Facultad 1

Componente para la categorización semántica de documentos en sistemas de recuperación de información

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

Autor:

David Álvarez Santes

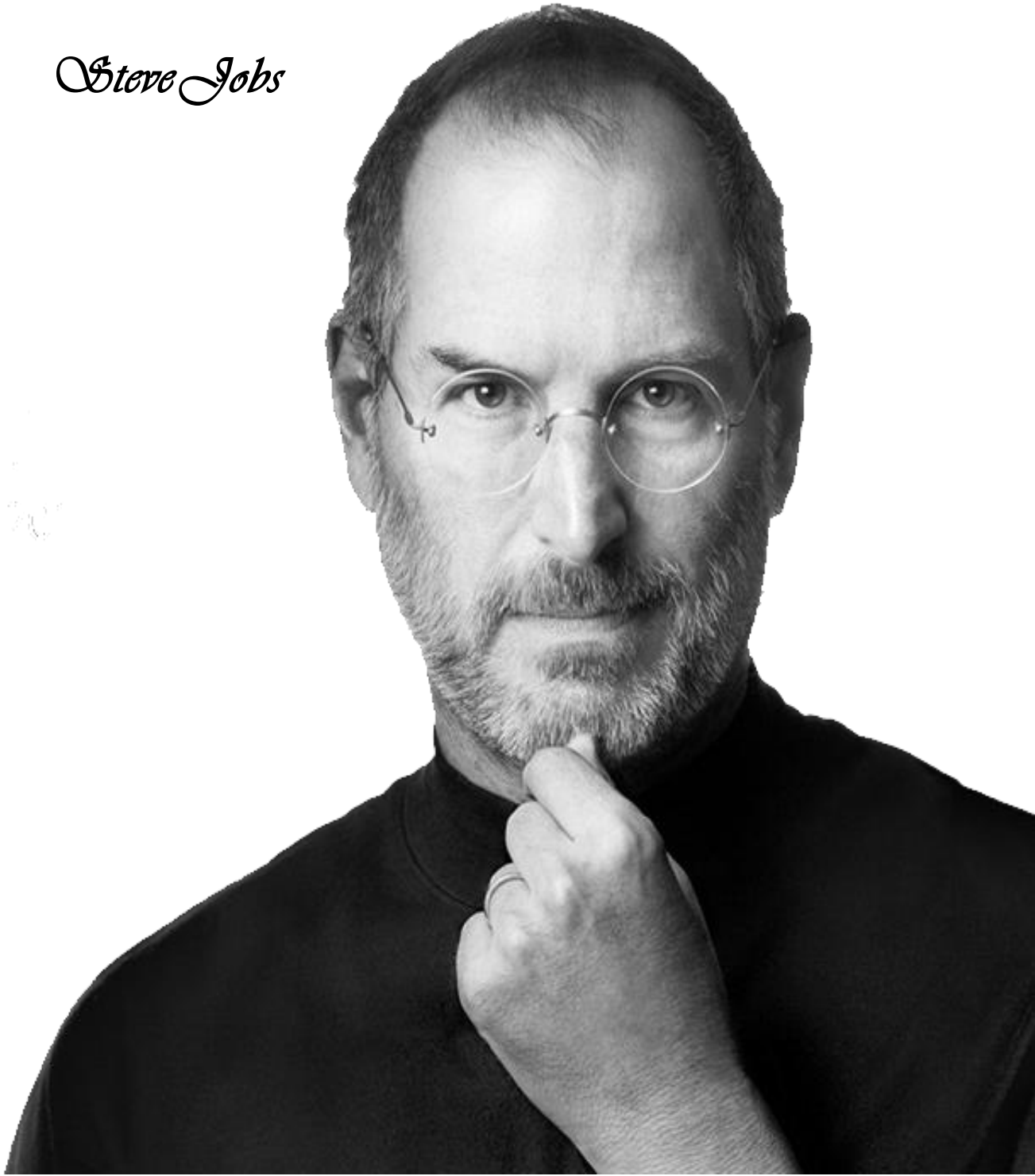
Tutores:

MSc. Delly Lien Gónzalez Hernández
MSc. Hubert Viltres Sala
Ing. Martha Luisa Gala Rodríguez
Ing. Alexander García O'reilly

La Habana, junio, 2018

"Si tú no trabajas por tus sueños, alguien te contratará para que trabajes por los suyos..."

Steve Jobs



Dedicatoria

A mi abuelo, por ser la luz protectora de mi familia.

Por enseñarme que el secreto del viajero no radica en la pasión por el viaje, sino en la certeza del regreso.

Porque, aun después de pasar todos los niveles de enseñanza, no he encontrado mejor profesor.

Por el legado de tu nombre y todas tus enseñanzas, por dejarme sentir tu ausencia cuando estoy a la deriva.

Por este sueño hecho realidad, por el orgullo de saberte mi abuelo.

Por todas las cosas que no llegué a conocer contigo.

Por no dejarme ver tu sonrisa al enseñarte mi título.

Y, porque en algún lugar estás sonriendo.

Agradecimientos

A mis padres por todo el apoyo y educación que me han brindado a lo largo de todos estos años, por mostrarme el camino cuando me sentía perdido. Porque son lo más bello que la vida me ha regalado.

A mi abuela por ser tan consentida, por su preocupación y apoyo en cada momento difícil.

A mi hermana por estar presente, y por todo el apoyo que supo ofrecerme.

A mi novia por la paciencia, y por darme luz en tiempos de oscuridad.

A mi familia por todo el apoyo brindado en todos estos años. Por permitir entrar a la familia a un ingeniero más.

A mis copis del 111 102 por ser faros y guías en cada momento que compartimos. Por aguantarme estos cinco años y aun así estar siempre presentes.

A mis amigos y hermanos por estos cinco años que me han regalado. Por todo lo que compartimos juntos

A mis tutores por su esfuerzo y labor, por forjarme y convertirme en el hombre que soy. Por guiarme y darme todo su apoyo.

A mis profesores por formarme como docente y como persona.

Declaración de autoría

Declaro por este medio que yo, David Álvarez Santes, con carnet de identidad 94072729709 soy el autor principal del trabajo titulado **"Componente para la categorización semántica de documentos en sistemas de recuperación de información"**, y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

Para que así conste firmamos el presente documento a los __ días del mes de _____ del año 2018.

David Álvarez Santes

_____

Firma del Autor

Delly Lien González Hernández

_____

Firma del Tutor

Martha Luisa Gala Rodríguez

_____

Firma del Tutor

Hubert Viltres Sala

_____

Firma del Tutor

Alexander García O´relly

_____

Firma del Tutor

Resumen

En la presente investigación se propone el desarrollo del componente de categorización semántica de documentos indexados, para contribuir en el proceso de recuperación de información en la red cubana. El estudio del arte obtuvo como resultado el funcionamiento, métodos y los algoritmos más utilizados en el proceso de categorización, permitiendo determinar la utilización de las ontologías para mejorar el procesamiento de la información. El componente procesa la información mediante la ontología y categoriza el documento con la categoría más relevante. Para la implementación de la propuesta de solución, guiada por la metodología AUP en su versión adaptada por la Universidad de las Ciencias Informáticas, se seleccionaron las herramientas: Apache Solr como mecanismo de indexación, Spring Boot para las consultas a la ontología y Freeling con el objetivo de procesar la información obtenida. Se determinó utilizar Java como lenguaje de programación y para el entorno de desarrollo integrado se selecciona Netbeans. Los artefactos generados fueron modelados mediante la herramienta Visual Paradigm. La aplicación de las pruebas unitarias, de integración y las funcionales permitieron comprobar el correcto funcionamiento del *software* implementado, para optimizar el proceso de recuperación de la información. Los expertos coincidieron para un 92% en la valoración de muy adecuado, identificándose que el componente constituye una solución funcional, actual y de calidad, evidenciándose la mejora del procesamiento de la información.

Palabras claves: algoritmos, categorización, documentos, ontología.

Índice de Contenido

Índice de Tablas

Índice de Figuras

Introducción

En Internet según Internet Live Stats (2017) existen más de 1.3 billones de sitios web que contienen información en diferentes formatos (pdf, docx, html). Estos datos son consultados por casi más de 4 billones de usuarios, que al interactuar generan nuevos conocimientos, mediante la creación de nueva documentación y comparten la ya existente según confirma Internet Live Stats (2017). Para acceder a la información los usuarios utilizan Sistemas de Recuperación de Información (SRI) que emplean diferentes técnicas para localizar, seleccionar y mostrar resultados de sencilla compresión.

Los SRI utilizan diferentes métodos para procesar las consultas realizadas por los usuarios, el contenido disponible en la web, determinan la relevancia y ordenan los resultados de búsqueda (Viltres et al., 2018). Entre los métodos para mejorar la recuperación de la información se encuentra la agrupación o categorización de los documentos, que permite optimizar los índices de búsqueda. Al analizar datos de la compañía Cumulus Media en el año 2017 se realizaron más de 3.5 millones de búsquedas en Google en un minuto; planteándose un reto para los SRI que necesitan recuperar la información más relevante y satisfacer las necesidades de los usuarios.

La información en Internet crece exponencialmente, en el 2017, en tan solo un minuto, es subido a Instagram un total 46 200 fotos, en Snapchat, 1.8 millones de snaps, son enviados más de 15 000 archivos GIF[1] a través de Facebook Messenger, en Spotify son reproducidos 40 000 horas, se crean alrededor de 156 millones de correos electrónicos y son enviados 16 millones de SMS; además, es reproducido 4.1 millones de videos en YouTube (Cumulus Media, 2017). La variedad de formatos, estructuras y la cantidad de contenido disponible en la web limita la capacidad de los usuarios de obtener información relevante y plantea un reto en el proceso de Recuperación de la Información (RI).

Las investigaciones de Fernández et al. (2011) demuestran que para mejorar el proceso de RI se emplean tecnologías de la Web Semántica (WS), que permiten comprender y personalizar los resultados de búsqueda de los usuarios. En la WS, según Berners-Lee, Hendler y Lassila (2001), la información tiene un significado bien definido, facilita a las computadoras trabajar en una mayor cooperación con los seres

[1] Formato de Intercambio de Gráficos

humanos, entiende la necesidad del usuario y analiza la información disponible a través de algoritmos que simulan comprensión o entendimiento.

Entre las tecnologías de la WS se desarrollan los tesauros[2], SKOS[3] y las ontologías que persiguen la obtención coherente de la estructura informativa según investigaciones de Pastor Sánchez y Martínez Méndez (2009). En el caso de las ontologías permiten un mayor procesamiento de la información según Janik y Kochut (2008), su utilización logra disminuir la ambigüedad de los términos, la extracción del conocimiento y el procesamiento de la información según las necesidades del usuario.

Para mejorar el proceso de RI se emplea la categorización de documentos, en el que es utilizado criterios de agrupamiento que determinan la similitud por temas, conceptos o características de documentos (Pavone, 2015). Existen diferentes herramientas, técnicas y métodos para realizar el proceso de categorizar según investigaciones realizadas por Chinniyan, Gangadharan y Sabanaikam (2017), que propone utilizar la clasificación bayesiana, árboles de decisión, AIRS (sistema de clasificador inmunológico de recursos limitados), k-vecino más cercano y máquinas de vector de soporte.

En Cuba, como parte del proceso de informatización de la sociedad, la Universidad de las Ciencias Informáticas (UCI) contribuye al desarrollo del *software*. En su infraestructura productiva existe el Centro de Ideoinformática (CIDI), que entre sus objetivos de trabajo se traza el desarrollo de la plataforma c·u·b·a. (Contenidos Unificados para la Búsqueda Avanzada). Esta plataforma tiene como propósito recuperar la información alojada en los 6756 dominios (.cu) mediante una interfaz más dinámica y específica a las necesidades de la web cubana (Cubanic, 2017).

La recomendación de información, creación de perfiles y la optimización del cálculo de la relevancia de la información son procesos utilizados por la plataforma c·u·b·a. que constituyen elementos fundamentales en la optimización del proceso de RI. En la implementación de estas funcionalidades se utiliza un campo categoría que permite la obtención de información previamente categorizada.

[2] Listado de palabras o términos controlados, empleados para representar conceptos.
[3] Simple Knowledge Organization System (Sistema simple de organización del conocimiento), proporciona un modelo para representar la estructura básica y el contenido de esquemas conceptuales de un vocabulario controlado.

La plataforma c·u·b·a. dispone de un componente para la categorización automática de documentos implementada por Martínez (2016), que utiliza el algoritmo probabilístico Naive Bayes en su variante Multinomial. Este algoritmo se basa en la Frecuencia de Aparición del Término (FAT), método que determina la categoría predominante según FAT dentro del documento y su representación en las categorías definidas para c·u·b·a. La implementación del algoritmo propuesto por Martínez (2016) presenta como principal inconveniente que se clasifica el documento de acuerdo a la frecuencia de aparición y no del contexto del término dentro del documento. Además, un documento puede estar representado por varias categorías lo que produce la descontextualización e inexactitud respecto a la categoría obtenida. La técnica de frecuencia de aparición del término recupera grandes cantidades de información, con baja precisión y exactitud. Al no analizar el contexto, un término puede representar más de un concepto y no se desambigua[4], por lo que vuelve a ser recuperado y se obtienen resultados poco relevantes.

Tomando en consideración la problemática anteriormente descrita, se plantea como **problema de investigación**: ¿Cómo mejorar el procesamiento de los documentos en un sistema de recuperación de información?

Para llevar a cabo la investigación se toma como **objeto de estudio**: los procesos de categorización de documentos; y la categorización semántica de documentos en un sistema de recuperación de información como **campo de acción**.

Para dar solución al problema antes descrito, se enmarca el siguiente **objetivo general**:

- Desarrollar un componente de categorización semántica de documentos, mediante tecnologías semánticas (TS), para mejorar el procesamiento de la información.

En virtud de solucionar la problemática existente, el objetivo general ha sido desglosado en los siguientes o**bjetivos específicos**:

1) Caracterizar los fundamentos teóricos relacionados con la categorización semántica de documentos.
2) Definir las tecnologías, herramientas y metodología de desarrollo para la implementación del componente de categorización semántica.

[4] Según la Real Academia Española: Efectuar las operaciones necesarias para que una palabra, frase o texto pierdan su ambigüedad (palabra que puede entenderse de varios modos).

3) Diseñar las funcionalidades del componente de categorización semántica.
4) Implementar el componente de categorización semántica.
5) Validar el componente de categorización semántica.

Luego de haber definido los objetivos primordiales se define la siguiente **hipótesis científica:** El desarrollo de un componente de categorización de documentos, basado en tecnologías de la web semántica, mejorará el procesamiento de los resultados en las búsquedas realizadas por los usuarios.

Se utilizaron **métodos** de trabajo científico con el objetivo de establecer una vía a seguir por el autor para describir el conocimiento científico sobre la base de la teoría, y la práctica para dar cumplimiento al objetivo de la investigación:

Métodos Teóricos:

Histórico-Lógico: Utilizado para estudiar el desarrollo evolutivo de los componentes de categorización semántica de documentos, así como de los algoritmos más utilizados para mejorar la comprensión de su funcionamiento, surgimiento y desarrollo de los sistemas de recuperación de la información, en especial los buscadores web.

Analítico-Sintético: Empleado para la descomposición del objeto de estudio, permitiendo combinar los elementos en una propuesta de solución.

Modelado: Utilizado para realizar una reproducción simplificada de la realidad, mostrando las relaciones internas y características del sistema a través de diagramas.

Métodos Empíricos:

Entrevista: A través de un conjunto de preguntas realizadas a los principales especialistas del proyecto, permitiendo conocer las características y problemas que surgen por falta de un proceso de categorización semántico de documentos.

El presente trabajo de diploma consta de la siguiente estructura: Introducción, Tres Capítulos, Conclusiones, Recomendaciones, Bibliografía y Anexos.

Capítulo 1: "Categorización semántica de documentos en Sistemas de Recuperación de Información"

Contiene los fundamentos teóricos que giran sobre los componentes de categorización. Además, se hace un estudio de sistemas homólogos, obteniéndose: objetivos de su uso, funcionamiento y tecnologías que utilizan.

Capítulo 2: "Análisis y diseño del componente de categorización semántica de documentos"

En este capítulo se explica cómo se desarrolla el flujo actual de los procesos, y se describe la propuesta de solución para resolver el problema planteado. Por otra parte, se especifican los requisitos funcionales y no funcionales, y los elementos fundamentales del diseño y arquitectura.

Capítulo 3: "Implementación y validación del componente de categorización semántica de documentos"

En este epígrafe se incluye la programación realizada a partir de los requisitos, así como las pruebas realizadas para su validación. Además, se establecen los estándares de codificación que serán utilizados para el desarrollo del componente.

Capítulo 1: "Categorización semántica de documentos en Sistemas de Recuperación de Información"

Con el objetivo de facilitar la comprensión de la investigación, en el presente capítulo se analizan los principales conceptos asociados al objeto de estudio y al campo de acción. Se realiza un estudio sobre los componentes de categorización, sus características principales, herramientas y tecnologías más utilizadas, para concretar una propuesta de solución.

1.1 Proceso de Recuperación de Información

El desarrollo alcanzado en la actualidad por las tecnologías de la información y las comunicaciones (TIC), y el rápido crecimiento de Internet en los últimos años, ha logrado aumentar de forma exponencial el volumen de información disponible. Para obtener acceso a este cúmulo se utilizan los SRI, estos sistemas basan su implementación en técnicas de la RI, "disciplina que estudia la representación, la organización y el acceso eficiente a la información que se encuentra registrada en documentos" según Abadal et al. (2005), es también "la localización, procesamiento y presentación a un usuario de información relevante a una necesidad de información expresada como una pregunta" (Korfhage, 2008).

Los sitios web publican contenido en diferentes formatos (pdf, docx, excel, otros), al existir diversidad de información se dificulta el proceso de la RI. Esta información puede encontrarse de forma estructura o no estructurada, dificultándose el proceso de extracción de la información relevante. Esto trae como resultado que los SRI necesitan estandarizar los documentos recolectados para un posterior procesamiento.

El proceso de RI puede modelarse desde distintos enfoques, como son el estadístico, algebra de Boole, algebra de vectores, lógica difusa, procesamiento de lenguaje natural y otros al analizar las investigaciones de Bordignon et .al (2010), dentro de estos métodos se encuentran tres modelos clásicos: booleano, probabilístico y vectorial según Baeza-Yates y Ribeiro-Neto (2011) y Martínez Comeche (2006), utilizados por los SRI.

Para la extracción del conocimiento se utilizan diferentes técnicas, entre ellas la Web Semántica, en consonancia con los razonamientos de Berners-Lee, Hendler y Lassila (2001), Matthews (2005) y Ekaputra, Sabou, Serral, Kiesling y Biffl (2017), permite que la información en la web este bien definida y dotada de un significado de forma que logre ser interpretada tanto por agentes humanos como agentes

computarizados, lográndose una mayor interpretación, automatización, integración, representación y reutilización.

1.2 Función de similitud

La similitud semántica en el área de procesamiento de lenguajes naturales, es la medida de relación existente entre dos palabras, este concepto es fundamentado en la idea que se tiene de la lingüística sobre la coexistencia de palabras y del discurso coherente. Dos palabras o términos por el hecho de tener su existencia en un mismo documento poseen un contexto similar, por lo que se puede deducir su distancia semántica (Rada et al, 1989).

Estas funciones de similitud consideran cada cadena de caracteres como una secuencia ininterrumpida de caracteres. Al examinar los resultados de las investigaciones de Amón y Jimenez (2010), Morato et al. (2014) y Valero (2017) fueron identificadas las siguientes funciones de similitud:

Distancia de edición

La distancia de edición entre dos cadenas de texto *A* y *B* se basa en el conjunto mínimo de operaciones de edición necesarias para trasformar *A* en *B* (o viceversa). Las operaciones de edición permitidas son eliminación, inserción y sustitución de un carácter.

En el modelo original, cada operación de edición tiene costo unitario, siendo referido como distancia de Levenshtein (Levenshtein, 1966). Needleman y Wunsch (1970) lo modificaron para permitir operaciones de edición con distinto costo, permitiendo modelar errores ortográficos y tipográficos comunes. Por ejemplo, en el idioma español es frecuente encontrar la letra “n” en lugar de la “m” (Ramírez Bustamante y López, 2006), entonces, tiene sentido asignar un costo de sustitución menor a este par de caracteres que a otros dos sin relación alguna.

Función Coseno

La similitud entre dos vectores de términos se puede calcular mediante el ángulo que forman, en concreto, con la utilización del coseno del ángulo (considerando que cuanto más próximos están dos vectores mayores es la similitud entre ellos). Cuando el ángulo entre los vectores es menor, la similitud es mayor y en consecuencia el coseno del ángulo es mayor (Cacheda Seijo et al., 2011). La ecuación (1) representa la fórmula del coseno:

$$sim(d_i, d_j) = cos(\alpha) = \frac{\sum_{k=1}^{m} d_{ik} \cdot d_{jk}}{\sqrt{\sum_{k=1}^{m} d_{ik}^2} \sqrt{\sum_{k=1}^{m} d_{jk}^2}} \quad \textbf{(1)}$$

En la fórmula 1 d_i y d_j son los documentos y d_{ik} representa el peso del rasgo *k* en el documento d_i.

Distancia de brecha afín

La distancia de edición y otras funciones de similitud tienden a fallar al identificar cadenas equivalentes que han sido demasiado truncadas, ya sea mediante el uso de abreviaturas o la omisión de *tokens* ("Jorge Eduardo Rodríguez López" vs "Jorge E Rodríguez"). La distancia de brecha afín ofrece una solución al penalizar la inserción/eliminación de *k* caracteres consecutivos (brecha) con bajo costo, mediante una función afín p(*k*) = *g* + *h* * (*k* - 1), donde *g* es el costo de iniciar una brecha, *h* el costo de extenderla un carácter, y *h* << *g* (Gotoh, 1982). Bilenko y Mooney (2003) describen un modelo para entrenar automáticamente esta función de similitud a partir de un conjunto de datos. La distancia de brecha afín no normalizada es calculada con un orden computacional 0 (*nm*) mediante el algoritmo de programación dinámica propuesto por Gotoh (1982).

Similitud Smith-Waterman

La similitud Smith-Waterman (SW) entre dos cadenas *A* y *B* es la máxima similitud entre una pareja denotadas (*A* ', *B* '), sobre todas las posibles, tal que *A*' es subcadena de *A* y *B*' es subcadena de *B*, este problema es conocido como alineamiento local. El modelo original de Smith y Waterman define las mismas operaciones de la distancia de edición, y además permite omitir cualquier número de caracteres al principio o final de ambas cadenas (Mott, 2005). El análisis del planteamiento anterior demuestra que la similitud SW es adecuado para identificar cadenas equivalentes con prefijos/sufijos que, al no tener valor semántico, pueden ser descartados.

Es posible normalizar la similitud de Smith-Waterman con base en la longitud de la cadena de mayor longitud, la de menor longitud o la longitud media de ambas cadenas (Christen, 2006), que corresponden a los coeficientes de Jaccar, Overlap y Dice respectivamente. La similitud Smith-Waterman puede ser calculada en 0 (*nm*) mediante el algoritmo de Smith-Waterman (1981). Baeza-Yates y Gonnet (1992)

presentan un algoritmo que toma 0 (*n*) para verificar si la similitud Smith-Waterman entre dos cadenas es menor que cierta constante *k*.

Similitud de Jaro

Jaro (1976) desarrolló una función de similitud que define la trasposición de dos caracteres como la única operación de edición permitida. Los caracteres no necesitan ser adyacentes, sino que pueden estar alejados cierta distancia *d* que depende de la longitud de ambas cadenas.

Winkler (1990) propone una variante que asigna puntajes de similitud mayores a cadenas que comparten algún prefijo, basándose en un estudio realizado por Pollock y Zamora (1984). En Cohen *et al.* (2003) se propone un modelo basado en distribuciones Gaussianas. William (2006) compara la eficacia de la similitud de Jaro y algunas de sus variantes. La similitud de Jaro puede ser calculada con un orden de complejidad de 0 (*n*).

Selección de la función de similitud

El análisis de las funciones de similitud identificadas por la bibliografía permite la selección de la Función Coseno por presentar facilidades de mapeo a dimensiones diferentes en el modelo espacio-vectorial, permitiendo la similitud entre palabras relacionadas semánticamente (Sidorov et al., 2014). La complejidad de esta función es cuadrática, permitiéndole ser completamente aplicable a problemas del mundo real. La principal característica de esta función consiste en realizar una normalización de los vectores de forma más suave, no asignando demasiada importancia a los documentos cortos, según investigaciones de Torres y Arco (2016). El estudio de las funciones también demostró que para el desarrollo de la investigación se puede poner en práctica cualquiera de las anteriormente descritas. Luego de aplicar la función se utiliza un algoritmo de categorización de documentos para obtener la relevancia de las categorías.

1.3 Categorización de documentos

La categorización de documentos según varios autores Johnson y Zhang (2014), Selvi et al. (2017) y Gadri y Moussaoqui (2017) consiste en organizar los documentos o separarlos en grupos de modo que posteriormente permita mejorar su recuperación y análisis. En investigaciones realizadas por Raju, Subrahmanian y Sivakumar (2017), Fatima y Srinivasu (2017) y Wadhawan y Mittal (2017) plantean como principales técnicas: clasificador bayesiano, árboles de decisión, algoritmo de Rocchio, k-vecino más cercano y máquinas de vector de soporte.

Bayesian Classifier

Es uno de los algoritmos conocidos más efectivos para la clasificación de documentos, se basa en la teoría de clasificación estadística bayesiana. El objetivo es clasificar una muestra X en una de las clases conocidas C1, C2, ... Cn, mediante la utilización de un modelo de probabilidad definido según la teoría de Bayes. Cada categoría se caracteriza por una probabilidad previa de observar la categoría Ci. Además, suponemos que en una determinada muestra que pertenece a una categoría Ci con la función de densidad de probabilidad condicional p (x | Ci) ∈ [0,1]. Luego, utilizando las definiciones anteriores y basadas en la fórmula de Bayes, se define la probabilidad posterior:

$$p(C_i|x) = \frac{p(x|C_i)p(C_i)}{p(x)} \qquad \textbf{(2)}$$

Un patrón de entrada se clasifica en una categoría cuando la parte posterior es la de más alta probabilidad. El clasificador bayesiano más simple es el clasificador Naive Bayes. El *Naive Bayes Classifier* se basa en la suposición simplificadora de que los valores de los atributos son condicionalmente independientes dado el valor objetivo. La suposición es que, dado el valor objetivo de la instancia, la probabilidad de observar la conjunción α1, α2 ... αn es solo el producto de las probabilidades. El valor objetivo generado por *Naive Bayes Classifier* (uNB) es:

$$u_{NB} = \arg\max_{u_j \in V} p(u_j) \prod_i p(\alpha_i | u_j) \qquad \textbf{(3)}$$

En teoría, los clasificadores Bayesianos tienen el porcentaje de error más bajo en comparación con el resto de los clasificadores. En la práctica esto no siempre es cierto, a falta de datos disponibles para el cálculo preciso de las probabilidades condicionales. Sin embargo, las investigaciones han demostrado que *Naive Bayes Classifier* es competitivo con otros clasificadores bien conocidos, como *Decision Trees* (Árbol de decisión) y *Neural Networks*.

Decision Trees

Se construye sobre la base de un conjunto de entrenamiento de datos pre-categorizados. Cada nodo interno del árbol especifica una prueba de un atributo de la instancia y cada rama que desciende de ese nodo corresponde a uno de los valores posibles para este atributo. El procedimiento para clasificar una nueva instancia mediante la utilización de un árbol de decisión es la siguiente: se comienza en la raíz del árbol con el análisis del atributo especificado por este nodo, los nodos internos sucesivos son visitados hasta que se alcanza una hoja. En cada nodo interno, la prueba del nodo se aplica a la instancia.

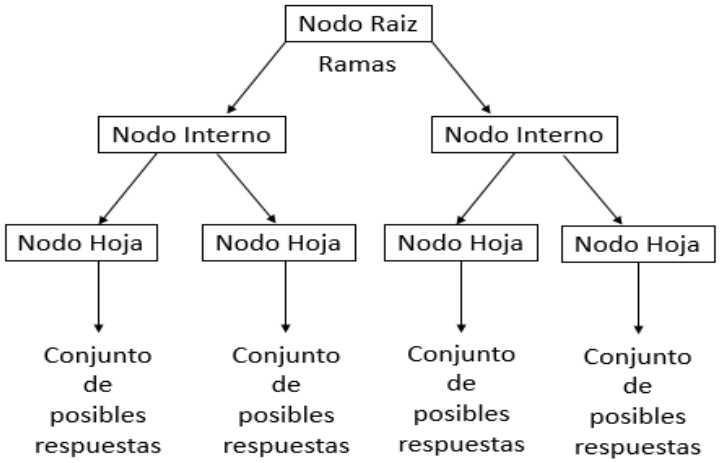


Figura 1: Árbol de decisión.

La mayoría de los algoritmos desarrollados para construir árboles de decisión tienen dos fases distintas, una fase de construcción y una fase de poda. En la fase de construcción, el conjunto de datos de entrenamiento se divide recursivamente hasta que todas las instancias en una partición tengan la misma clase. El resultado de este procedimiento es un árbol que clasifica cada *ítem* de datos del conjunto de entrenamiento. Estos algoritmos realizan una fase de poda después de una fase de construcción, en la cual los nodos son podados para evitar el sobreajuste y obtener un árbol con mayor precisión.

Máquinas de vector de soporte (SVM del inglés *Support Vector Machine*)

Son un conjunto de algoritmos de aprendizaje supervisado, propiamente relacionados con la resolución de problemas de clasificación no lineales y desarrollados a partir de la teoría del riesgo estructural. Estos métodos funcionan dado un conjunto de puntos, subconjunto de un conjunto mayor (espacio), en el que cada uno de ellos pertenece a una de dos posibles categorías, se construye un algoritmo SVM mediante un modelo capaz de predecir si un punto nuevo (cuya categoría es desconocida) pertenece a una categoría o a la otra. La SVM busca un hiperplano que separe de forma óptima a los puntos de una clase de la de otra,

que eventualmente han podido ser previamente proyectados a un espacio de dimensionalidad superior. De esta forma, los puntos del vector que son etiquetados con una categoría estarán a un lado del hiperplano y los casos que se encuentren en la otra categoría estarán al otro lado. SVM demuestra ser eficaz para la categorización de texto debido a su propiedad de aprendizaje independientemente de la dimensionalidad de las características de su espacio. Las SVM no necesitan ajuste de parámetros, ya que tienen una forma automática de encuentra buenos parámetros.

k-Nearest Neighbor (kNN) (Vecino más cercano)

Los métodos de aprendizaje anteriores se llaman métodos ansiosos porque dan una descripción explícita de la función objetivo en todo el conjunto de entrenamiento. Por otra parte existen los métodos basados en instancia que se llaman métodos perezosos, porque el modelo de clasificación no está construido a priori. En su proceso de aprendizaje se almacenan todas las instancias de entrenamiento y en el procedimiento de clasificación que asignan el objetivo funcionar a una nueva instancia. El método kNN estima el valor de la función de densidad de probabilidad o directamente la probabilidad a posteriori de que un elemento *x* pertenezca a la clase C_j a partir de la información proporcionada por el conjunto de prototipos. Los ejemplos de entrenamiento son vectores en un espacio característico multidimensional, cada ejemplo está descrito en términos de *p* atributos considerando *q* clases para la clasificación.

La fase de entrenamiento del algoritmo consiste en almacenar los vectores característicos y las etiquetas de las clases de los ejemplos de entrenamiento. En la fase de clasificación, la evaluación del ejemplo (del que no se conoce su clase) es representada por un vector en el espacio característico. Se calcula la distancia entre los vectores almacenados y el nuevo vector, y se seleccionan *los k* ejemplos más cercanos. El nuevo ejemplo es clasificado con la clase que más se repite en los vectores seleccionados.

Este método supone que los vecinos más cercanos nos dan la mejor clasificación mediante el análisis de todos los atributos; el problema de la suposición es que es posible que se tengan muchos atributos sin importancia que dominen sobre la clasificación: dos atributos relevantes perderían peso entre otros veinte irrelevantes.

Rocchio

El llamado algoritmo de Rocchio es bien conocido y aplicado en la realimentación de consultas. En este ámbito, la idea es simple: formulada y ejecutada una primera consulta, el usuario examina los documentos devueltos y determina cuáles le resultan relevantes y cuáles no. Con estos datos, el sistema genera automáticamente una nueva consulta, basándose en los documentos que el usuario señaló como relevantes o no relevantes. En este contexto, el algoritmo de Rocchio proporciona un sistema para construir el vector de la nueva consulta, recalculando los pesos de los términos de ésta y aplicando un coeficiente al peso de la consulta inicial, otro a los documentos relevantes y otro distinto a los que no son relevantes. En el ámbito de la categorización, el algoritmo de Rocchio proporciona un sistema para construir los patrones de cada una de las clases o categorías de documentos. Así, partiendo de una colección de entrenamiento, categorizada manualmente de antemano, y aplicando el modelo vectorial, se construyen vectores patrón para cada una de las clases, se consideran como ejemplos positivos los documentos de entrenamiento de esa categoría, y como ejemplos negativos los de las restantes categorías.

Selección del algoritmo

La categorización de documentos puede obtenerse mediante diferentes técnicas. Fue realizado un estudio de los algoritmos más frecuentes en la categorización de documentos; al margen de mayor o menor complejidad de cada uno de ellos, se tratan de técnicas suficientemente maduras, cuyos resultados son equiparables a los conseguidos por los categorizadores inteligentes, los humanos.

El análisis de las principales técnicas para la categorización de documentos, permite la selección del algoritmo ***Bayesian Classifier***. Este método es uno de los más simples, rápido y en comparación a otros modelos produce resultados de mayor precisión y es uno de los más utilizados en el proceso de categorización (Tarragó, 2014). Este método es robusto al posible ruido en los ejemplos de entrenamiento y a la posibilidad de existir en estos, datos incompletos o posiblemente erróneos (Lantz, 2013). Se basa en la aplicación de la Regla de Bayes para predecir la probabilidad condicional de la pertenencia de un documento a una clase a partir de la probabilidad de los documentos dada la clase y la probabilidad a priori de la clase en el conjunto de entrenamiento (McCallum y Nigam, 1998).

1.3 Sistemas para la categorización de documentos

La categorización de documentos es una tarea compleja y subjetiva incluso para diferentes personas, que podrían asignar el documento correspondiente a grupos diferentes, mediante diversos criterios válidos. A continuación, se presenta el estudio realizado a los sistemas del ámbito internacional y nacional en el área de la categorización de documentos, para la comprensión del funcionamiento de estos sistemas en específico.

Aspire

Aspire Content Processing es un marco innovador y potente, diseñado específicamente para datos no estructurados. Forma parte de la colección de recursos de tecnología independiente de *Search Technologies* que ayudan a las organizaciones a optimizar sus arquitecturas de búsqueda y grandes cúmulos de datos. Esta herramienta apoya los enfoques primarios de categorización: basado en vocabulario, mediante la utilización de una taxonomía (opcionalmente con sinónimos y reglas de retención) u otros recursos semánticos y la categorización basada en ejemplos, utilizando documentos de semillas. En algunos casos, una combinación de estos métodos proporciona los mejores resultados. La versión corporativa de Aspire, proporciona una solución completa, flexible y total compatibilidad con las necesidades de categorización de Solr (Search Technologies, 2017).

Proyecto Scorpion

Desarrollado por OCLC (*Online Computer Library Center*, Centro de Biblioteca de Computadoras en Línea en español), cuyo principal objetivo es demostrar la efectividad de la utilización de esquemas de clasificación y encabezamientos de materia para la indización automatizada. Actualmente su centro de atención primordial es la categorización de recursos electrónicos mediante DDC (*Dewey Decimal Classification*, Clasificación Decimal Dewey en español, sistema de clasificación de bibliotecas creado en 1876). La base de datos inicial utilizada en este proyecto está formada por los registros empleados en la Edición Electrónica de la DDC para el sistema operativo *Windows*. En esta base de datos se analiza la naturaleza jerárquica de las relaciones conceptuales de la DDC de forma que la información sobre el significado de cada notación concreta, se asocia la información sobre la clase a la que pertenece, consiguiéndose contextualizar cualquier término. A partir de esta base de datos se utiliza la ponderación y el cálculo de la similitud entre los términos extraídos de los documentos para representar su contenido y los términos utilizados para representar los conceptos de los registros. El proceso inicia con una definición de los registros o *cluster*, los documentos son tomados como una búsqueda y se intenta localizar una ubicación

en alguno de ellos. Para mejorar los resultados se han desarrollado diferentes filtros que permitan eliminar los registros demasiado genéricos o demasiado específicos dentro de una clase determinada (OCLC, 2017).

Expert System

Permite a las empresas mantenerse competitivas en un mundo que requiere un procesamiento cada vez más rápido de grandes cantidades de información diversa y en constante aumento. Esto es posible mediante la utilización de la tecnología inteligente y aplicaciones que proporcionan una comprensión exacta, automática e inmediata del texto**.**

Cogito Discover, es un *software* de la compañía que realiza la minería de textos, con enriquecimiento del contenido semántico e identificación de entidades potente y escalable que reconoce e identifica elementos relevantes de la información ocultos en el texto y enriquece los metadatos de los documentos. Incluye motores sintácticos, estadísticos y taxonómicos, además de realizar la extracción de información, soporta múltiples idiomas, lo que permite implementar aplicaciones de extracción de alto rendimiento en un amplio rango de situaciones y geografías. El sistema ejecuta análisis amplios y detallados, con clasificación incluida, en conjuntos de datos estructurados y no estructurados, garantizándose que las fuentes de conocimiento más exhaustivas están identificadas, categorizadas y disponibles para una búsqueda y un análisis más efectivo. Identifica y asocia correctamente el contenido con las clases y nodos relevantes, generándose metadatos listos para su uso. Fue implementado con la facilidad de realizar tareas de categorización, clasificación semántica automática y etiquetado de documentos. Además, se puede integrar como un sistema de clasificación completamente automatizado (Expert System, 2017).

Plataforma c·u·b·a

Es un sistema de recuperación de información encargado de encontrar los contenidos alojados en la red nacional. La personalización de este buscador a su entidad le permite realizar búsquedas sobre los contenidos publicados en la web mediante una interfaz amigable e intuitiva. La plataforma dispone de un sistema de categorización de documentos Martínez (2016), encargado de clasificar mediante la implementación del algoritmo probabilístico Naive Bayes en su variante Multinomial. Este modelo utiliza el método de FAT, que asignar una categoría al documento según la frecuencia con que aparezca un término. Presenta como principal inconveniente que un documento es categorizado mediante la frecuencia en que pueda aparecer una palabra en el texto y no analizar el contexto del término dentro del documento.

Valoración

Tabla 1: Resumen de los sistemas de categorización de documentos.

Sistema	Tipo de licencia	Utilización de ontologías	Lenguaje de programación	Tipos de datos
Aspire	Privativa	No	Java	Estructurado
Proyecto Scorpion	Privativa	No	Java	Estructurado
Expert System	Privativa	General	Java	No estructurado
Plataforma c·u·b·a.	Libre	No	Java	No estructurado

El análisis de los sistemas de categorización permitió analizar que en Internet existen diversos componentes de categorización, pero no responden a las necesidades de la aplicación que se desea implementará, se detectó como deficiencia que el tipo de licencia es privativa en el caso del ámbito internacional. En la esfera nacional existe un componente para la categorización automática de documentos, pero no presenta elementos que formen parte de la propuesta de solución. Además, el estudio evidencia la utilización de Java como lenguaje de programación, el tipo de datos no estructurado que utilizará el componente, el desarrollo mediante herramientas de *software* libre y el uso de ontologías, como tecnología de la WS para mejorar el procesamiento de la información.

1.5 Web Semántica

El término “Web Semántica” fue presentado por Tim Berners-Lee, considerado el padre de la Web actual, James Hendler y Ora Lassila en su conocido artículo “The Semantic Web” (Berners-Lee et al., 2001). En este artículo los autores remarcan la carencia de significado de la web y proponen el cambio de representación de contenido para que incluya “una semántica bien definida”, que sea entendible para los seres humanos y para las máquinas, de forma que sean capaces de realizar determinadas tareas de forma autónoma.

La Web Semántica surge como respuesta a las limitaciones que presenta la web actual en lo referente al procesado automático de la información, la interoperabilidad con los sistemas de información, e incluso una indexación que permita manejar, de manera más eficiente, la ingente cantidad de información para buscar y encontrar de manera rápida, fácil y precisa la información que se desea.

La mayoría de los contenidos de la web son implementados mediante lenguajes de etiquetado orientados a la presentación de datos. Estos lenguajes ofrecen escasa información de los documentos y de su contenido, y poseen un número finito de etiquetas para describir los contenidos por lo que su nivel de expresividad es bastante limitado. La WS propone la modificación de la forma en que se presentan los contenidos, sino que también incluya información que describa el contenido para facilitar su procesamiento por parte de las máquinas. Una vía de solución ante el problema sería la utilización de ontologías, esta tecnología proporciona un vocabulario que permite describir las relaciones entre diferentes términos de manera flexible y sin ambigüedades, facilitando su interpretación por las máquinas y los humanos (Horrocks et al., 2003).

1.6 Tecnologías, herramientas y metodologías

1.6 .1 Ontologías

Diferentes aplicaciones de la Web Semántica emplean ontologías como esquemas conceptuales para dotar de significado a los términos. La utilización de las ontologías en las Ciencias de la Computación posibilita simular el lenguaje de comunicación de las personas y crea modelos para representar áreas de conocimiento de la vida cotidiana de las personas y posibilitar su interpretación por las computadoras (Maedche y Staab, 2012).

Una ontología es "una especificación explicita y formal de una conceptualización" según la definición ofrecida por Gruber (1993) y extendida por Studer, Benjamins y Fensel (1998). La ontología está formada por una taxonomía relacional de conceptos y por un conjunto de axiomas o reglas de inferencia mediante los cuales se podrá inferir nuevo conocimiento

La palabra ontología se ha convertido en relevante para la comunidad de Ingeniería del Conocimiento, el término se refiere a un modelo de conocimiento restringido a un dominio específico según Albacete (2016). Una ontología es utilizada para representar conocimiento, permite la realización de búsquedas de información con carácter semántico, con la finalidad de obtener resultados mucho más óptimos (Melgar, 2011).

Steve, Gangemi y Pisanelli (1997) hacen referencia a tres tipos de ontologías:

- **Ontologías de dominio:** Representa el conocimiento especializado pertinente a un dominio o subdominio, como la medicina, las aplicaciones militares, la cardiología, etc.

- **Ontologías genéricas:** Representan conceptos generales y funcionales del conocimiento como las estructuras parte/todo, la cuantificación, los procesos o los tipos de objetos.
- **Ontologías representacionales:** Especifican las conceptualizaciones que subyacen a los formalismos de representación del conocimiento, también denominadas meta-ontologías.

En investigaciones realizadas por Guarino (1998) se añade a las anteriores ontologías, las desarrolladas con el propósito de representar una aplicación, tarea o actividad específica, un ejemplo de esta última sería la venta de productos o el diagnostico de una enfermedad. En la investigación se emplearán ontologías de tipo tarea y aplicación, encargadas de contener los conceptos asociados a cada categoría correspondiente.

1.6.2 Base tecnológica

Java

Lenguaje de programación concurrente, basado en clases, y orientado a objetos, fue diseñado específicamente para tener pocas dependencias de implementación como fuera posible. Una de las características más importantes es que los programas "ejecutables", creados por el compilador de Java, son independientes de la arquitectura. Es multiplataforma, un *software* de distribución libre, es completo y poderoso, se pueden realizar muchas tareas con él, pues posee librerías y utilidades muy completas que facilitan la programación (Gironés, 2012).

Spring Boot

Es un *framework* de desarrollo de código libre para la plataforma Java. Su aspecto modular lo hace flexible y configurable para cualquier tipo de aplicación. Se compone de múltiples módulos que abarcan gran variedad de servicios que facilitan la implementación como son: *Spring MVC* (basado en HTTP para aplicaciones web y servicios) y el Contenedor de inversión de control (para la inyección de dependencias). *Spring* aplica el concepto de *Convention over Configuration*, paradigma de programación, que minimiza las decisiones que toman los desarrolladores mediante la generación rápida de aplicaciones, simplificándose las tareas (Gutiérrez Faraoni, 2015).

Visual Paradigm for UML

Es una herramienta CASE (*Computer Aided Software Engineering* por sus siglas en inglés, Ingeniería de *software* asistidas por computadoras) multiplataforma, que soporta el ciclo completo del *software*: análisis, diseño, implementación y pruebas. Facilita la construcción de aplicaciones informáticas con un menor coste

que destacan por su alta calidad y contribuye a mejorar la experiencia del usuario mediante el diseño de un gran número de artefactos de ingeniería de *software*. Permite la generación de bases de datos, conversión de diagramas entidad-relación a tablas de base de datos, mapeos de objetos y relaciones, ingeniería directa e inversa, la gestión de requisitos de *software* y la modelación de procesos del negocio (Visual Paradigm, 2017). Será empleada para el diseño metodológico documentado y generando información sobre el componente a desarrollar.

Entorno de Desarrollo Integrado (IDE) Netbeans

Es un entorno de desarrollo integrado y una plataforma de desarrollo. Aunque inicialmente, *NetBeans IDE* sólo podía ser utilizado para desarrollar aplicaciones *Java*, a partir de la versión 6, *NetBeans* soporta varios lenguajes de programación, ya sea a través de una función de apoyo, o mediante la instalación de complementos adicionales, algunos de estos lenguajes son *Java*, C, C ++, PHP, HTML, *JavaScript*, y *Scala* (Heffelfinger, 2015). *NetBeans* proporciona funcionalidades adicionales como la compilación en tiempo real, la comprobación de tipos, refactorización, navegadores de clase y soluciones rápidas para los errores en tiempo de compilación.

Freeling

Conjunto de herramientas de análisis de lenguaje de código abierto, publicado bajo la Licencia Pública General de Affero GNU de la *Free Software Foundation*. Proporciona funcionalidades de análisis de lenguaje (análisis morfológico, detección de entidad nombrada, etiquetado PoS, análisis sintáctico, desambiguación de sentido de palabra, etiquetado de función semántica, etc.) para una variedad de idiomas (inglés, español, portugués, italiano, francés, alemán, ruso, catalán, gallego, croata, esloveno, entre otros). También proporciona un *front-end* de línea de comandos que se puede utilizar para analizar textos y obtener el resultado en el formato deseado (XML, JSON, CoNLL) (Freeling, 2017).

Solr

Apache Solr es un motor de búsquedas de código abierto que proporciona potentes funcionalidades de búsqueda y navegación por facetas, explora la información desde diversas perspectivas. Las funcionalidades son difíciles de implementar sobre una base de datos relacional. Solr tiene la capacidad de manejar complejos criterios de búsqueda, corrige la ortografía de las consultas, permite destacar resultados, configurar la relevancia de términos, entre otras funcionalidades (Estrada, 2012).

1.6.3 Metodologías de desarrollo

En ingeniería de *software* la metodología de desarrollo es un marco de trabajo utilizado para estructurar, planificar y controlar los procesos de desarrollo en los sistemas de información. Las metodologías de desarrollo se clasifican en dos grandes categorías: las tradicionales, diseñadas para proyectos de gran extensión y las metodologías ágiles, utilizadas en proyectos de menor magnitud y dimensión. Para la construcción del componente de categorización semántica de documentos se realizó un estudio sobre tres metodologías de desarrollo ágil: e FALLIS Programming (XP), Open Unified Process (OpenUP) y Agile Unified Process (AUP-UCI) con el objetivo de definir cuál responde a las necesidades del *software* a desarrollar.

Programación extrema (eXtreme Programming)

Es una metodología de desarrollo de la ingeniería de *software*, desatacado dentro de estos procesos. Se diferencia de las tradicionales en la realización de mayor énfasis en la adaptabilidad (Bustamante et al., 2014).

Los pasos fundamentales dentro de las fases son:

- **Desarrollo iterativo o incremental:** Pequeñas mejoras, unas de otras.
- **Pruebas unitarias continuas:** Frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja siempre escribir el código de la prueba antes de la codificación.
- **Programación por parejas:** Se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. Obteniéndose de esta manera un código de calidad, revisado y discutido en lo que es descrito, lográndose no incurrir en una posible pérdida de productividad.
- **Refactorización del código:** Reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad, pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.
- **Simplicidad de código:** Cuando todo funcione se podrá añadir alguna funcionalidad si es necesario. La programación extrema apuesta a que es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere.

Open Unified Process (OpenUP)

Es una metodología que solo incluye el contenido fundamental y necesario, por lo que no provee lineamientos para los elementos que se manejan en un proyecto, pero tiene los componentes básicos que pueden servir de base a procesos específicos. La mayoría de los elementos de OpenUP están declarados para fomentar el intercambio de información entre los equipos de desarrollo y mantener un entendimiento compartido del proyecto, sus objetivos, alcance y avances (Eclipse Process Framework, 2017).

Principios de OpenUP:

- Colaborar para sincronizar el interés y compartir conocimiento. Este principio promueve prácticas que impulsan un ambiente de equipo saludable, facilita la colaboración y desarrollan un conocimiento compartido del proyecto.
- Equilibrar las prioridades para maximizar el beneficio obtenido por los intereses en el proyecto. Este principio promueve prácticas que permiten a los integrantes de los proyectos desarrollar una solución viable que maximice los beneficios obtenidos por los participantes y cumple con los requisitos del proyecto.
- Centrarse en la arquitectura de forma temprana para minimizar el riesgo y organizar el desarrollo.
- Desarrollo evolutivo para obtener retroalimentación y mejoramiento continuo. Este principio promueve prácticas que permiten a los equipos de desarrollo obtener retroalimentación temprana y continua de los participantes del proyecto, permitiendo demostrarles incrementos progresivos en la funcionalidad.

Agile Unified Process (AUP)

El proceso unificado (Unified Process UP) es un marco de desarrollo de *software* iterativo e incremental. Considerado como un proceso altamente formal porque especifica muchas actividades y artefactos involucrados en el desarrollo de un proyecto de *software*. AUP propone que aquellos elementos con alto riesgo obtengan prioridad en el proceso de desarrollo y sean abordados en etapas tempranas del mismo, es realizado mediante listas que identifican los riesgos en etapas iniciales del proyecto. En este sentido se desarrollan prototipos ejecutables durante la base de elaboración del producto, donde se demuestre la validez de la arquitectura para los requisitos clave del producto y que determinan los riesgos técnicos (García et al., 2018).

Al igual que en RUP, en AUP se establecen cuatro fases que transcurren de manera consecutiva y que acaban con hitos claros alcanzados:

- **Concepción:** El objetivo de esta fase es obtener una comprensión común entre cliente-equipo de desarrollo del alcance del nuevo sistema y definir una o varias arquitecturas candidatas a aplicar.
- **Elaboración:** El objetivo es que el equipo de desarrollo profundice en la comprensión de revisar los requisitos del sistema y en validar la arquitectura.
- **Construcción:** Durante esta fase el sistema es desarrollado y probado en el ambiente de desarrollo.
- **Transición:** El sistema se utilizado en los entornos de pre-producción donde es sometido a pruebas de validación y aceptación, finalmente se despliega en los sistemas de producción.

Selección de la metodología de desarrollo a utilizar

La valoración de las características de las metodologías de desarrollo expuestas en el presente epígrafe, llega a la conclusión de emplear AUP en una versión adaptada por la UCI. Esta metodología es recomendada para proyectos no muy extensos y permite ser adaptada a las peculiaridades de cada proyecto, lográndose así que el proceso sea configurable.

Las fases de AUP-UCI son: inicio, ejecución y cierre. Durante el inicio del proyecto se elaboran las actividades relacionadas con la planeación. En la segunda fase se ejecutan las actividades requeridas para desarrollar el *software*, incluyendo el ajuste de los planes del proyecto, considerándose los requisitos y la arquitectura. En la fase de cierre se analizan los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre de proyecto.

AUP-UCI define cuatro escenarios para modelar el sistema en dependiendo de la complejidad y características del proceso de desarrollo. El escenario número cuatro de esta metodología se adapta al desarrollo del componente propuesto: Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio bien definido. El cliente se encontrará siempre acompañando al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos (Universidad de las Ciencias Informáticas, 2015).

Conclusiones del capítulo

Luego de analizar los principales elementos teóricos abordados en el capítulo, se arriban a las siguientes conclusiones:

- El estudio realizado sobre los sistemas homólogos del ámbito internacional brindan resultados concretos sobre el funcionamiento y principales técnicas empleadas, permitiendo determinar las características que constituyen la base para el diseño de las funcionalidades que se definen en la propuesta de solución.
- El estudio de las metodologías y herramientas arribó a definir los componentes para el desarrollo de la solución. Es definido AUP-UCI como metodología conveniente para el desarrollo del presente trabajo de diploma y como herramienta de modelado y diseño de los artefactos a *Visual Paradigm*. Se selecciona Spring Boot para las consultas a la ontología, Apache Solr como servidor de indexación y Netbeans como IDE de desarrollo.

Capítulo 2: "Análisis y diseño del componente de categorización semántica de documentos"

En el capítulo se detalla las características del componente de categorización semántica de documentos según plantea la metodología AUP-UCI en su escenario 4, se presenta la propuesta de solución a desarrollar. Se especifican las funcionalidades del componente con la descripción de los requisitos funcionales, no funcionales y las historias de usuario como artefacto principal generado. Se describe la arquitectura a utilizar, el diagrama de clases con los patrones de diseño utilizados y el diagrama de despliegue del sistema.

2.1 Descripción de la propuesta de solución

El proceso del componente de categorización semántica comienza con la extracción de los documentos del servidor de indexación Apache Solr, esta información es procesada mediante la limpieza de palabras vacías o "***stopword***" (no aportan información al texto) y es transformada a entidades nombradas (términos clave dentro del contexto del documento) mediante la herramienta Freeling.

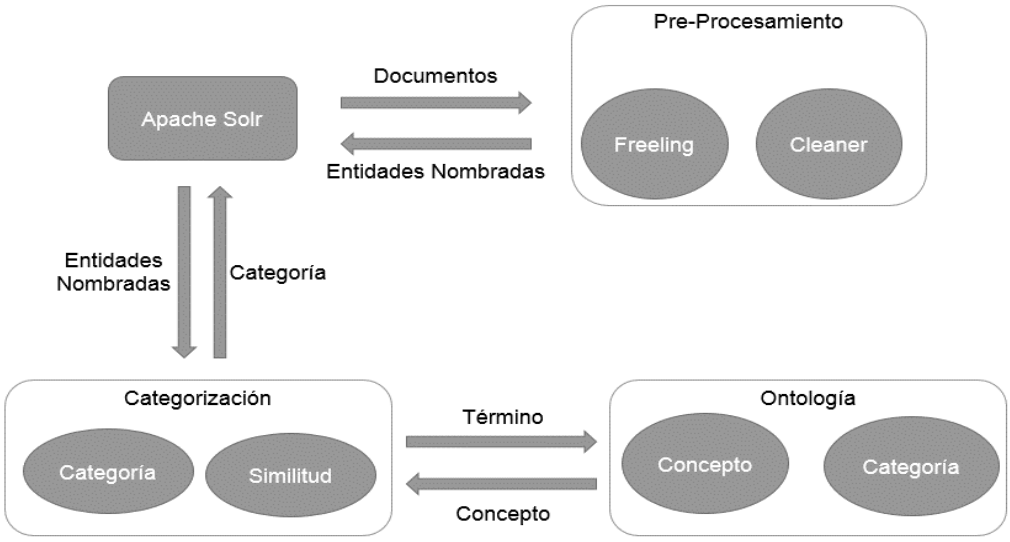


Figura 2: Proceso de categorización semántica de documentos en SRI.

En Apache Solr es creado el metadato "***keywords***", para almacenar las entidades nombradas. Estos términos son asociados y representados por los conceptos de una ontología, mediante la utilización de una función de similitud. El componente prosigue con un algoritmo de categorización que retorna la relevancia de las categorías obtenidas. Se crean dos nuevos campos con el nombre "***semantic_keywords***" y

"***semantic_category***", en los que se guardarán los conceptos de las entidades nombradas y las categorías relevantes al documento respectivamente.

2.2 Modelo conceptual

Un modelo del dominio, o modelo conceptual, es una representación visual de las clases conceptuales u objetos del mundo real en un dominio de interés. Es importante resaltar que un modelo del dominio no representa componentes de *software*. No es tratado como un conjunto de diagramas que describen clases de *software*, u objetos *software* con responsabilidades (Larman, 2004). A continuación, se muestra el diagrama de modelo de dominio del componente de categorización semántica de documentos:

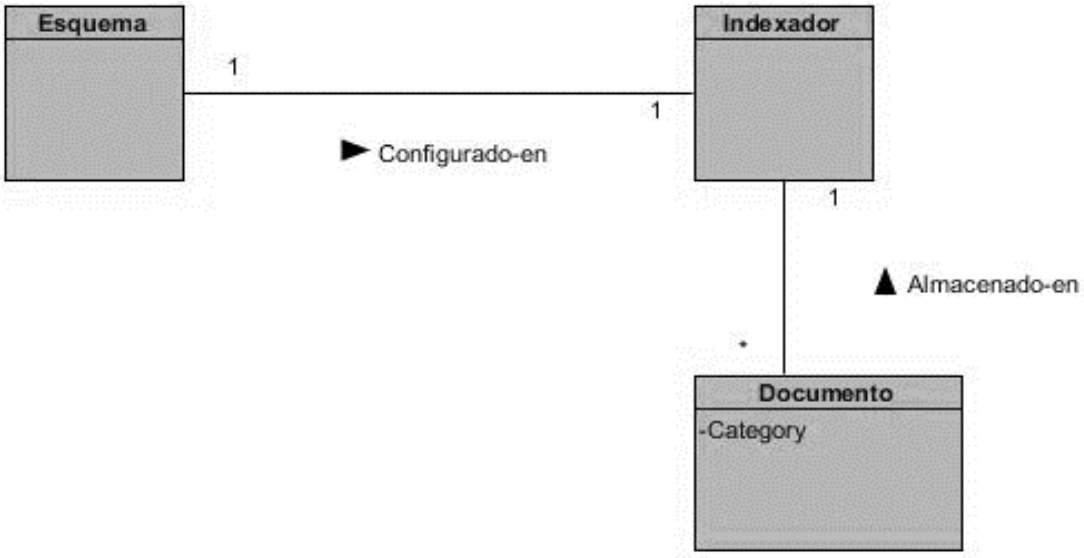


Figura 3: Diagrama de modelo de dominio del componente de categorización de documentos.

Descripción de los objetos presentes en el modelo de estudio:

- ✓ **Esquema:** archivo de configuración que rige al servidor de indexación.
- ✓ **Indexador:** es la herramienta que se encarga de indexar los documentos recuperados de la Web.
- ✓ **Documento:** documento recuperado de la Web.
- ✓ **Categoría:** es el campo que contiene la categoría del documento.

2.3 Especificación de requisitos del Software

Los requisitos del *software* permiten establecer lo que el sistema debe hacer, sus características fundamentales y las restricciones en el funcionamiento del sistema y los procesos de desarrollo del *software*. De manera general, estos requisitos expresan las necesidades objetivas que presentan los usuarios, ante

un sistema que resuelve un problema en particular de un determinado dominio (Sommerville, 2005). Estas cualidades se clasifican en:

- **Requisitos funcionales (RF):** Describen lo que el sistema debe realizar (Sommerville, 2005).
- **Requisitos no funcionales (RNF):** Son aquellos que no se refieren a las funcionalidades específicas que proporciona el sistema, sino a las propiedades emergentes de éste como fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento (Sommerville, 2005).

Requisitos funcionales

La Tabla 2 muestra un listado de los requisitos funcionales y su correspondiente prioridad (alta, media y baja), con el objetivo de esclarecer la comunicación entre usuario y *software*:

Tabla 2: Listado de requisitos funcionales.

No. RF	Nombre	Descripción	Prioridad
1	Obtener documentos de Apache Solr.	Establecer flujo de comunicación con el servidor Apache Solr.	Media
2	Identificar las entidades nombradas.	Pre-procesamiento de los documentos obtenidos de Apache Solr.	Media
3	Almacenar entidades nombradas.	Actualizar el campo "*keywords*" con los resultados obtenidos.	Media
4	Identificar los conceptos en la ontología.	Se identifican y asocia cada concepto presente en la ontología con las entidades nombradas que aparezcan en el documento.	Alta
5	Almacenar los conceptos.	Actualizar el campo "*semantic_keywords*" con los resultados obtenidos mediante una función de similitud.	Media
6	Categorizar documento.	Aplicar el algoritmo de categorización.	Alta
7	Obtener la relevancia de las categorías.	Se realiza una selección de la categoría más relevante.	Alta

8	Almacenar las categorías.	Actualizar el campo "*semantic_category*" con los resultados obtenidos.	Media

Requisitos no funcionales

El componente de categorización semántica de documentos debe cumplir con los siguientes RNF:

Tabla 3: Listado de requisitos no funcionales.

No. RNF	Nombre	Atributo
1	Emplear Java como lenguaje de programación.	*Software*
2	Los servidores donde se instalarán cada uno de los elementos del componente seguirán una distribución GNU/Linux.	*Software*
3	La aplicación se ejecutará en cualquier SO que soporte las librerías de Java 8.	Funcionalidad
4	Utilizar una herramienta que almacene la información indexada.	Funcionalidad
5	Requisitos mínimos en el servidor: 16 Gb de memoria RAM y CPU con 4 *cores* de procesamiento a 3.0 GHz de velocidad.	Hardware
6	Los servidores para la indexación de información deben poseer un disco duro de 2 TB de almacenamiento.	Hardware
7	El componente no debe requerir de información confidencial.	Seguridad
8	El componente debe ser capaz de responder en el menor tiempo posible, en dependencia del tipo de petición y los datos que son manejados.	Rendimiento
9	Será distribuido bajo licencia libre de código abierto GPL (*General Public License*), permitiéndose la modificación, uso y distribución libre.	Licencia
10	Debe permitir su actualización y mantenimiento.	Soporte
11	Un usuario autorizado y con experiencia podrá interactuar con el componente, por lo que deberá ser intuitivo.	Usabilidad

12	El componente deberá ser empaquetado en un .jar que garantice su portabilidad e integración con la herramienta de indexado de la información.	Portabilidad

2.4 Historias de usuario

Las Historias de Usuario son un enfoque de requerimientos ágil que se focaliza en establecer conversaciones sobre las necesidades de los clientes. Son descripciones cortas y simples de las funcionalidades del sistema, narradas desde la perspectiva de la persona que desea implementar cada funcionalidad (Izaurralde, 2013).

A continuación, se muestran las historias de usuarios descritas para cada requisito funcional.

Tabla 4: Historia de usuario #1.

Número: 1	**Nombre de requisito:** Obtener documentos de Apache Solr.
Programador: David Álvarez Santes	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo:** 48 horas
Riesgo: Alto	**Tiempo real:** 40 horas
Descripción: Establecer flujo de información entre el **componente** de categorización semántica de documentos y el servidor de indexación Apache Solr. Permite consultar los documentos indexados en el servidor obteniéndose los valores de los campos id y contenido.	
Observaciones:	
Prototipo de Interface: No aplica.	

Tabla 5: Historia de usuario #2.

Número: 2	**Nombre de requisito:** Identificar las entidades nombradas.
Programador: David Álvarez Santes	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo:** 48 horas
Riesgo: Alto	**Tiempo real:** 32 horas
Descripción:	

Comienza con la eliminación de todos los caracteres innecesarios como los números, palabras superfluas, vacías o "***Stopword***", representan datos que no aportan información, como son los artículos, preposiciones, conjunciones entre otras palabras a través de un archivo (*stopwordsES*) consumido por el sistema en su directorio raíz.
Observaciones:
Prototipo de Interface: No aplica.

Tabla 6: Historia de usuario #3.

Número: 3	**Nombre de requisito:** Almacenar las entidades nombradas.
Programador: David Álvarez Santes	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo:** 24 horas
Riesgo: Medio	**Tiempo real:** 16 horas
Descripción: Se actualiza el campo "***keywords***" en los documentos indexados en el servidor Apache Solr después de realizado el proceso de búsqueda y procesamiento de las entidades nombradas.	
Observaciones:	
Prototipo de Interface: No aplica.	

Tabla 7: Historia de usuario #4.

Número: 4	**Nombre de requisito:** Identificar los conceptos en la ontología.
Programador: David Álvarez Santes	**Iteración Asignada:** 2
Prioridad: Alta	**Tiempo:** 72 horas
Riesgo: Alto	**Tiempo real:** 60 horas
Descripción: Se identifican los conceptos de la ontología y se analiza su relación con las entidades nombradas del documento mediante la utilización de la función de similitud coseno.	
Observaciones:	
Prototipo de Interface: No aplica.	

Tabla 8: Historia de usuario #5.

Número: 5	**Nombre de requisito:** Almacenar los conceptos.
Programador: David Álvarez Santes	**Iteración Asignada:** 3
Prioridad: Media	**Tiempo:** 24 horas
Riesgo: Alto	**Tiempo real:** 18 horas
Descripción: Se actualiza el campo "***semantic_keywords***" en los documentos indexados en el servidor Apache Solr a partir de cada concepto asociado.	
Observaciones:	
Prototipo de Interface: No aplica.	

Tabla 9: Historia de usuario #6.

Número: 6	**Nombre de requisito:** Categorizar documento.
Programador: David Álvarez Santes	**Iteración Asignada:** 2
Prioridad: Alta	**Tiempo:** 96 horas
Riesgo: Alto	**Tiempo real:** 96 horas
Descripción: Se aplica el algoritmo de categorización seleccionado, para agrupar los documentos según su similitud.	
Observaciones:	
Prototipo de Interface: No aplica.	

Tabla 10: Historia de usuario #7.

Número: 7	**Nombre de requisito:** Obtener la relevancia de las categorías.
Programador: David Álvarez Santes	**Iteración Asignada:** 2
Prioridad: Alta	**Tiempo:** 72 horas
Riesgo: Alto	**Tiempo real:** 72 horas
Descripción: Se selecciona las categorías más relevantes a las que corresponde de acuerdo a los datos obtenidos del algoritmo de categorización.	

Observaciones:
Prototipo de Interface: No aplica.

Tabla 11: Historia de usuario #8.

Número: 8	**Nombre de requisito:** Almacenar las categorías.
Programador: David Álvarez Santes	**Iteración Asignada:** 3
Prioridad: Media	**Tiempo:** 32 horas
Riesgo: Alto	**Tiempo real:** 28 horas
Descripción: Se actualiza el campo "***semantic_category***" en los documentos indexados en el servidor Apache Solr a partir de la categoría más relevante a la que pertenece el documento.	
Observaciones:	
Prototipo de Interface: No aplica.	

2.5 Arquitectura del sistema

La arquitectura basada en componentes es una rama de la Ingeniería de *Software*, que realiza énfasis en descomponer el *software* en componentes funcionales. La descomposición permite convertir componentes preexistentes en piezas más grandes de *software*, este proceso de construcción de una pieza de *software*, da origen al principio de reutilización del *software* (Scribd, 2018).

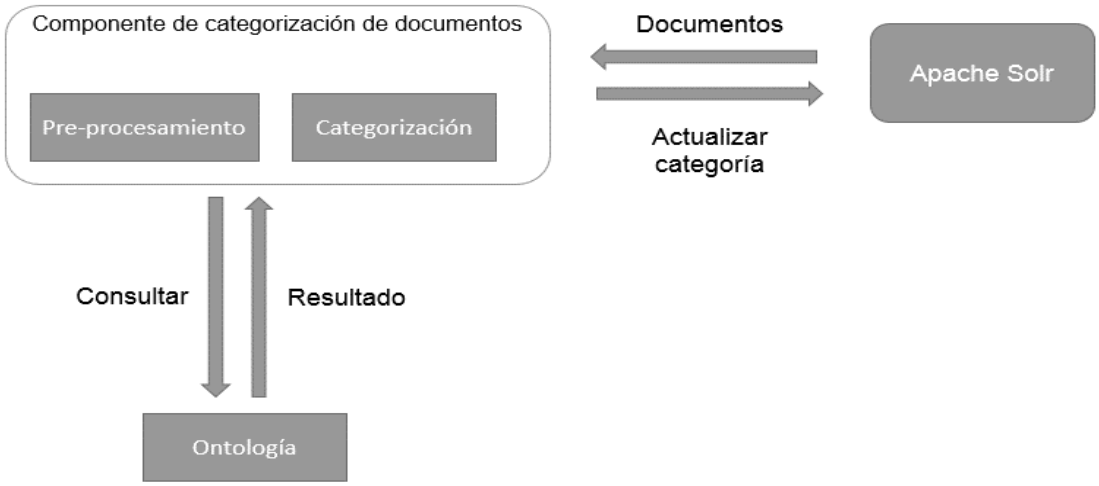


Figura 4: Arquitectura del componente de categorización semántica de documentos.

El componente **Apache Solr** es el responsable de almacenar los documentos indexados, posteriormente el **componente de categorización semántica de documentos** consultará los documentos almacenados para un Pre-procesamiento de datos y consecutivamente se le realizará el proceso de categorizar con la utilización de los conceptos asociados a la **Ontología**, donde una vez finalizado el proceso devolverá la categoría más relevante.

2.6 Patrones de diseño

Los patrones de diseño representan la descripción de un problema particular y recurrente, que aparece en contextos específicos y presenta un esquema genérico para su solución; este último se especifica mediante la descripción de los componentes que la constituyen, sus responsabilidades, desarrollos y la forma de colaboran entre sí (Larman, 2004).

GRASP

Según Larman (2004), son enfocados en los principios fundamentales de asignación correcta de responsabilidades en el diseño orientado a objetos. Describen los patrones que forman la solución y su aplicación en el desarrollo de los componentes. A continuación, se muestran los patrones utilizados y un ejemplo de cómo se evidencia en la aplicación.

- **Creador:** guía la asignación de responsabilidades relacionadas con la creación de objetos, la intención básica del patrón Creador es encontrar un creador que necesite conectarse al objeto creado en alguna situación (Larman, 2004). En el componente de categorización de documentos se encuentra la clase **Categorizar**, responsable de crear y cargar los objetos necesarios para el resto de las clases existentes.

```
public class Categorizar {

    public static void main(String[] args) throws SolrServerException, IOException {
    Analyzer analyzer = new Analyzer();
        analyzer.f main();
     Conectar_Solr solr = new Conectar_Solr();
     SolrDocumentList list = solr.CantDoc();
```

Figura 5: Ejemplo de utilización del patrón Creador.

- **Bajo Acoplamiento:** impulsa la asignación de responsabilidades de forma en que su localización no incremente el acoplamiento hasta un nivel de resultados negativos que pueden producir un alto acoplamiento. El Bajo Acoplamiento soporta el diseño de clases que son más independientes, reduciéndose el impacto al cambio (Larman, 2004). Las clases **Analyzer** y **Conectar_Solr** no tienen relaciones entre ellas, solo se comunican con la clase **Categorizar**, y pueden ser reutilizadas como componentes para otros sistemas sin que ocurran grandes impactos por los cambios. En la Figura 5 se detalla como la clase **Categorizar** carga los datos necesarios para **Analyzer** y **Conectar_Solr**.
- **Alta cohesión:** La información que almacena una clase es coherente y está relacionada con la clase (Larman, 2004). En el componente es utilizado este patrón en las clases **Analyzer**, **Conectar_Solr** y **Categorizador**.
- **Experto:** plantea que se debe asignar una responsabilidad al experto en información, o la clase que tiene los datos necesarios para cumplir con este patrón (Larman, 2004). La clase **Analyzer** es un ejemplo de Experto, encargada del pre-procesamiento del contenido de cada documento y es experta en esa función.

GoF

Describen las formas comunes en que diferentes tipos de objetos pueden ser organizados para trabajar unos con otros. Tratan la relación, combinación y la formación de estructuras de mayor complejidad entre clases (Guerrero et al., 2013). En el desarrollo del componente se identificó ***Iterator*** (Iterador), encargado de realizar recorridos sobre objetos compuestos de forma independiente a su implementación. A continuación, se muestra un ejemplo de su utilización en una lista.

```
Iterator<SolrDocument> iter = list.iterator();
```

2.7 Diagrama de clases de diseño

El diagrama de clases de diseño describe gráficamente las especificaciones de las clases de *software* (Larman, 2004). En el diagrama de clases del componente a desarrollar, se muestra la pertenencia de clases con sus relaciones.

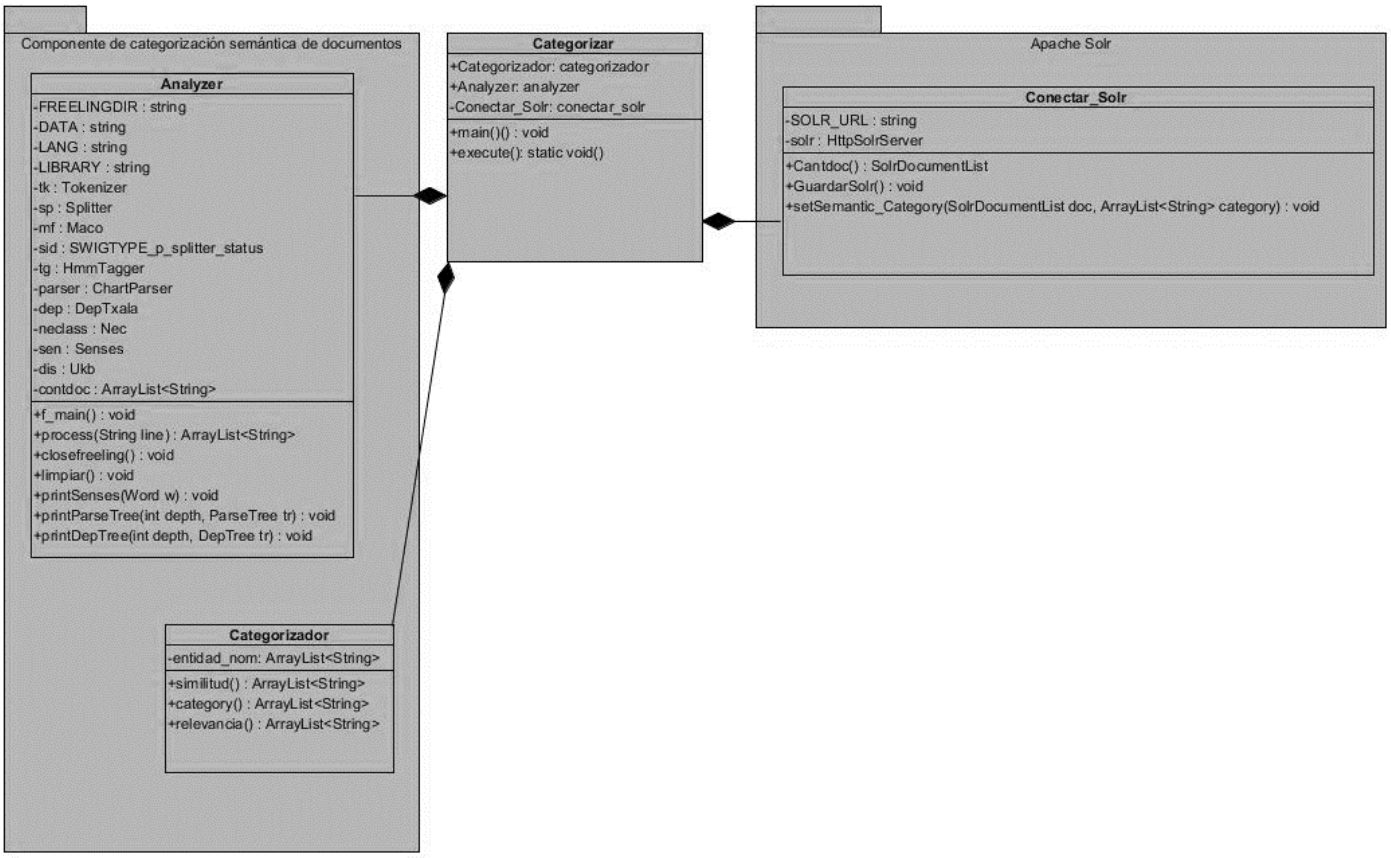


Figura 6: Diagrama de clases de diseño del componente de categorización semántica de documentos.

Analyzer: Es la clase encargada del pre-procesamiento de los documentos consultados.

Categorizar: Es la clase principal, encargada de establecer el flujo de datos entre el resto de clases.

Categorizador: Es la clase encargada de categorizar los documentos, mediante el algoritmo de categorización.

Conectar_Solr: Es la clase encargada del flujo de comunicación con el servidor Apache Solr.

2.8 Modelo de datos

Orientado a presentar los elementos que procesan el sistema, la composición y atributos de los mismos, muestra la ubicación en donde serán almacenados estos objetos, su relación y los procesos que los transforman (Pressman, 2010). A continuación, se muestra el modelo de datos con que será implementado el componente de categorización semántica de documentos.

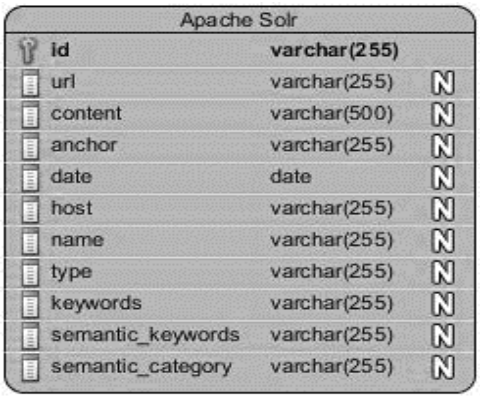


Figura 7: Diagrama de modelo de datos de Apache Solr.

En la Figura 7 se muestra el nombre de los campos y su formato para el modelo de datos de **Apache Solr**. Para el desarrollo del componente se requiere la extracción del valor de los campos "***id***" y "***content***". Los nuevos campos "***keywords***", "***semantic_keywords***" y "***semantic_category***", almacenan las entidades nombradas, los conceptos obtenidos de la representación ontológica y las categorías semánticas respectivamente.

2.9 Diagrama de Secuencia

El diagrama de secuencia realiza una representación del modo en que los eventos del sistema causan el flujo de uno a otro como función del tiempo. Representa las clases y eventos que hacen que el comportamiento avance de una clase a otra (Pressman, 2010).

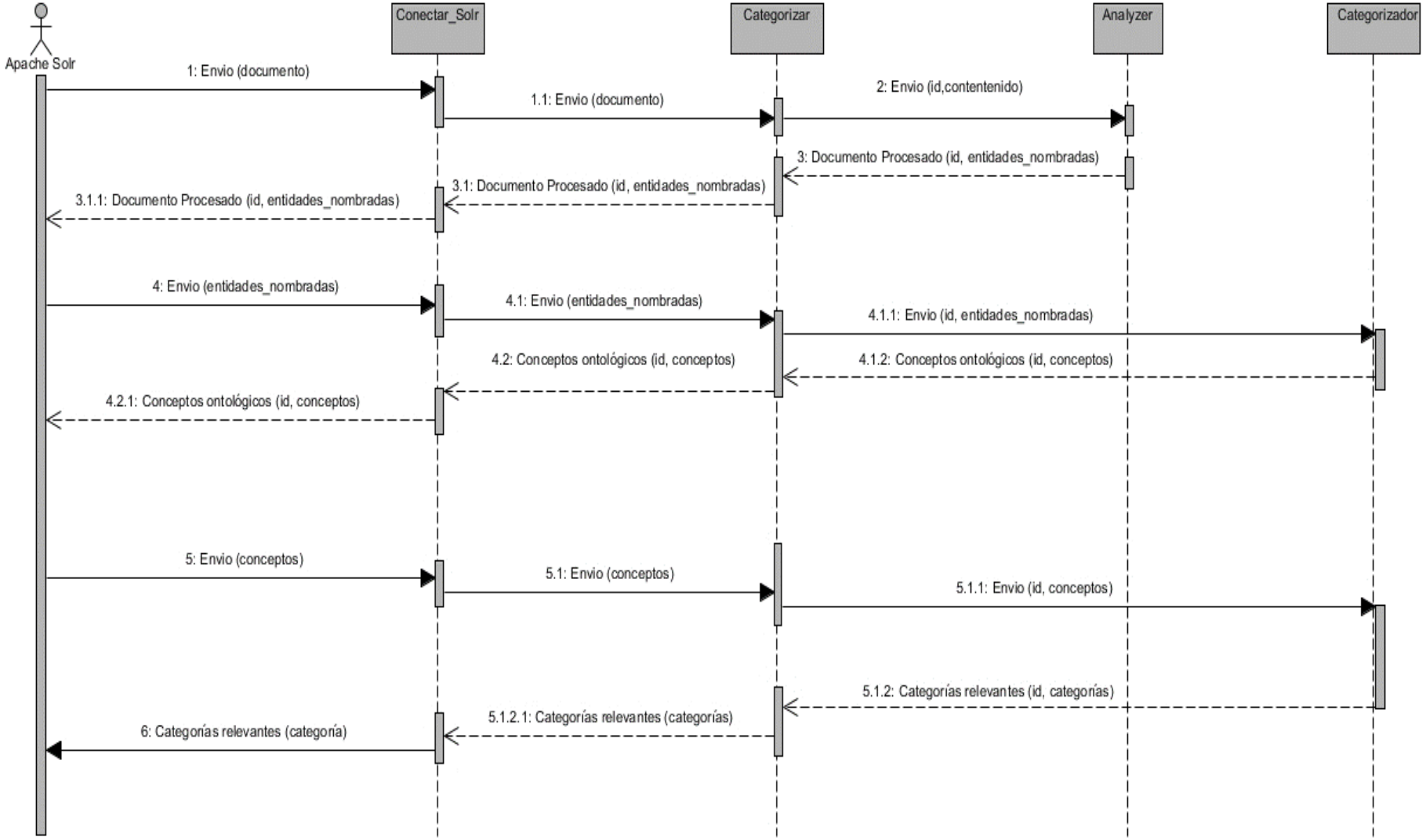


Figura 8: Diagrama de secuencia del componente de categorización semántica de documento.

2.10 Modelo de despliegue

Un modelo de despliegue representa la relación física que se establece entre los distintos componentes o nodos que describen la topología de un sistema. Estos definen la configuración de los elementos de hardware, y detalla cómo los elementos y artefactos del *software* se relacionan en esos nodos (SparxSystems, 2018).

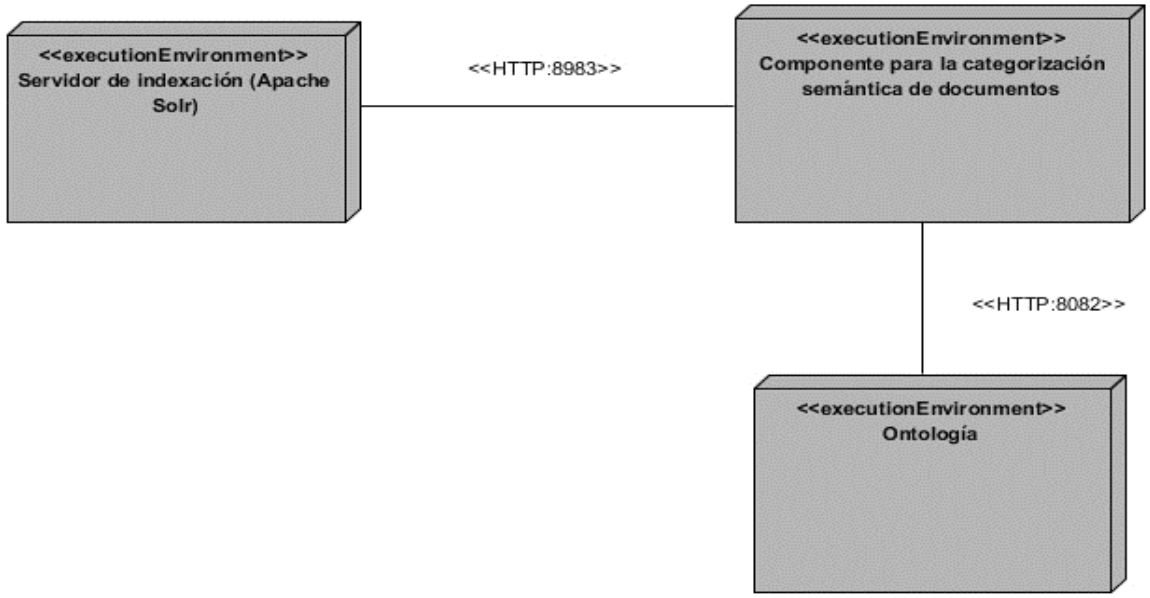


Figura 9: Diagrama de despliegue del componente categorización semántica de documentos.

La Figura 9 muestra el nodo **Apache Solr** como servidor de indexación; esta herramienta recibe consultas del nodo **componente para la categorización semántica de documentos** mediante el protocolo HTTP por el puerto 8983, la aplicación a su vez intercambia información con el nodo **Ontología** por HTTP en el puerto 8082.

Conclusiones del capítulo

Con el estudio de los temas referentes al análisis y diseño de la propuesta del componente para la categorización semántica de documentos se pudo arribar a las siguientes conclusiones:

- Los requisitos funcionales y no funcionales obtenidos a partir del levantamiento de requisitos responden a que la propuesta de solución satisface las necesidades del cliente.
- Los artefactos generados según la metodología AUP-UCI, la arquitectura y los patrones de diseño crearon las bases necesarias para la construcción de la propuesta de solución.

Capítulo 3: "Implementación y validación del componente de categorización semántica de documentos"

En el presente capítulo se caracterizan los elementos definidos en el proceso de desarrollo, para obtener un producto de calidad. Es presentado y descrito el diagrama de componentes para una mayor comprensión de los elementos físicos del sistema y sus relaciones. En esta sección se incluye la especificación del estándar de codificación desarrollado para la implementación de la aplicación. Se realizan las pruebas del *software*, con el objetivo de comprobar el correcto funcionamiento del componente y validar la propuesta de solución.

3.1 Diagrama de componentes

El diagrama de componentes permite visualizar la estructura de un sistema informático mediante el análisis de las partes que lo conforman. Los componentes son utilizados como una unidad de composición independiente e indispensable dentro de un sistema, donde se identifica las dependencias que existen entre cada uno de los elementos. Algunos ejemplos de componentes físicos lo constituyen los archivos, módulos, librerías, ejecutables y binarios (Sommerville, 2005).

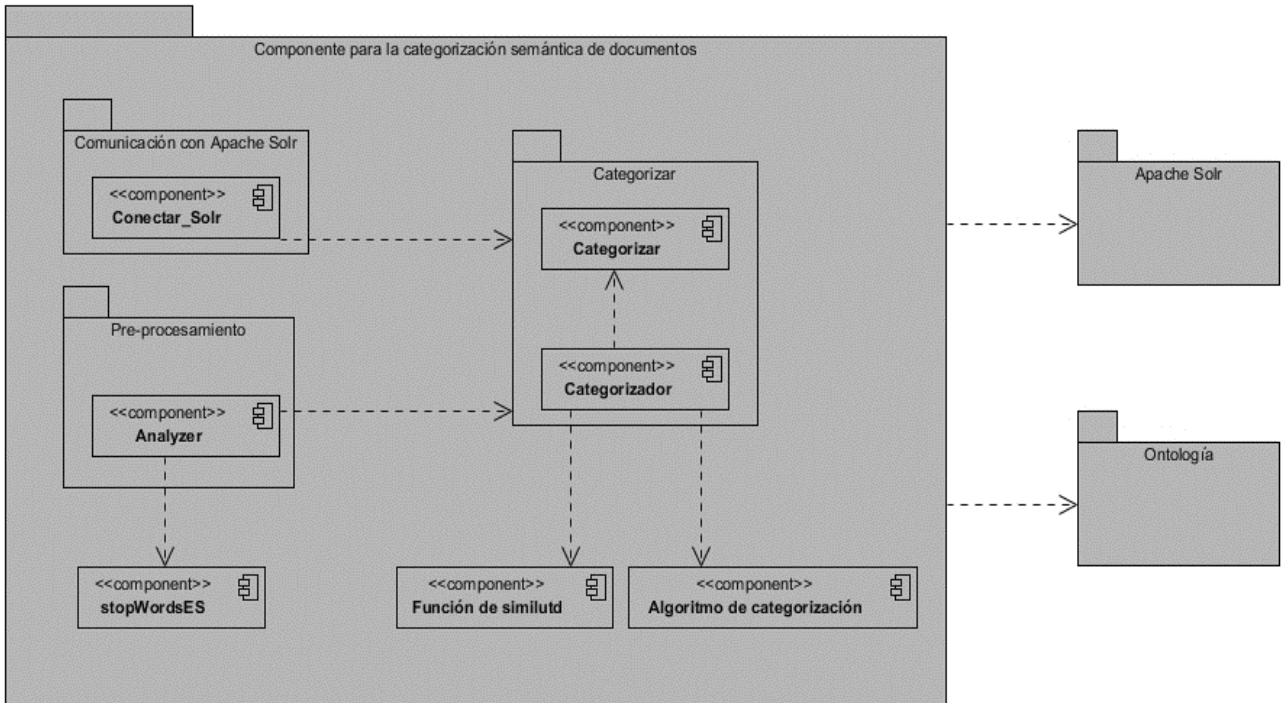


Figura 10: Diagrama de Componente de la categorización semántica de documentos.

Descripción de los componentes del diagrama

- **Comunicación con Apache Solr:** Este componente es el responsable de la extracción, inserción y actualización de información con el servidor de indexación Apache Solr. Contiene la clase Conectar_Solr que implementa los métodos (CantDoc, GuardarSolr) necesarios para que los demás componentes puedan ejecutar su función.
- **Pre-procesamiento:** Contiene la clase *Analyzer* y el archivo *stopWordsES.txt* donde se desarrolla el pre-procesamiento de los documentos indexados por Apache Solr mediante los métodos de la herramienta Freeling y los métodos de la clase *Cleaner*. Este componente tiene como resultado la adquisición de las entidades nombradas de los documentos.
- **Categorizar:** Este componente contiene la clase principal Categorizar, responsable de solicitar y brindar toda la información requerida por el resto de los componentes. Dentro del componente también está la clase Categorizador, que ejecuta la función de similitud y el algoritmo de categorizar para obtener la representación entre las entidades nombradas y los conceptos asociados a la categoría de la ontología.
- **Algoritmo de categorización:** Implementa el algoritmo para determinar las categorías relevantes a través de los conceptos asociados a la ontología.
- **Apache Solr:** motor de búsqueda basado en la biblioteca Java del proyecto Lucene, con APIs en XML/HTTP y JSON, resaltado de resultados, búsqueda por facetas, caché, y una interfaz para su administración.
- **Ontología:** es una definición formal de tipos, propiedades, y relaciones entre entidades que fundamentalmente existen para un dominio de discusión en particular.

3.2 Estándar de codificación

Los estándares de codificación son un conjunto de reglas o patrones de codificación a seguir por los desarrolladores con el objetivo de establecer un orden y un formato común en el código fuente del sistema en desarrollo. Estos estándares cumplen con el principio de legibilidad y mantenibilidad, correspondientes al objetivo de que el programador entienda el código y sea capaz de modificarlo (Microsoft, 2016). En la implementación del componente es utilizado la siguiente estandarización:

- ✓ Se utiliza el margen adicional de cuatro espacios.

```
public Cleaner() throws FileNotFoundException, IOException {
    loadStopWords("stopwordsES");
}
```

Figura 11: Fragmento del código fuente de la clase Analyzer.

- ✓ Las variables son declaradas de forma independiente, en líneas nuevas y no en las mismas sentencias.

```
public void loadStopWords(String path) throws FileNotFoundException, IOException{
    stopWordsSet = new HashSet();

    FileReader fr = new FileReader(path);
    BufferedReader br = new BufferedReader(fr);

    String line;
    while ((line = br.readLine()) != null){
        stopWordsSet.add(line);
    }
```

Figura 12: Fragmento del código fuente de la clase Analyzer.

Las implementaciones de las clases tienen la siguiente estructura:

1. Atributos de la clase.
2. Constructores
3. Métodos de acceso.

4. Métodos.

```
public final class Cleaner {
    private Set<String> stopWordsSet;

    public Cleaner() throws FileNotFoundException, IOException {
        loadStopWords("stopwordsES");
    }

    public Cleaner(String stopWordsPath) throws FileNotFoundException, IOException{
        loadStopWords(stopWordsPath);
    }

    public void loadStopWords(String path) throws FileNotFoundException, IOException{
        stopWordsSet = new HashSet();

        FileReader fr = new FileReader(path);
        BufferedReader br = new BufferedReader(fr);

        String line;
        while ((line = br.readLine()) != null){
            stopWordsSet.add(line);
        }
    }
```

Figura 13: Fragmento del código fuente de la clase Analyzer.

- Los atributos relacionados se declaran en una misma línea. El resto se declara en líneas separadas.

```
public class Conectar_Solr {

    private String SOLR_URL = "http://localhost:8983/solr/ecured";
    private HttpSolrServer solr = new HttpSolrServer(SOLR_URL);
```

Figura 14: Fragmento del código fuente de la clase Conectar-Solr.

- Los comentarios se declaran con una gramática correcta y contienen los signos de puntuación apropiados.

```
// Create analyzers.
// language detector. Used just to show it. Results are printed  but ignored.
// See below.
LangIdent lgid = new LangIdent(DATA + "/common/lang_ident/ident.dat");
System.out.println("Cargando Sub-modulos");
tk = new Tokenizer(DATA + LANG + "/tokenizer.dat");
sp = new Splitter(DATA + LANG + "/splitter.dat");
sid = sp.openSession();
```

Figura 15: Fragmento del código fuente de la clase Analyzer.

La utilización de los estándares de codificación garantizó que el código resultante fuera de sencilla comprensión, independientemente del desarrollador del producto. Permitió asegurar calidad, legibilidad y la no contención de errores para un futuro mantenimiento.

3.3 Estrategia de pruebas

Según Navarro (2014) las pruebas de *software* comprenden una fase del proceso de desarrollo que se centra en asegurar la calidad, fiabilidad y robustez de un *software*, dentro de un contexto o escenario donde está previsto que este sea utilizado. Se encuentra encaminado a medir el cumplimiento de las funcionalidades establecidas por el cliente, reduciendo de esta manera el número de errores no detectados. Esta fase es una de las más costosas del ciclo de vida del *software*.

- ✓ Pruebas unitarias.
- ✓ Pruebas de integración.
- ✓ Pruebas funcionales.

Pruebas unitarias

Las pruebas unitarias realizadas utilizan el método de caja blanca y de la técnica del camino básico. El método del camino básico permite al diseñador de casos de prueba derivar una medida de complejidad lógica de un diseño procedural y utilizar esa medida como guía para la definición de un conjunto básico de caminos de ejecución (Pressman, 2006). Las unidades de prueba más pequeñas son las operaciones dentro de la clase. A continuación, se realiza la técnica de camino mínimo al método *cleanText* de la clase *Cleaner* que posibilita la obtención del texto procesado para la posterior clasificación.

```
1   public String cleanText(String text){
2           text = text.replaceAll("[^A-Za-z\\sñáéíóú]", "").toLowerCase();
            text = text.replaceAll("^ +| +$|( )+", "$1");
            String[] words = text.split(" ");
            StringBuilder builder = new StringBuilder();
3           for (String word : words) {
4               if (!stopWordsSet.contains(word)) {
5                   if(builder.length() > 0)
6                       builder.append(" ");
7                   builder.append(word);
8                }
9           }
10          return builder.toString();
11   }
```

Figura 16: Implementación del método cleanText de la clase Cleaner.

Luego de enumerar el código, se diseña la gráfica del método en la Figura 17, que describe el flujo de control lógico mediante la utilización de nodos y aristas.

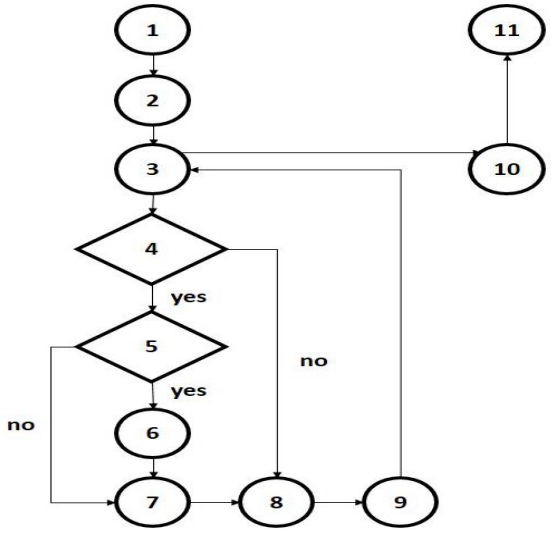


Figura 17: Grafo de flujo.

A partir del grafo obtenido con 11 nodos y 13 aristas se calcula la complejidad ciclomática (V(G)). La complejidad ciclomática se basa en el recuento del número de caminos lógicos individuales contenidos en un programa. Para hallar la complejidad ciclomática, el programa se representa como un grafo, y cada instrucción que contiene, un nodo del grafo. Las posibles vías de ejecución a partir de una instrucción (nodo) se representan en el grafo como aristas.

V(G) = cantidad_aristas – cantidad_nodos + 2 (4)

V(G) = 13 – 11 + 2 = 4

Una ruta independiente es cualquier trayecto del programa que ingrese al menos un nuevo conjunto de instrucciones de procesamiento o una nueva condición (Pressman, 2006). La cantidad de rutas independientes se establecen por la complejidad ciclomática; fueron identificadas 4 rutas, tal y como se muestra en la Tabla 12:

Tabla 12: Listado de caminos independientes.

No. Ruta	Camino
1	1-2-3-10-11
2	1-2-3-4-5-6-7-8-9-3-10-11
3	1-2-3-4-5-7-8-9-3-10-11
4	1-2-3-4-8-9-3-10-11

El valor de V(G) ofrece además un límite superior del número de pruebas que debe diseñarse y ejecutarse para garantizar la cobertura de todas las instrucciones (Pressman, 2006). A continuación, se diseñan casos de pruebas para ser aplicados a cada ruta independiente:

Tabla 13: Caso de prueba unitaria. Ruta 1.

Caso de Prueba de Unidad	
No. Ruta: 1	**Ruta:** 1-2-3-10-11
Descripción de la prueba: Obtener cadena de palabras relevantes para la categorización.	
Entrada: Se envía por parámetro un texto.	
Resultado esperado: Devolver cadena de palabras.	
Evaluación de la prueba: Satisfactorio. Se obtiene el listado con las palabras relevantes.	

Tabla 14: Caso de prueba unitaria. Ruta 2.

Caso de Prueba de Unidad	
No. Ruta: 2	**Ruta:** 1-2-3-4-5-6-7-8-9-3-10-11
Descripción de la prueba: Obtener cadena de palabras relevantes para la categorización.	
Entrada: Se envía por parámetro un texto con palabras no relevantes.	
Resultado esperado: Eliminar palabras no relevantes del texto.	
Evaluación de la prueba: Satisfactorio. Se obtiene el listado con las palabras relevantes.	

Tabla 15: Caso de prueba unitaria. Ruta 3.

Caso de Prueba de Unidad	
No. Ruta: 3	**Ruta:** 1-2-3-4-5-7-8-9-3-10-11
Descripción de la prueba: Obtener cadena de palabras relevantes para la categorización.	
Entrada: Se envía por parámetro un texto.	
Resultado esperado: Dejar un espacio después de la última palabra de la cadena conformada.	
Evaluación de la prueba: Satisfactorio. El espacio fue generado.	

Tabla 16: Caso de prueba unitaria. Ruta 4.

Caso de Prueba de Unidad	
No. Ruta: 4	**Ruta:** 1-2-3-4-8-9-3-10-11
Descripción de la prueba: Obtener cadena de palabras relevantes para la categorización.	
Entrada: Se envía por parámetro un texto sin *stopwords*.	
Resultado esperado: Devolver cadena de palabras.	
Evaluación de la prueba: Satisfactorio. Se obtuvo la cadena de palabras.	

Resultados de la prueba

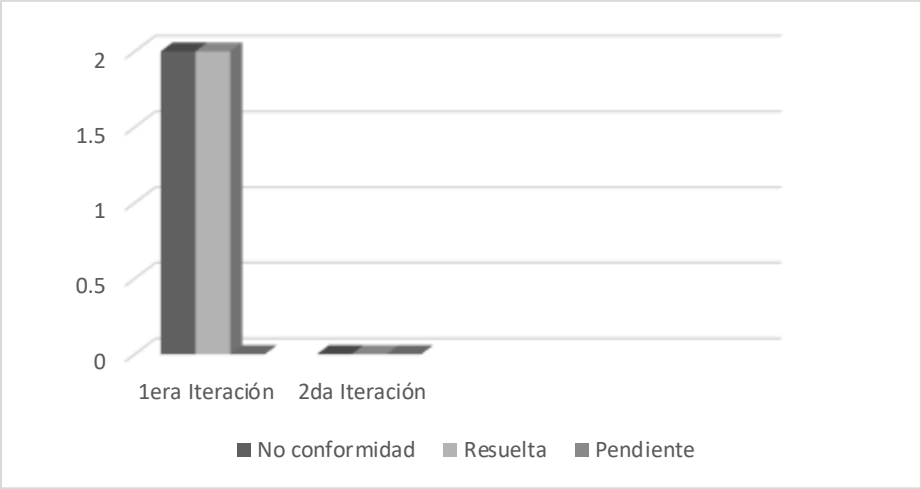


Figura 18: Gráfico de resultados de las pruebas de unitarias.

En la primera iteración fueron detectadas dos no conformidades relacionadas a errores de validación y al tratamiento de excepciones. Las deficiencias fueron corregidas mediante una revisión detalla al código y al correcto lanzamiento de excepciones. Fue realizada una segunda iteración donde no se evidenciaron no conformidades.

Pruebas de integración

Las pruebas de integración se centran en comprobar que los módulos probados por separado funcionen en conjunto, con el objetivo de verificar que interactúan correctamente a través de sus interfaces y cubren las funcionalidades establecidas en los requisitos (Pressman, 2010). En la Tabla 17 se muestran los resultados de la prueba de integración:

Tabla 17: Resultados de la prueba de integración.

Componente	Funcionalidad	Funcionalidad Integridad	Resultado de la prueba
Servidor Apache Solr	Obtener documentos almacenados en Apache Solr.	Se realiza la extracción de documentos desde el servidor de indexación hacia el componente.	En la primera iteración realizada se evidencia dos no conformidades, relacionadas a una mala referencia entre las clases Categorizar y Conectar_Solr y la

			segunda, a falta de librerías, ambas fueron corregidas. Fue realizada una segunda iteración donde no fueron encontradas deficiencias.
	Agregar y actualizar la información en Apache Solr.	Permite agregar y actualizar las entidades nombradas, conceptos y categorías relevantes en el servidor de indexación.	El componente inserta y actualiza, de forma satisfactoria, la información en Apache Solr.
Spring Boot	Permite la relación entre el componente y la ontología.	La funcionalidad que integra es la realización de consultas, obteniéndose relaciones entre los conceptos y las entidades nombradas mediante la función de similitud, y categorizar el documento a través del algoritmo de categorización.	En una primera iteración fue encontrada una incoherencia relacionado a un mal tratamiento de excepciones, fue debidamente corregida. Fue realizada una segunda iteración donde no fueron encontradas deficiencias.

Resultados de la prueba

Las ejecuciones de las pruebas de integración permitieron verificar el trabajo en conjunto del componente con los servidores. Se realizó especial énfasis en la comunicación entre el componente y Apache Solr, con el objetivo de detectar incoherencias en el funcionamiento de la aplicación. El análisis de los resultados obtenidos de las pruebas concluye con la solución de las incoherencias encontradas y la correcta integración con los servidores, necesarios para la categorización semántica.

Pruebas funcionales

Las pruebas funcionales se centran en comprobar que los sistemas desarrollados funcionan acorde a las especificaciones funcionales y requisitos del cliente. Este servicio ayuda a su organización a detectar los posibles defectos derivados de errores en la fase de programación (Globe, 2018).

Tabla 18: Caso de prueba a la HU "Almacenar entidades nombradas".

Escenario	Descripción	V1	V2	V3	R/ del componente	Flujo Central
EC 1.1 Almacenar las entidades nombradas	El componente de categorización semántica de documentos almacena las entidades nombradas en el servidor de indexación Apache Solr.	V Fidel	V Castro	V Ruz	El componente almacena la entidad nombrada en el servidor de indexación.	El componente analiza e identifica la entidad nombrada en un documento mediante la herramienta Freeling y lo almacena en el campo *"keywords"* del servidor de indexación Apache Solr del documento correspondiente.

Tabla 19: Caso de prueba a la HU "Categorizar documento".

Escenario	Descripción	V1	V2	R/ del componente	Flujo Central
EC 2.1 Introducir categorías	El componente de categorización semántica de documentos aplica el algoritmo de categorización.	V Artículo de deporte[5]	V Artículo de historia[6]	El componente categoriza el artículo de forma satisfactoria	El componente calcula las categorías más relevantes a las que pertenece un documento a través de los conceptos

[5] https://www.ecured.cu/Juegos_Panamericanos

[6] https://www.ecured.cu/Guerra_de_los_Diez_A%C3%B1os_(1868-1878)

					asociados a categorías de la ontología mediante un algoritmo de categorización.

Tabla 20: Caso de prueba a la HU "Identificar las entidades nombradas".

Escenario	Descripción	V1	V2	V3	R/ del componente	Flujo Central
EC 3.1 Identificar las entidades nombradas de forma correcta	El componente de categorización semántica de documentos identifica las entidades nombradas de los documentos indexados por Apache Solr mediante la herramienta Freeling.	V Barcelona	V Messi	V Piqué	El componente identifica la entidad nombrada.	El componente procesa el documento e identifica las entidades nombradas mediante la herramienta Freeling.
EC 3.2 Identificar las entidades nombradas de forma incorrecta	El componente identifica las entidades nombradas de los documentos indexados por Apache Solr a través de la	I último	I debajo	V Roma	El componente identifica la entidad nombrada de forma incorrecta.	La aplicación utiliza Freeling para el procesamiento de los documentos, en la primera iteración es encontrada una no conformidad que identifica las palabras vacías o

	herramienta Freeling.					*stopword*, como entidades nombradas. Esta no conformidad fue erradicada mediante el desarrollo de la clase *Cleaner*, responsable de realizar la limpieza de estas palabras. Fue realizado una segunda iteración en la que no se evidenció deficiencias.

Tabla 21: Caso de prueba a la HU "Identificar los conceptos en la ontología".

Escenario	**Descripción**	**V1**	**V2**	**V3**	**R/ del componente**	**Flujo Central**
EC 4.1 Identificar los conceptos en la ontología	El componente de categorización semántica de documentos analiza la similitud entre las entidades nombradas y los conceptos presentes en la ontología a través	V medalla	V festival	V robot	El componente retorna la mayor similitud entre las entidades nombradas y los conceptos asociados a la ontología mediante la utilización de una	El componente calcula la relación entre las entidades nombradas y los conceptos de la ontología mediante el uso de una función de similitud. Se realizó dos iteraciones, la primera con la función de Levenshtein y la segunda con el

	de la utilización de una función de similitud.				función de similitud.	Coseno, se obtuvo como resultado 70% y 95% de concordancia para cada caso, respectivamente. La prueba demostró la utilización de función Coseno para la similitud de términos en la aplicación.

Resultados de la prueba

La realización de las pruebas funcionales posibilitó la detección y corrección temprana de errores en la aplicación. Fueron realizadas un total de cuatro iteraciones donde fueron encontradas dos no conformidades. La primera en el caso de prueba: identificar las entidades nombradas, asociada a que la herramienta Freeling devolvía palabras vacías como entidades nombradas, fue solucionada mediante la implementación de la clase *Cleaner*, encargada de apoyar el pre-procesamiento. La segunda no conformidad fue encontrada en el caso de prueba: identificar los conceptos en la ontología, estaba relacionada a la comparación de la función de similitud Levenshtein y Coseno, evidenciándose una mejoría en los resultados con el Coseno.

3.4 Validación de la hipótesis

Para la validación de la hipótesis científica se utiliza el método de consulta a expertos en su variante Delphi (Sánchez, 2015) siguiendo los puntos siguientes:

- Identificación de los posibles expertos.
- Selección de los expertos.
- Realización de consultas a expertos, procesamiento y valoración de la información obtenida.

Para la identificación de los posibles expertos, se tiene en consideración: experiencia laboral, disposición de participar en la encuesta, competencia, relación con este tipo de *software* y conocimiento sobre tecnologías de la web semántica. Estas características permitirán que el experto se involucre más con la propuesta de solución y facilite su implantación. En la Tabla 22 se muestran la selección de expertos:

Tabla 22: Expertos utilizados en la validación de la propuesta de solución.

No.	Experto	Entidad	Años de experiencia
1	Paúl Rodríguez Leyva	CIDI	7
2	Miguel Angel Chavez Alfonso	CIDI	7
3	Yusniel Hidalgo Delgado	FAC3-Programación	8
4	Walter Daniel Camejo López	CIDI	4
5	Yoan Antonio López Rodríguez	FAC3-Programación	10

Luego de la selección, se sometió a consideración un instrumento para validar el componente de categorización semántica de documentos. El instrumento se compone de 5 sentencias relacionadas con el funcionamiento del componente. Los expertos para expresar su opinión o valoración pueden utilizar las siguientes categorías:

- Muy adecuado (MA).
- Bastante adecuado (BA).
- Adecuado (A).
- Poco adecuado (PA).
- Inadecuado (I).

Tabla 23: Sentencias a evaluar por los expertos para validar la hipótesis científica.

No.	Sentencias plasmadas en la consulta realizada a expertos
1	¿El componente extrae e identifica los términos del documento correctamente?
2	¿El componente identifica y asocia los conceptos con los términos del documento correctamente?
3	¿El componente extrae correctamente las categorías asociadas al documento?
4	¿El componente clasifica correctamente los documentos?
5	Considera que la clasificación de documentos mejora el procesamiento de la información.

Se calcula el coeficiente de Kendall que permite el análisis de la concordancia en las valoraciones realizadas por los expertos (Sampieri, 2014). El coeficiente de concordancia (W) será un índice de la divergencia del acuerdo efectivo por parte de los expertos y se calcula con la siguiente ecuación (4):

$$W = 12S/K2\ (N3-N) \qquad (5)$$

Donde S representa el cuadrado de las desviaciones medias, K el número de expertos y N el número total de aspectos a evaluar, el valor de W oscila en el rango de 0 a 1. El valor 1 significa una concordancia de acuerdos total y el 0, un desacuerdo total.

Se aplica además la Prueba de Significación de Hipótesis para comprobar el grado de consideración de Kendall, planteándose la hipótesis nula y la alternativa de la siguiente forma:

- **H0:** no existe concordancia entre los expertos.
- **H1**: existe concordancia entre los expertos.

$$X2 = K\,(N-1)\,W \qquad (6)$$

X2 = 0.35

El resultado de X2 es comparado con el valor tabulado en la tabla de la distribución X2. Para tener un 95% de confianza se utilizará α = 0.05. Si se cumple que el resultado de X2 es menor que X2(α, N-1) se obtiene que 0.352 < 9.4877 entonces se valida la hipótesis alternativa H1, de que existe concordancia entre los expertos.

Los criterios aportados por los expertos son sometidos a una prueba estadística no paramétrica, que permite concluir la valoración final que tiene cada uno de estos aspectos a evaluar. Para los datos anteriores se debe confeccionar una distribución de frecuencia a partir de los datos primarios para cada uno de los aspectos sometidos a consulta (Castro, 2014).

Tabla 24: Distribución de frecuencia para los datos primarios obtenidos.

Categorías evaluativas	Frecuencia Absoluta	Frecuencia Relativa
Muy adecuado	23	0.92
Bastante adecuado	1	0.04
Adecuado	1	0.04
Poco adecuado	0	0
Inadecuado	0	0

Los resultados obtenidos en la validación se muestran en la Figura 19:

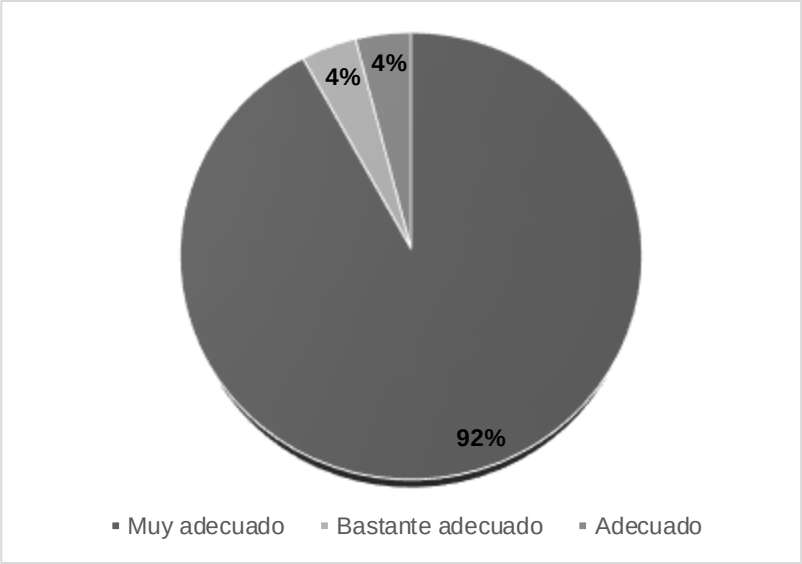


Figura 19: Comportamiento de la valoración de los expertos por categorías evaluadas.

Los datos de la Figura 19 muestran que un 92% de los aspectos fueron evaluados de muy adecuado, el 4% de bastante adecuado y un 4% de adecuado. El análisis de los resultados permitió identificar la existencia de una coherencia en las valoraciones realizadas. Los indicadores fueron evaluados satisfactoriamente y de esta forma se demuestra la validez de la hipótesis científica de la investigación, evidenciándose la mejora del procesamiento de la información.

Conclusiones del capítulo

En este capítulo se abordaron aspectos correspondientes a la implementación y validación del componente de categorización semántica de documentos, arribándose a las siguientes conclusiones:

- El diagrama de componentes facilitó la comprensión de la estructura general de la aplicación.
- Al aplicar los estándares de codificación se logró adoptar una estructura homogénea que facilita la comunicación y asegura la calidad, menos errores y fácil mantenimiento.
- La aplicación de pruebas unitarias, funcionales e integración permitieron identificar las principales deficiencias en el desarrollo del componente en etapas tempranas, así como solucionar los errores detectados y obtener un producto con un alto valor, pertenencia y utilidad.

Conclusiones

- El análisis de los referentes teóricos permitió definir la utilización de ontologías en el desarrollo del componente de categorización semántica de documentos para mejorar el procesamiento de la información.
- La modelación de los artefactos permitió obtener una arquitectura sólida de la aplicación y garantizó la base para la organización lógica del código fuente. La implementación del componente de categorización semántica mejora el procesamiento de los documentos en un Sistema de Recuperación de Información y proporciona una solución aceptable a la situación problemática existente.
- El componente de categorización semántica de documentos constituye una solución funcional y con calidad, conforme a los resultados obtenidos de las pruebas unitarias, funcionales y de integración aplicadas.
- La valoración realizada por los expertos demostró alto nivel de satisfacción con la aplicabilidad, actualidad y novedad del componente de categorización semántica de documentos y su contribución a mejorar el procesamiento de la información.

Recomendaciones

Una vez concluida la investigación y el desarrollo de la propuesta de solución, el autor del presente trabajo recomienda:

- ✓ Incrementar los conceptos y categorías presentes en la ontología con el objetivo de mejorar la recuperación de la información.
- ✓ Integrar el componente de categorización semántica de documentos en el entorno de la Plataforma c·u·b·a.

Bibliografía

ABADAL FALGUERAS, Ernest y CODINA BONILLA, Lluís. *Bases de datos documentales: características, funciones y método*. 2005. ISBN 8497562631.

AMÓN, Iván y JIMÉNEZ, Claudia. *Funciones de Similitud sobre Cadenas de Texto: Una Comparación Basada en la Naturaleza de los Datos*. In: *International Conference on Information Resources Management*. 2010. p. 13.

BAEZA-YATES, Ricardo A. y GONNET, Gaston H. *A New Approach to Text Searching*. *Communications of the ACM* [en línea]. 1992. Vol. 35, p. 74-82. Disponible en: http://dl.acm.org/citation.cfm?id=135243

BAEZA-YATES, Ricardo y RIBEIRO-NETO, Berthier. *Modern Information Retrieval: The Concepts and Technology behind Search*. *Information Retrieval*. 2011. Vol. 82, p. 944.

BERNERS-LEE, Tim, HENDLER, James y LASSILA, Ora. *The semantic web*. 2001. ISBN 9783540762973.

BILENKO, Mikhail y MOONEY, Raymond J. *Learning to Combine Trained Distance Metrics for Duplicate Detection in Databases*. *Artificial Intelligence*. 2002, p. 19.

BORDIGNON, Fernando y TOLOSA, Gabriel. *Recuperación de información: un área de investigación en crecimiento*. *Ciencias de la Información* [en línea]. 2007. Vol. 6, no. 1, p. 53-76. Disponible en: http://www.redalyc.org/articulo.oa?id=181414865002.

CACHEDA SEIJO, Fidel; FERNÁNDEZ LUNA, Juan Manuel; HUETE GUADIX, Juan Francisco. *Recuperación de Información. Un enfoque práctico y multidisciplinar*. Ra-Ma, 2011.

CANÓS, José H y LETELIER, MªCarmen Penadés Patricio. *Metodologías ágiles en el desarrollo de software*. In: *Metodologías Ágiles en el Desarrollo de Software*. 2012.

CASTRO, L. *Guía de gestión del riesgo tecnológico para el tratamiento de la seguridad durante el proceso de desarrollo de software*. 2014.

CHINNIYAN, Kavitha, GANGADHARAN, Sudha y SABANAIKAM, Kiruthika. *Semantic Similarity based Web Document Classification Using Support Vector Machine*. *The International Arab Journal of Information Technology*. 2017. Vol. 14, no. 3.

CHRISTEN, Peter. *A Comparison of Personal Name Matching: Techniques and Practical Issues*. In: *Sixth IEEE International Conference on Data Mining - Workshops (ICDMW'06)*. 2006. p. 290-294. ISBN 0-7695-2702-7.

COHEN, William W, RAVIKUMAR, Pradeep y FIENBERG, Stephen E. *A comparison of string metrics for matching names and records*. *KDD Workshop on Data Cleaning and Object Consolidation*. 2003. Vol. 3, p. 73-78.

CUBANIC. [en línea]. 2017. [Consultado el: 25 octubre 2017]. Disponible en: http://www.nic.cu/.

CUMULUS MEDIA. [en línea]. 2017. [Consultado el: 21 octubre 2017]. Disponible en: https://www.cumulus.com/

ECLIPSE FOUNDATION. *Eclipse Process Framework* (EPF). *OpenUP* [en línea]. 2012. [Consultado el: 15 octubre 2017]. Disponible en: http://epf.eclipse.org/wikis/openup/

EKAPUTRA, Fajar J, SABOU, Marta, SERRAL, Estefanía, KIESLING, Elmar y BIFFL, Stefan. *Ontology-Based Data Integration in Multi-Disciplinary Engineering Environments: A Review*. *Open Journal of Information Systms*. 2017. Vol. 4, no. 1, p. 1-26.

ESTRADA RAMOS, Luis Miguel. *Apache Solr, un motor de búsqueda de código abierto*. *Revista Digital Universitaria*. 2012. Vol. 13, no. 11, p. 1-9.

FALLIS, A.G. Metodologia Actual Metodologia XP. *Journal of Chemical Information and Modeling* [en línea]. 2013. Vol. 53, no. 9, p. 1689-1699. [Consultado el: 18 octubre 2017] Disponible en: http://blogs.unellez.edu.ve/dsilva/files/2014/07/Metodologia-XP.pdf

FATIMA, Shugufta y SRINIVASU, B. *Text Document categorization using support vector machine*. 2017.

FERNÁNDEZ, Miriam, CANTADOR, Iván, LÓPEZ, Vanesa, VALLET, David, CASTELLS, Pablo y MOTTA, Enrico. *Semantically enhanced Information Retrieval: An ontology-based approach*. *Journal of Web Semantics*. 2011. Vol. 9, no. 4, p. 434-452.

FLORES, E. *Metodologías Ágiles. Proceso Unificado Ágil (AUP)*. Universidad Unión Bolivariana, 2009.

FREELING. [en línea]. 2017. [Consultado el: 9 diciembre 2017]. Disponible en: http://nlp.lsi.upc.edu/freeling/

GADRI, S. y MOUSSAOUI, A. *Application of a new set of pseudo-distances in documents categorization*. *Neural Network World*. 2017. Vol. 27, no. 2, p. 231-245.

GARCÍA, E. *An ontology for human-like interaction systems*. Universidad Carlos III de Madrid, 2016.

GARCIA-HOLGADO, A y GARCÍA-PEÑALVO, F. J. *Metodologías de Ingeniería de Software*. 2018.

GIRONÉS, J. T. El gran libro de Android. México, Marcombo, S.A, 2012, p. 9-11.

GLOBE. *Pruebas de rendimiento* [en línea]. 2016. [Consultado el: 22 marzo 2018]. Disponible en: https://www.globetesting.com/pruebas-de-rendimiento/

GOTOH, Osamu. *An improved algorithm for matching biological sequences*. *Journal of Molecular Biology*. 1982. Vol. 162, no. 3, p. 705-708.

GRUBER, Thomas R. *Toward principles for the design of ontologies used for knowledge sharing*. *International Journal of Human - Computer Studies*. 1995. Vol. 43, no. 5-6, p. 907-928.

GUARINO, N. *Formal ontology, conceptual analysis and knowledge representation*. *Formal Ontology in Conceptual Analysis*. 1993. P. 1-21.

GUERRERO, C.A., SUÁREZ, J.M. y GUTIÉRREZ, L.E. *GOF (the gang of four) design patterns in the context of process development of web-oriented applications*. *Información Tecnológica*. 2013. Vol. 24, no. 3.

GUTIÉRREZ FARAONI, Federico Julián. *Desarrollo de una aplicación web con Spring Framework para un gestor de un recetario*. Universidad Politécnica de Madrid, 2015.

HEFFELFINGER, D. *Java EE 7 Development with NetBeans 8* [en línea]. Packt Publishing Ltd, 2015. [Consultado el: 13 octubre 2017] Disponible en: https://books.google.com.cu/books?hl=es&lr=&id=VY92BgAAQBAJ&oi=fnd&pg=PP1&dq=Java+EE+7+Development+with+NetBeans+8+&ots=P4THdparyA&sig=EE_ZjEVFhxNlFqv7KbxRmMJiKzk&redir_esc=y#v=onepage&q=Java EE 7 Development with NetBeans 8&f=false

HORROCKS, Ian, PATEL-SCHNEIDER, Peter F. y VAN HARMELEN, Frank. From SHIQ and RDF to OWL: *The making of a Web Ontology Language*. *Web Semantics*. 2003. Vol. 1, no. 1, p. 7-26.

IZAURRALDE, M. P. *Caracterización de Especificación de Requerimientos en entornos Ágiles: Historias de Usuario*. *Trabajo de especialidad*. 2013.

JANIK, Maciej y KOCHUT, Krys J. *Wikipedia in action: Ontological knowledge in text categorization*. In: *Proceedings - IEEE International Conference on Semantic Computing 2008, ICSC 2008*. 2008. p. 268-275. ISBN 9780769532790.

JARO, M. A. UNIMATCH, *a Record Linkage System: User's Manual*. *Bureau of the Census*. 1980.

JOHNSON, Rie y ZHANG, Tong. *Effective Use of Word Order for Text Categorization with Convolutional Neural Networks*. 2015. No. 2011, p. 103-112.

KORFHAGE, R. R. *Information storage and retrieval*. 2008.

LANTZ, Brett. *Classification using Naive Bayes*. In: *Machine Learning with R*. 2013. p. 396. ISBN 1782162151.

LARMAN, Craig. *Applying UML and Patterns*. 2004. ISBN 0131489062.

LEVENSHTEIN, Vladimir I. *Binary codes capable of correcting deletions, insertions, and reversals*. *Soviet Physics Doklady*. 1966. Vol. 10, no. 8, p. 707-710.

LUIS, Pedro, NAVARRO, Mateo, PÉREZ, Gregorio Martínez y RUIZ, Diego Sevilla. *Open HMI Tester: un Framework Open-source para Herramientas de Pruebas de Software 2 Arquitectura Open HMI Tester*. *Event London*. 2009. Vol. 3, no. 4, p. 61-66.

LUQUE, Constantino Malagón. *Clasificadores Bayesianos. El Algoritmo Naïve Bayes*. *Mayo*. 2003.

MAEDCHE, Alexander y STAAB, Steffen. *Learning Ontologies for the Semantic Web*. 2001.

MARTÍNEZ COMECHE, Juan Antonio. *Los modelos clásicos de Recuperación de información y su vigencia*. In: *Memoria del 3º seminario Hispano-mexicano de investigación en Bibliotecología y Documentación: Tendencias de la investigación en Bibliotecología y Documentación en México y España* [en línea]. 2006. p. 187-206. [Consultado el: 12 diciembre 2017]. ISBN 970-32-3961-7. Disponible en: http://eprints.ucm.es/5979/1/Modelos_RI_preprint.pdf

MARTÍNEZ, J. I. *Sistema de categorización de documentos para el buscador cubano Orión*. Universidad de las Ciencias Informáticas, 2016.

MATTHEWS, B. *Semantic web technologies*. *E-learning*. 2005. Vol. 6, p. 8.

MCCALLUM, Andres y NIGAM, Kamal. *A Comparison of Event Models for Naive Bayes Text Classification*. *AAAI/ICML-98 Workshop on Learning for Text Categorization* [en línea]. 1998. [Consultado el: 9 octubre 2017] P. 41-48. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.65.9324&rep=rep1&type=pdf

MICROSOFT. *Revisiones de código y estándares de codificación* [en línea]. [Consultado el: 15 marzo 2018]. Disponible en: https://msdn.microsoft.com/es-es/library/aa291591(v=vs.71).aspx

MORATO, J., SÁNCHEZ-CUADRADO, S., RUIZ-ROBLES, A. y MOREIRO-GONZÁLEZ, J.-A. *Visualización y recuperación de información en la web semántica*. *Profesional de la Información*. 2014. Vol. 23, no. 3.

MOTT, Richard. *Smith-Waterman Algorithm*. In: *Encyclopedia of Life Sciences* [en línea]. 2005. [Consultado el: 14 diciembre 2017] ISBN 047001590X. Disponible en: http://doi.wiley.com/10.1038/npg.els.0005263

NEEDLEMAN, Saul B. y WUNSCH, Christian D. *A general method applicable to the search for similarities in the amino acid sequence of two proteins*. *Journal of Molecular Biology*. 1970. Vol. 48, no. 3, p. 443-453.

NELGAR, H. A. S. *Un modelo para la visualización de conocimiento basado en imágenes semánticas*. *Universidad Federal de Santa Catarina*, 2011.

NOY, Natalya F. y MCGUINNESS, Deborah L. *Ontology Development 101: A Guide to Creating Your First Ontology*. *Stanford Knowledge Systems Laboratory*. 2001, p. 25.

OCLC. [en línea]. 2015. [Consultado el: 1 diciembre 2017]. Disponible en: http://www.oclc.org/research/activities/scorpion.html

PASTOR SÁNCHEZ, Juan Antonio y MARTÍNEZ MÉNDEZ, Francisco Javier. *Aplicación de tesauros, taxonomías y ontologías en los sistemas de gestión de contenidos mediante tecnologías de la Web Semántica*. *Ibersid*. 2009. P. 143-153.

PAVONE, P. *Un Método de Text Mining para la categorización Fuzzy de documentos*. 2015.

POLLOCK, Joseph J. y ZAMORA, Antonio. *Automatic spelling correction in scientific and scholarly text*. *Communications of the ACM*. 1984. Vol. 27, no. 4, p. 358-368.

PRESMAN, Roger S. 5ta edición. [en línea]. 2006. [Consultado el: 25 enero 2018]. Disponible en: http://bibliodoc.uci.cu/pdf/reg02689.pdf

PRESSMAN, Roger S. *Ingeniería del software. Un enfoque práctico. 7ma Edición*. 2010.

INTERNET LIVE STATS. *Real Time Statistics*. [en línea]. 2017. [Consultado el: 12 octubre 2017] Disponible en: http://www.internetlivestats.com/

PURI, S, VECTOR, *Support y CLASSIFIER, Machine. A Fuzzy Similarity Based Concept Mining Model for Text Classification*. *International Journal of Advanced Computer Science and Applications*. 2011. Vol. 2, no. 11, p. 115-121.

RADA, Roy, MILI, Hafedh, BICKNELL, Ellen y BLETTNER, Maria. *Development and Application of a Metric on Semantic Nets*. *IEEE Transactions on Systems, Man and Cybernetics*. 1989. Vol. 19, no. 1, p. 17-30.

RAJU, Miji K., SUBRAHMANIAN, Sneha T. y SIVAKUMAR, T. *A Comparative Survey on Different Text Categorization Techniques. Journal of Computer Science and Engineering*. 2017. Vol. 5, p. 1612-1618.

RAMÍREZ BUSTAMANTE, Flora y LÓPEZ DÍAZ, Enrique. *Spelling error patterns in Spanish for word processing applications*. In: *LREC 2006. Proceedings of the 5th International Conference on Language Resources and Evaluation*. 2006, p. 93-98.

SAMPIERI, H. *Medición y operacionalización. Variables. Indicadores* [en línea]. 2010. [Consultado el: 22 marzo 2018] Disponible en: https://es.slideshare.net/Eulaliaperalta/operacionalizacin-variables-sampieri.

SÁNCHEZ, S*. Estrategia de soporte técnico para el proceso de migración a código abierto en los Organismos de la Administración Central del Estado*. *Universidad de las Ciencias Informáticas*, 2015.

SCRIBD INC. *Arquitectura basada en Componentes* [en línea]. 2016. [Consultado el: 23 enero 2018]. Disponible en: http://www.scribd.com/doc/14704374/Arquitectura-Basada-en-Componentes/

SEARCH TECHNOLOGIES. *Arquitectura Basada en Componentes* [en línea]. 2017. [Consultado el: 10 diciembre 2017]. Disponible en: http://www.scribd.com/doc/14704374/Arquitectura-Basada-en-Componentes

SELVI, S. Thamarai, KARTHIKEYAN, P., VINCENT, A., ABINAYA, V., NEERAJA, G. y DEEPIKA, R. *Text categorization using Rocchio algorithm and random forest algorithm*. In: *2016 8th International Conference on Advanced Computing, ICoAC 2016*. 2017. p. 7-12. ISBN 9781509058884.

SIDOROV, Grigori, GELBUKH, Alexander, GÓMEZ-ADORNO, Helena y PINTO, David*. Soft similarity and soft cosine measure: Similarity of features in vector space model. Computación y Sistemas*. 2014. Vol. 18, no. 3, p. 491-504.

SMITH, T. F. y WATERMAN, M. S. *Identification of common molecular subsequences*. *Journal of Molecular Biology*. 1981. Vol. 147, no. 1, p. 195-197.

SOMMERVILLE, Ian. *Ingeniería del software* [en línea]. 2005. [Consultado el: 17 enero 2018] ISBN 8478290745. Disponible en: http://danielr.obolog.es/ingenieria-software-355416%5Cnhttp://fondoeditorial.uneg.edu.ve/citeg/numeros/c02/c02_art10.pdf

SPARXSYSTEMS. *Diagrama de despliegue UML 2* [en línea]. 2014. [Consultado el: 25 enero 2018]. Disponible en: http://www.sparxsystems.com.ar/resources/tutorial/uml2_deploymentdiagram.html

STEVE, Geri, GANGEMI, Aldo y PISANELLI, Domenico M. *Integrating medical terminologies with ONIONS methodology*. 1997.

STUDER, Rudi, BENJAMINS, V.Richard y FENSEL, Dieter. *Knowledge engineering: Principles and methods*. *Data & Knowledge Engineering*. 1998. Vol. 25, no. 1-2, p. 161-197.

TARRAGO, D. S. *Algoritmos para la Clasificación Multinstancia*. Universidad de Granada, 2014.

THE APACHE SOFTWARE FOUNDATION. Apache Jena Fuseki. *Apache Jena* [en línea]. 2011. [Consultado el: 20 octubre 2018]. Disponible en: http://jena.apache.org.

TORRES, Carmen y ARCO, Leticia. *Representación textual en espacios vectoriales semánticos*. *Revista Cubana de Ciencias Informáticas*. 2016. Vol. 10, no. 2, p. 148-180.

UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS. *Metodología de desarrollo para la Actividad productiva de la UCI*. 2015.

VALERO M., Ana Isabel. *Técnicas estadísticas en minería de textos*. 2017.

VIKRAMKUMAR, B, Vijaykumar y TRILOCHAN. *Bayes and Naive Bayes Classifier*. 2014.

VILTRES, H. S., LEYVA, P. R., FEBLES, J. P. y SENTÍ, V. E. *Procesamiento Semántico de información en Sistemas de Recuperación de Información*. *Revista Cubana de Ciencias Informáticas*. 2018. Vol. 12, p. 102-116.

VISUAL PARADIGM. *Software design tools for agile software development* [en línea]. 2014. [Consultado el: 10 diciembre 2017]. Disponible en: http://www.visual-paradigm.com/product/vpuml

WADHAWAN, Rimpy y MITTAL, Saurabh. *Improved Nearest Neighbour Approach for Document Categorization*. 2017.

WILLIAM E. YANCEY. *Evaluating string comparator performance for record linkage*. *Statistical Research Division*. 2005. P. 3905-3912.

WINKLER, William E. *Frequency-based matching in Fellegi-Sunter model of record linkage*. *Bureau of the Census Statistical Research Division*. 2000. Vol. 14.

Anexos

Tabla 25: Plan de preguntas. Entrevista a los principales especialistas del proyecto.

No.	Preguntas
1	¿Qué estructura tiene la plataforma c·u·b·a.?
2	¿Cómo almacena los documentos?
3	¿Cómo categoriza esta documentación?
4	¿Qué herramientas tecnológicas se utilizan en su implementación?
5	¿Qué metodología de desarrollo empleó el proyecto?