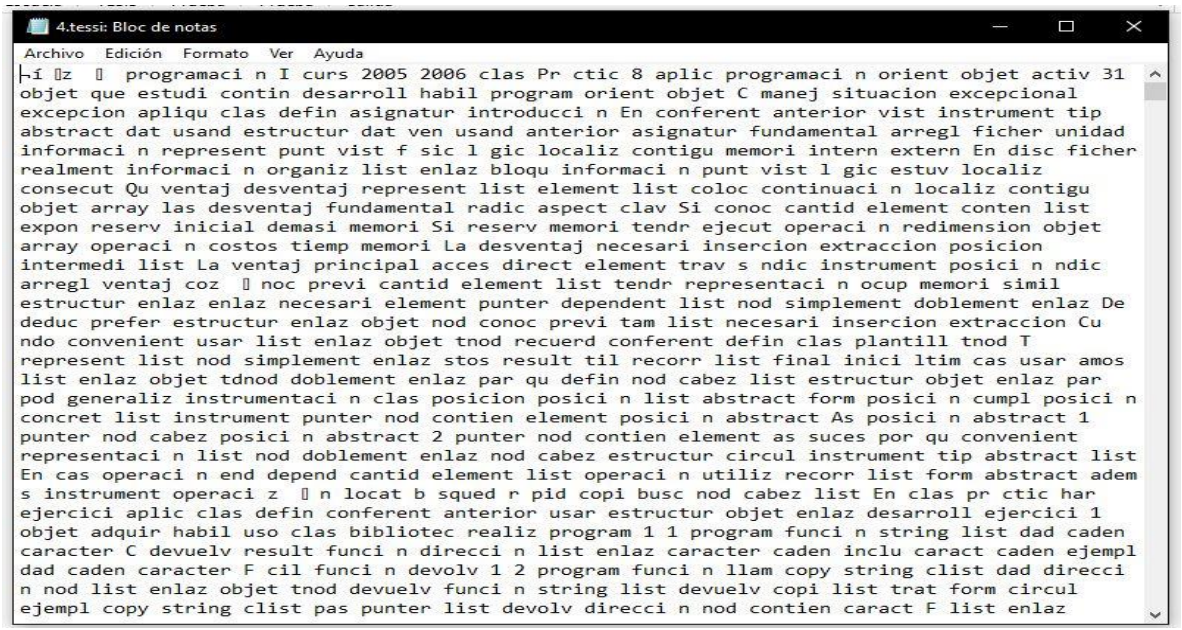


Fig. 15 Salida del algoritmo

Como se observa en la Figura 10 el algoritmo crea un objeto que contiene la colección de entrada donde cada palabra de dicha colección esta lematizada.

- ✓ Para la colección de 28389 palabras se obtienen 28389 coincidencias con 0 errores, para una taza de efectividad del 100 %.

La Figura 11 muestra la salida del software para la colección de prueba.

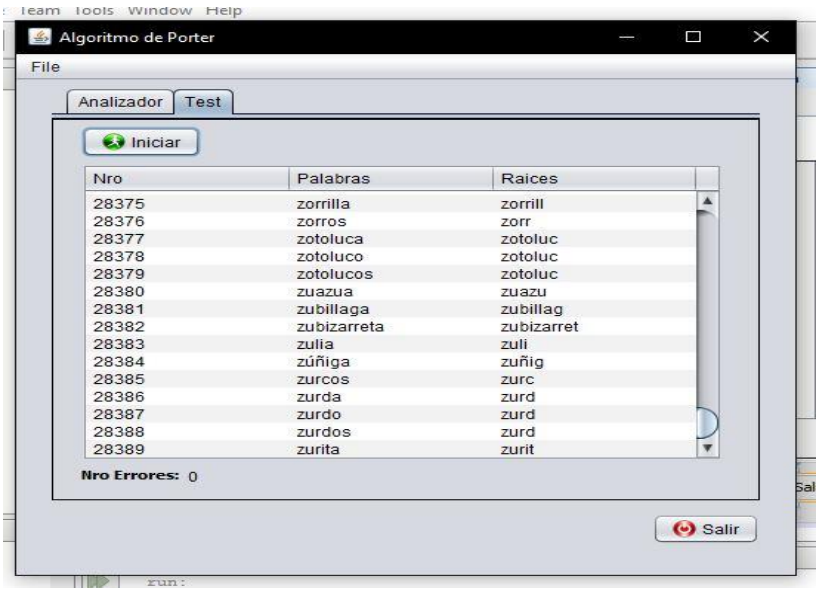


Fig. 16 Resultado de prueba

El algoritmo es capaz de derivar cada palabra de la colección y esta es la misma para cada ocurrencia de la palabra. Por tanto, en términos de representación del corpus textual de los documentos el algoritmo implementado funciona correctamente.

3.2 Comprobación del funcionamiento del software desarrollado

La comprobación del funcionamiento del software desarrollado 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 del sistema se dividen en dos grupos: pruebas unitarias, encargadas de verificar el código y diseñadas por los programadores y pruebas de aceptación destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida por el cliente.(Microsoft 2015) 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; 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.

3.2.1 Estrategia de Pruebas

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. Por tanto, la estrategia de prueba permite definir las técnicas y criterios a tener en cuenta para realizar las pruebas al componente desarrollado.

La estrategia de prueba define:

- ✓ Técnicas de pruebas (manual o automática) y herramientas a ser usadas.
- ✓ Qué criterios de éxitos y culminación de la prueba serán usados.
- ✓ Consideraciones especiales afectadas por requerimientos de recursos o que tengan implicaciones en la planificación.

Los niveles de pruebas que se distinguen son:

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

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

Los métodos de pruebas definidos son:

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

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

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

3.2.2 Pruebas unitarias

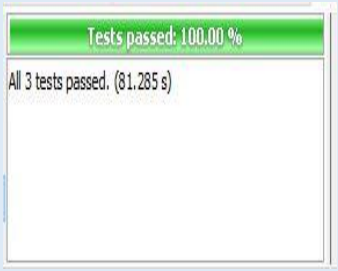
Una prueba unitaria es un método que prueba una unidad estructural de código, por lo general son simples y rápidas de codificar. Las pruebas unitarias son una forma para verificar la estructura y el buen funcionamiento de una parte del código, están dirigidas a probar clases aisladamente y relacionadas con el código y la responsabilidad de cada clase, así como sus fragmentos de código más críticos.(Microsoft 2015) 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 manual.

Pruebas unitarias. Iteración 1

La primera iteración de pruebas unitarias fue realizada sobre las historias de usuario siguientes: HU-1 Estandarizar colección de documentos.

Las tablas 14 muestra el resultado de la ejecución de las baterías de prueba realizada a la clase Convertidor1, generada para la resolución de la historia de usuario HU-1 Estandarizar colección de documentos. En ellas se muestran el método u operación a probar, la entrada proporcionada, los resultados esperados y obtenidos, si funciona correctamente.

Tabla 14 . HU-1 Estandarizar colección de documentos. Clase Convertidor1

Método a probar	Entrada	Resultado esperado	Resultado obtenido	Funciona correctamente	**Observaciones**
Main()	El método recibe como entrada un documento en formato .doc o .docx.	El método debe retornar un documento en formato .txt y codificación UTF-8.	Documento Convertido a .txt y codificado a UTF-8.	Si	

Durante la primera iteración no se detectaron errores en la codificación del método probado, obteniéndose el resultado esperado en la prueba realizadas.

Pruebas unitarias. Iteración 2

La segunda iteración de pruebas unitarias se realizó sobre las historias de usuario siguientes:HU-3 Filtrar colección de entrada, HU4- Realizar stemming a la colección de entrada y HU-5 Construcción de las

estructuras de datos definidas en la representación del corpus textual en contribución al Modelo de Espacio Vectorial.

Las tablas 15, 16 y 17 muestran el resultado de la ejecución de las baterías de prueba realizadas a la clase Filtrar, generada para la resolución de las historias de usuario HU-3 Filtrar colección de entrada, HU-4 Realizar stemming a la colección de entrada y HU-5 Construcción de las estructuras de datos definidas en la representación del corpus textual en contribución al Modelo de Espacio Vectorial. En ellas se muestran los mismos campos descritos anteriormente.

Tabla 15 . HU-3 Filtrar colección de entrada. Clase Filtrar

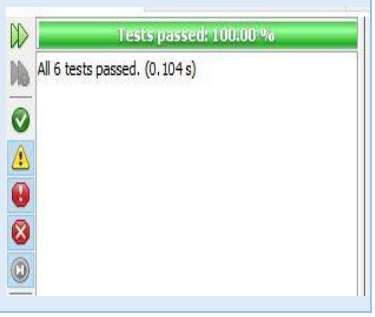
Método a probar	Entrada	Resultado esperado	Resultado obtenido	Funciona correctamente	**Observaciones**
Con()	Recibe como entrada una línea de texto del documento que se esté analizando.	Debe eliminar un conjunto de palabras no relevantes para el documento analizado. recibido	Se obtuvo la línea de texto parcialmente modificada.	Si	

Tabla 16 . HU-4 Lematizar colección de entrada. Clase filtrar

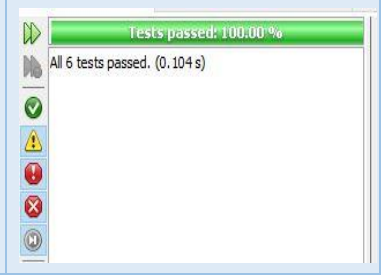
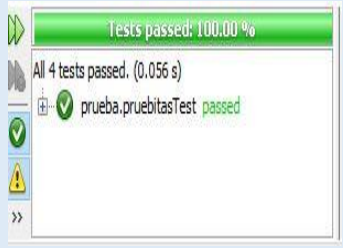
Método a probar	Entrada	Resultado esperado	Resultado obtenido	Funciona correctamente	**Observaciones**
Con()	Recibe como entrada una palabra del texto analizado.	Debe Lematizar cada palabra del texto recibido	Todas las palabras del texto analizado fueron lematizadas.	Si	

Tabla 17 .HU-5 Construcción de las estructuras de datos definidas en la representación del corpus textual. Clase Filtrar

Métodos a probar	Entrada	Resultado esperado	Resultado obtenido	Funciona correctamente	Observaciones
objeto () objeto2() objeto3() objeto4()	Recibe como entrada una palabra del texto analizado luego del proceso de lematización.	Debe almacenar todas las palabras lematizadas del texto analizado	Todas las palabras fuero almacenadas.	Si	

Durante la segunda iteración no se detectaron errores en la codificación de los métodos probados, obteniéndose el resultado esperado en la prueba realizadas.

Como se observa en la Fig. 17 durante el proceso de pruebas unitarias se verificaron un total de 4 operaciones del componente implementado. Durante la primera iteración no se encontró error y en la segunda iteración de dos operaciones no se encontró ningún problema.

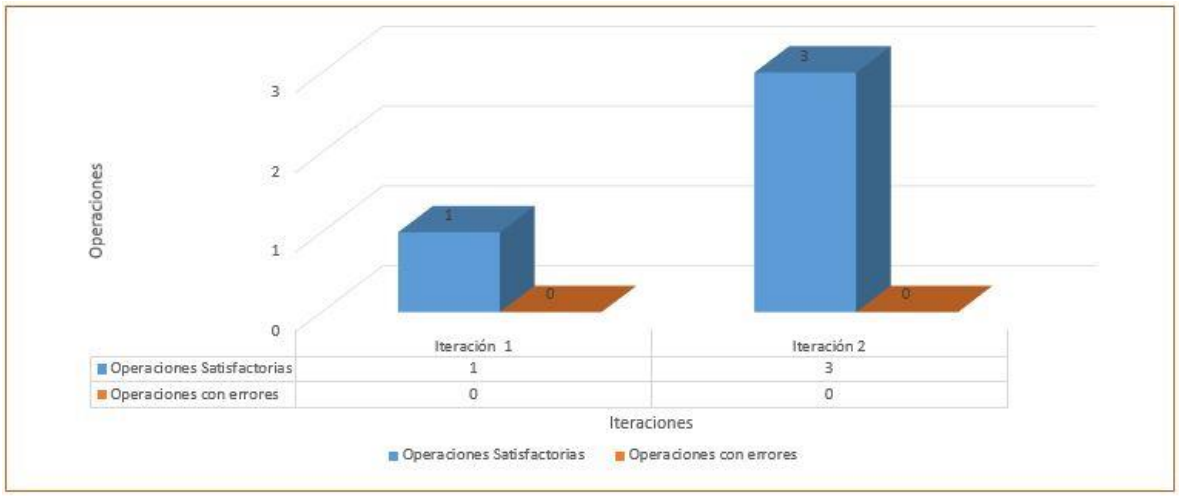


Fig. 17 Pruebas Unitarias

3.2.3 Pruebas de aceptación

Las pruebas de aceptación son las últimas pruebas realizadas donde el cliente prueba el software y verifica que cumpla con sus expectativas. Estas pruebas generalmente son funcionales y se basan en los requisitos definidos por el cliente y deben hacerse antes de la salida a producción. Las pruebas de aceptación son fundamentales por lo cual deben incluirse obligatoriamente en el plan de pruebas de software. Estas

pruebas se realizan una vez que ya se ha probado que cada módulo funciona bien por separado, que el software realice las funciones esperadas y que todos los módulos se integran correctamente.(Training 2017)

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

Pruebas de aceptación. Iteración 1

La tabla 18 muestra los datos obtenidos a partir de la aplicación de las pruebas de aceptación a las historias de usuario HU-1 (por su número de historia).

Tabla 18 Pruebas de aceptación. Iteración 1

Pruebas de Aceptación				
Acción a probar	**Datos de prueba**	**Resultado esperado**	**Resultado obtenido**	**Evaluación de la prueba**
HU-1: el componente debe ser capaz de decodificar la colección de documentos de entrada.	La prueba da inicio con una colección de 2 000 documentos en distintos formatos.	Todos los documentos deben ser convertidos a formato de texto plano con codificación UTF-8.	El componente procesó correctamente los datos de prueba y generó una lista de errores con aquellos que no pudo convertir. El resultado obtenido muestra una precisión del 95 %.	Aceptada

La primera iteración pasa satisfactoriamente la prueba de aceptación.

Pruebas de aceptación. Iteración 2

La tabla 17 muestra los datos obtenidos a partir de la aplicación de las pruebas de aceptación a las historias de usuario HU-2, HU-3, HU-4 y HU-5 (por su número de historia).

Tabla 19 Pruebas de aceptación. Iteración 2

Pruebas de Aceptación

Acción a probar	Datos de prueba	Resultado esperado	Resultado obtenido	Evaluación de la prueba
HU-2: el componente debe leer todos los documentos en formato de texto y codificación UTF-8	La prueba da inicio con 2000 documentos	Todos los documentos deben ser leídos por el componente.	Todos los documentos fueron leídos satisfactoriamente.	Aceptada
HU-3: el proceso de filtrado de los textos está compuesto por dos filtros fundamentales: LowerCaseFilter y StopWord.	La prueba da inicio con 2000 documentos	Para el LowerCaseFilter el componente debe emitir como salida todos los textos en letra minúscula. El StopWord debe eliminar de todos los textos un listado de palabras predefinidas por el cliente.	Para el LowerCaseFilter el componente emite la salida esperada. En el StopWord el componente emite la salida esperada..	Aceptada
HU-4: los textos filtrados deben pasar por un proceso de lematización que reduzca todas las palabras a su raíz semántica	La prueba da inicio con 2000 documentos	El lematizador debe reducir todas las palabras del contenido de los documentos a su raíz semántica	El lematizador reduce todas las palabras del dominio a la raíz semántica.	Aceptada
HU-5: el componente debe almacenar los resultados obtenidos para su posterior uso.	La prueba da inicio con 2000 documentos	El componente debe almacenar las palabras lematizadas en un archivo binario.	El componente almacena todas las palabras lematizadas en un fichero binario.	Aceptada

La segunda iteración pasa satisfactoriamente la prueba de aceptación y culmina el proceso de desarrollo del software. Como se muestra en la Fig. 13 el proceso de pruebas de aceptación se realizó en tres iteraciones. Las tres iteraciones se realizaron de forma satisfactoria.

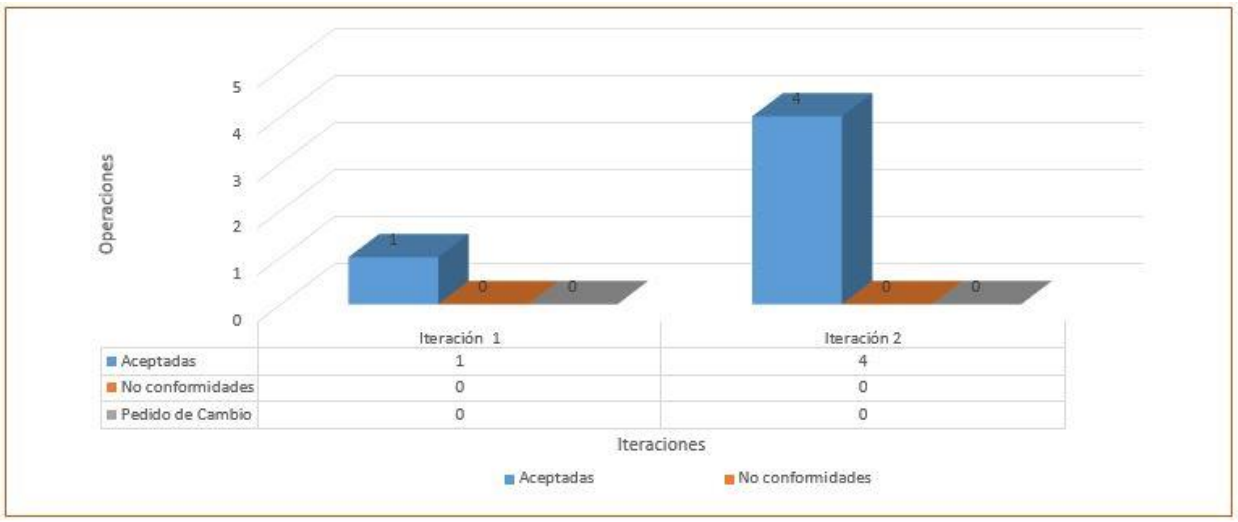


Fig. 18 Pruebas de aceptación

Conclusiones del Capítulo

En este capítulo se realizó la comprobación del algoritmo implementado mediante la aplicación del proceso de estandarización de la colección de entrada, filtrado de la colección de entrada y al *stemming*. La efectividad obtenida para el proceso de estandarización fue de un 99.60% para una muestra de 3560 documentos. Para el proceso de filtrado la efectividad fue del 100% para un texto de prueba de 28 palabras. El lematizador arrojo una efectividad del 100% para una muestra de 108 palabras y para la colección completa se obtuvo una efectividad del 100%. Luego se definió la estrategia de prueba a seguir para la comprobación del funcionamiento del componente, en la cual se decidió la aplicación de pruebas unitarias y de aceptación.

El proceso de prueba a las funcionalidades involucradas se dividió en dos iteraciones, describiendo cada una de las baterías de pruebas realizadas en las tablas elaboradas. Las dos iteraciones se realizaron satisfactoriamente culminando así el proceso de pruebas unitarias a las funcionalidades seleccionadas. Se comprobó de esta forma que cada una de ellas devuelve el resultado esperado acorde a los datos de entrada empleados.

Para la realización de las pruebas de aceptación se dividió el período en dos iteraciones. Se elaboró una tabla por cada iteración donde se describieron las pruebas realizadas, mostrando la acción a probar, los datos de prueba, el resultado esperado y observado, así como la evaluación de la prueba. Las dos iteraciones se realizaron de forma satisfactoria y se culminó el período de pruebas de aceptación. A través de las pruebas descritas anteriormente se comprobó el funcionamiento del algoritmo y el componente desarrollado.

Conclusiones

La presente investigación se centró en el desarrollo de un componente informático para la limpieza de datos en archivos digitales textuales, con la finalidad de facilitar el proceso de clasificación automótica de documentos y la recuperación de información. Durante el avance de la misma se cumplieron los objetivos propuestos y se arribó a las siguientes conclusiones:

- A partir de la elaboración del marco teórico de la investigación se enriqueció los conocimientos acerca de los procesos de limpieza de datos y lematización.
- El estudio realizado como parte de la investigación sirvió de apoyo en la toma de decisiones con vista a un desarrollo eficiente del componente informático a desarrollar.
- El enfoque ágil que propone la metodología AUP en su versión para la UCI fue la guía a seguir para describir todos los procesos y sub-procesos que se debían de ejecutar, además de la selección y el uso de herramientas para el desarrollo de la solución.
- Después de haber estudiado todos los elementos que intervienen en el proceso de recuperación de información fue posible realizar el diseño de la propuesta de solución.
- Se implementó un componente informático para la limpieza de datos en archivos digitales textuales a partir de los requerimientos del software establecidos, haciendo uso de la arquitectura y las tecnologías definidas por el equipo de desarrollo de software de dicho componente informático.
- Se diseñaron y aplicaron las pruebas definidas, probando cada una de las funcionalidades del componente para verificar el correcto funcionamiento del mismo.

Recomendaciones

Luego de haber finalizado la investigación y desarrollo de la propuesta de solución del presente trabajo se recomienda en vista de posibles mejoras:

- La internacionalización del componente desarrollado y la implementación en diferentes idiomas del lematizador.
- Escribir cada cierto tiempo en disco los resultados obtenidos y liberar la memoria para disminuir la carga de trabajo ante colecciones muy grandes de documentos

Referencias Bibliograficas

AGUILERA, I.S.O., 2015. Sincronización de procesos. *Sistemas Operativos* [en línea]. S.l.: s.n., Disponible en: http://repositorio.ub.edu.ar/bitstream/handle/123456789/5256/SOp502_U4_SincroProcesos_2015.pdf?sequence=1.

APACHE, 2016. Apache Lucene. *Apache Software Fundation* [en línea]. [Consulta: 19 enero 2018]. Disponible en: http://apachefoundation.wikispaces.com/Apache+Lucene.

CALENDAMAIA, 2014. GENBETA:dev. *NetBeans* [en línea]. [Consulta: 16 enero 2018]. Disponible en: https://www.genbetadev.com/herramientas/netbeans-1.

CONDE, R., 2005. UNONO. *¿Qué es un corpus y cómo se utiliza?* [en línea]. [Consulta: 30 marzo 2018]. Disponible en: http://articles.unono.net/qué-es-un-corpus-y-cómo-se-utiliza/.

DRAKE, J.M., 2015. PROGRAMACION CONCURRENTE Y DISTRIBUIDA. *Concurrencia con Java: Thread Java* [en línea]. [Consulta: 2 marzo 2018]. Disponible en: https://www.ctr.unican.es/asignaturas/procodis_3_ii/doc/procodis_3_01.pdf.

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

ERDÖDY, D.M.S., 2006. *UN FRAMEWORK PARA LA VISUALIZACIÓN DE PATRONES DE DISEÑO DISTRIBUIDOS Y CONCURRENTES IMPLEMENTADOS CON PROGRAMACIÓN ORIENTADA A ASPECTOS: ACVF(ASPECTUAL COMPONENT VISUALIZATION FRAMEWORK)* [en línea]. S.l.: UNIVERSIDAD DE BUENOS AIRES. Disponible en: http://materias.fi.uba.ar/7500/erdody-tesisdegradoingenieriainformatica.pdf.

GARCERANT, I., 2008. Tecnología y Synergix. *Modelo de Dominio* [en línea]. [Consulta: 3 marzo 2018]. Disponible en: https://synergix.wordpress.com/?s=modelo+de+dominio.

GUTIERREZ, J.A.S., 2007. WordPress. *PATRONES GRASP (Patrones de Software para la asignación General de Responsabilidad).Parte II* [en línea]. [Consulta: 2 marzo 2018]. Disponible en: https://jorgesaavedra.wordpress.com/2007/05/08/patrones-grasp-patrones-de-software-para-la-asignacion-general-de-responsabilidadparte-ii/.

ICTEA, 2018. W-ICTEA. *¿Qué es el lenguaje de programación JAVA?* [en línea]. [Consulta: 23 enero 2018]. Disponible en: http://www.ictea.com/cs/knowledgebase.php?action=displayarticle&id=8790.

INGESOFT2, 2014. wordpress. *Diseño de la arquitectura* [en línea]. [Consulta: 3 marzo 2019]. Disponible en: https://ingenieriasoftware2.wordpress.com/2014/08/05/diseno-de-la-arquitectura/.

ISRAELCCM., 2018. CCM-Comunidad informática. *Lenguajes de programación* [en línea]. [Consulta: 16 enero 2018]. Disponible en: http://es.ccm.net/contents/304-lenguajes-de-programacion.

JULIÁN PÉREZ PORTO, A.G., 2012. Definicion.de. *Concepto de información* [en línea]. [Consulta: 9 enero 2018]. Disponible en: https://definicion.de/informacion/.

JULIÁN PÉREZ PORTO, M.M., 2009. Definicion.de. *Definicion de Datos* [en línea]. [Consulta: 9 enero 2018]. Disponible en: https://definicion.de/datos/.

KYOCERA, 2017. Kyocera. *La limpieza de datos como activo empresarial* [en línea]. [Consulta: 18 mayo 2009]. Disponible en: https://smarterworkspaces.kyocera.es/blog/limpieza-datos-la-empresa/.

LETELIER, P., 2006. Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP). *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP)* [en línea]. [Consulta: 18 mayo 2028]. Disponible en: http://www.cyta.com.ar/ta0502/v5n2a1.htm.

LEXICOON, 2018. Lematizacion. *QUÉ SIGNIFICA LEMATIZACIÓN EN ESPAÑOL* [en línea]. [Consulta: 16 enero 2018]. Disponible en: http://lexicoon.org/es/lematizacion.

MICROSOFT, 2015. Microsoft. *Conceptos básicos de prueba unitaria* [en línea]. [Consulta: 5 abril 2018]. Disponible en: https://msdn.microsoft.com/es-es/library/hh694602.aspx.

NICHOLLS, J.D., 2013. Tips de Desarrollo Web. *Arquitectura N-Capas y LinqToSQL* [en línea]. [Consulta: 3 marzo 2018]. Disponible en: http://www.nicholls.co/blog/post/Arquitectura-N-Capas-y-LinqToSQL.

PICO.DEV, 2016. Blog Bitix. *Ejemplo sencillo de como crear un excel en Java con Apache POI* [en línea]. [Consulta: 2 junio 2018]. Disponible en: https://picodotdev.github.io/blog-bitix/2016/05/ejemplo-sencillo-de-como-crear-un-excel-en-java-con-apache-poi/.

PINO, M., 2004. E-COMS. *Búsqueda y Recuperación de Información* [en línea]. [Consulta: 9 enero 2018]. Disponible en: http://www.mariapinto.es/e-coms/busqueda-y-recuperacion-de-informacion/.

PMOINFORMATICA, 2013. PMOinformatica. *¿Qué son las historias de usuario?* [en línea]. [Consulta: 3 marzo 2018]. Disponible en: http://www.pmoinformatica.com/2013/04/que-son-las-historias-de-usuario-7.html.

PRESSMAN, R.S., 2002a. *Ingenieria de softwate, un enfoque practico*. Quinta. S.l.: s.n.

PRESSMAN, R.S., 2002b. Visual Paradigm. *Ingeniería de Software, un enfoque práctico.* Quinta. S.l.: s.n., ISBN 8448132149.

RESEARCH, P., 2017. Provalis Research. *Un análisis de contenido sumamente avanzado y software de minería de texto con inigualables manejo y capacidades de análisis.* [en línea]. [Consulta: 18 enero 2018]. Disponible en: http://provalisresearch.com/es/products/software-de-analisis-de-contenido/.

RUBENFA, 2014. Genbeta:Dev. *Patrones de diseño: qué son y por qué debes usarlos* [en línea]. [Consulta: 2 marzo 2018]. Disponible en: https://www.genbetadev.com/metodologias-de-programacion/patrones-de-diseno-que-son-y-por-que-debes-usarlos.

SOFTWARE-SHOP, 2018. T-Labs. *T-LAB* [en línea]. [Consulta: 16 enero 2018]. Disponible en: https://www.software-shop.com/producto/t-lab.

TRAINING, L.A., 2017. Los Andes Training. *Que son las pruebas de aceptación* [en línea]. [Consulta: 6 abril 2018]. Disponible en: https://losandestraining.com/2017/08/23/que-son-las-pruebas-de-aceptacion/.

WORDPRESS, 2009. Wordpress. *Que es un archivo digital* [en línea]. [Consulta: 30 marzo 2018]. Disponible en: https://doraduque.wordpress.com/tag/que-es-un-archivo-digital/.

WORDPRESS, 2013. Wordpress. *Entorno de Desarrollo Integrado (IDE).* [en línea]. [Consulta: 16 enero 2018]. Disponible en: https://fergarciac.wordpress.com/2013/01/25/entorno-de-desarrollo-integrado-ide/.

Bibliografía

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

AMBLER, S. W. 2012. Agile Modeling and eXtreme Programming (XP). *Agile Modeling.* [En línea] Ambysoft Inc., 2012. [Citado el: 20 de mayo de 2015.] http://agilemodeling.com/essays/agileModelingXP.htm.

APACHE. Apache Lucene. *Apache Software Fundation.* [En línea] [Citado el: 24 de marzo de 2015.] http://apachefoundation.wikispaces.com/Apache+Lucene.

BOLIVARIANA, U. Programación extrema XP. *Ingenieria de Software.* [En línea] [Citado el: 20 de mayo de 2015.] http://ingenieriadesoftware.mex.tl/52753_XP---Extreme-Programing.html.

CAROLINA MARTÍNEZ, I. 2010. *Modelo Conceptual / Modelo de Dominio.* [Documento] Caracas : Universidad Simón Bolívar, 2010.

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

DRAKE, J.M. Programación Concurrente. *CTR - Computadores y Tiempo Real.* [En línea] [Citado el: 27 de abril de 2015.] http://www.ctr.unican.es/asignaturas/procodis_3_ii/doc/procodis_1_01.pdf.

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

FLORES CUETO, J. J. y BERTOLOTTI ZUÑIGA, C. Diagrama de clases en UML. *Diagrama de clases en UML.* [En línea] [Citado el: 14 de abril de 2015.] http://es.scribd.com/doc/31096724/Diagrama-de-Clases-en-UML#scribd.

GOMEZ, D., ARANDA, E. y FABREGA, J. Programación Extrema. *Universidad del Valle.* [En línea] [Citado el: 20 de mayo de 2015.] http://eisc.univalle.edu.co/materias/WWW/material/lecturas/xp.pdf.

INNOCUO. 2006. Patrones de diseño, la importancia de aprenderlos. *innocuo.* [En línea] 11 de septiembre de 2006. [Citado el: 14 de abril de 2015.] http://blog.innocuo.com/2006/09/patrones-de-diseno-la-importancia-de-aprenderlos/.

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

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

LETELIER, P. 2006. Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP). [En línea] 15 de enero de 2006. [Citado el: 29 de marzo de 2015.] http://www.cyta.com.ar/ta0502/v5n2a1.htm. ISSN 1666-1680.

MARTÍNEZ DE SOUSA, J. 1995. *Diccionario de Lexicografía Práctica.* [Documento] Barcelona : s.n., 1995.

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

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

RESEARCH, P. *Un análisis de contenido sumamente avanzado y software de minería de texto con inigualables manejo y capacidades de análisis.* [En línea] [Citado el: 24 de marzo de 2015.] http://provalisresearch.com/es/products/software-de-analisis-de-contenido/.

UNIVERSO. 2012. Programación extrema: “Metodología para desarrollo ágil de aplicaciones”. *Universo: El periódico de los universitarios.* [En línea] 11 de junio de 2012. [Citado el: 20 de mayo de 2015.] http://www.uv.mx/universo/486/infgral/infgral_15.html.

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

WELLS, D. The Rules of Extreme Programming. *Extreme Programing.* [En línea] [Citado el: 20 de mayo de 2015.] http://www.extremeprogramming.org/rules.html.