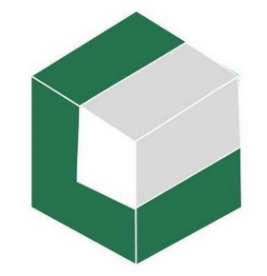
******

***Universidad Tecnológica de La Habana “José Antonio Echeverría”***

***Facultad de Ingeniería Informática***

***Trabajo de Diploma***

***Extracción automática de requisitos de software a partir de información textual no estructurada***

***Autor: Amanda Hernández Carreras (ahernandezrr@ceis.cujae.edu.cu)***

***Tutores: Dr. C Anaisa Hernández González (anaisa@ceis.edu.cu)***

***Dr. C Alfredo Simón Cuevas (asimon@ceis.cujae.edu.cu)***

***La Habana, Cuba 2022***

**Resumen**

La obtención de requisitos es una de las fases más importantes y críticas en el desarrollo de software, debido a la influencia de sus resultados en el éxito de los proyectos. El análisis documental constituye una de las técnicas más utilizadas en este proceso. La ejecución manual de este análisis se ha caracterizado por el alto consumo de tiempo y la frecuente aparición de errores, motivando el desarrollo de investigaciones enfocadas en su automatización. El procesamiento del lenguaje natural para la ingeniería de requisitos (NLP4RE) es un área de investigación y desarrollo que busca aplicar técnicas, herramientas y recursos de procesamiento del lenguaje natural (NLP) al proceso de ingeniería de requisitos (RE), para ayudar a los analistas humanos a llevar a cabo diversas tareas lingüísticas. En el trabajo se presentó un método para la extracción automática de requisitos de software, a partir de información textual no estructurada. El método propuesto se enfoca en el análisis sintáctico basado en patrones léxicos – sintácticos, en el análisis sintáctico basado en análisis de dependencias y un enfoque basado en la combinación de ambas vías de educción. Las métricas de Precisión y Cobertura fueron computadas comparando el requisito que se obtuvo, con el elaborado manualmente por el experto. En esa comparación se empleó la distancia Levenshtein, usando como umbral de aceptación el 60%. Los resultados obtenidos fueron relativamente altos, resaltando el valor de la cobertura en los 3 experimentos y en general los valores del método de extracción basado en enfoque híbrido.

**Abstract**

Requirements elicitation is one of the most important and critical phases in software development, due to the influence of its results on the success of projects. Documentary analysis is one of the most used techniques in this process. The manual execution of this analysis has been characterized by the high consumption of time and the frequent appearance of errors, motivating the development of research focused on its automation. Natural Language Processing for Requirements Engineering (NLP4RE) is an area of ​​research and development that seeks to apply natural language processing (NLP) techniques, tools, and resources to the requirements engineering (RE) process, to help human analysts to carry out various linguistic tasks. In the work, a method for the automatic extraction of software requirements, from unstructured textual information, was presented. The proposed method focuses on syntactic analysis based on lexical-syntactic patterns, on syntactic analysis based on dependency analysis and an approach based on the combination of both ways of eduction. The Accuracy and Coverage metrics were computed by comparing the requirement that was obtained, with the one elaborated manually by the expert. In this comparison, the Levenshtein distance was used, using 60% as the acceptance threshold. The results obtained were relatively high, highlighting the value of the coverage in the 3 experiments and, in general, the values ​​of the extraction method based on the hybrid approach.

Índice

[Introducción 1](#_Toc117810183)

[Capítulo 1. Fundamentos teóricos 5](#_Toc117810184)

[1.1 Ingeniería de requisitos 5](#_Toc117810185)

[1.1.1 Estándares y formalización de requisitos de software 6](#_Toc117810186)

[1.1.2 Gestión de requisitos con soporte computacional 9](#_Toc117810187)

[1.1.3 Rol del lenguaje natural en los requisitos de software 10](#_Toc117810188)

[1.1.3.1 Construcción de un requisito en lenguaje natural. 11](#_Toc117810189)

[1.1.3.2 Procesamiento de lenguaje natural 12](#_Toc117810190)

[1.2 Procesamiento de lenguaje natural aplicado a la Ingeniería de Requisitos 16](#_Toc117810191)

[1.2.1 Análisis de trabajos relacionados 17](#_Toc117810192)

[1.3 Métodos de evaluación 20](#_Toc117810193)

[1.3.1 Métodos de evaluación reportados en la literatura 20](#_Toc117810194)

[1.3.2 Métricas de análisis de resultados 20](#_Toc117810195)

[1.4 Análisis de tecnologías de implementación 21](#_Toc117810196)

[1.3.1 Python 21](#_Toc117810197)

[1.3.2 PyCharm (IDE) 22](#_Toc117810198)

[1.3.3 SpaCy 24](#_Toc117810199)

[Conclusiones parciales 26](#_Toc117810200)

[Capítulo 2. Extracción automática de requisitos de software 27](#_Toc117810201)

[2.1 Solución teórica 27](#_Toc117810202)

[2.1.1 Pre – procesamiento 28](#_Toc117810203)

[2.1.2 Extracción de requisitos candidatos 29](#_Toc117810204)

[2.1.2.1 Análisis sintáctico basado en patrones léxicos – sintácticos 29](#_Toc117810205)

[2.1.2.2 Análisis sintáctico basado análisis de dependencias 31](#_Toc117810206)

[2.1.3 Reducción de redundancias 32](#_Toc117810207)

[2.2 Desarrollo de la solución de extracción de requisitos de software 33](#_Toc117810208)

[2.2.1 Diagrama de casos de uso y descripción de casos de uso 34](#_Toc117810209)

[2.2.1.1 Actores del sistema 34](#_Toc117810210)

[2.2.1.2 Especificación de alto nivel de caso de uso 35](#_Toc117810211)

[2.2.2 Arquitectura del modelo 35](#_Toc117810212)

[2.2.3 Estilos y patrones de arquitectura 36](#_Toc117810213)

[Conclusiones parciales 37](#_Toc117810214)

[Capítulo 3. Evaluación de la solución propuesta 38](#_Toc117810215)

[3.1 Marco de evaluación 38](#_Toc117810216)

[3.1.1 Ejemplo de extracción automática 39](#_Toc117810217)

[3.2 Resultados y discusión 41](#_Toc117810218)

[Conclusiones 43](#_Toc117810219)

[Recomendaciones 44](#_Toc117810220)

[Referencias Bibliográficas 45](#_Toc117810221)

# Índice de Tablas

[Tabla 1 Resumen de revisión bibliográfica 19](#_Toc117805882)

[Tabla 2 Diferencias entre IDE Python 23](#_Toc117805883)

[Tabla 3 Comparación entre herramientas NLP 25](#_Toc117805884)

[Tabla 4 Patrones léxicos - sintácticos 30](#_Toc117805885)

[Tabla 5 Ejemplo de extracción usando análisis de dependencias 32](#_Toc117805886)

[Tabla 6 Actores del sistema y una descripción 34](#_Toc117805887)

[Tabla 7 Descripción del caso de uso Extraer requisitos de software. 35](#_Toc117805888)

[Tabla 8 Descripción de casos de estudio. 38](#_Toc117805889)

[Tabla 9 Requisitos extraídos usando patrones léxicos sintácticos 40](#_Toc117805890)

[Tabla 10 Requisitos extraídos usando análisis de dependencias 40](#_Toc117805891)

[Tabla 11 Resultados de la evaluación del método con documentos de entrevistas a usuarios. 41](#_Toc117805892)

# Índice de Ilustraciones

[Ilustración 1 Proceso de construcción de un requisito 11](#_Toc117265367)

[Ilustración 2 Estructura de un requisito en lenguaje natural 12](#_Toc117265368)

[Ilustración 3 Flujo de trabajo de la solución propuesta 27](#_Toc117265369)

[Ilustración 4 Diagrama de actividades del pre - procesamiento 29](#_Toc117265370)

[Ilustración 5 Árbol de dependencias generado por el analizador sintáctico SpaCy 32](#_Toc117265371)

[Ilustración 6 Diagrama de caso de uso 34](#_Toc117265372)

[Ilustración 7 Estructura en capas del modelo. 36](#_Toc117265373)

[Ilustración 8 Estructura modular del sistema. Arquitectura basada en componentes. 37](#_Toc117265374)

# Introducción

La industria del software presenta índices de éxito relativamente bajos en los proyectos, si se tienen en cuenta estudios e investigaciones realizados por Standish Group’s en los reportes Chaos, los cuales analizan el comportamiento de proyectos desde el año 1994. Según estos estudios aproximadamente el 50% de los expertos encuestados considera que el comportamiento de fallo en los proyectos es idéntico al de hace 10 años, el 48% considera que existen más fallos en los proyectos hoy en día y solamente el 2% considera que ha existido una mejoría en este aspecto. Los datos estadísticos reflejan que en el año 2018 solo el 16.2% de los proyectos de software fueron completados cumpliendo con el tiempo, el presupuesto y con al menos la mitad de los requisitos planteados inicialmente. Estos datos son incluso peores teniendo en cuenta los proyectos llevados a cabo por grandes empresas donde pueden llegar al 9% [1, 2].

Las principales causas de fallo de un proyecto han sido descritas en muchos estudios entre las principales se encuentran:

* Incompleta o errónea declaración de requisitos.
* Requisitos cambiantes.
* Expectativas poco realistas.
* Poca claridad en los objetivos y metas.
* Incompetencia de la tecnología.
* Metas de tiempo poco realistas.
* Calidad del producto.
* Poca claridad de los procesos de negocio.
* Satisfacción del cliente.
* Poca comunicación entre clientes, desarrolladores y usuarios finales.
* Políticas de los stakeholders (partes interesadas).

En el proceso de captura de requisitos de software se ha demostrado que el porcentaje de fallos sigue siendo considerablemente alto y el costo de corregir un error a partir de un mal proceso de ingeniería de requisitos (RE) puede suponer hasta 5 veces superior, en comparación a otros errores, aspectos que motivan y fundamentan la necesidad de continuar con el estudio de esta rama con el objetivo de disminuir este porcentaje de fallos en esta temprana etapa del proyecto. [3, 4].

Los requisitos provienen de varias partes interesadas que tienen diferentes necesidades, funciones y responsabilidades y, como tales, son propensas a que se produzcan conflictos, como la interferencia, la interdependencia y la incoherencia [5]. Además, los requisitos generalmente se especifican en lenguajes naturales, lo que aumenta la complejidad de la ingeniería de requisitos debido a la ambigüedad inherente, la incompletitud y la inexactitud de los lenguajes naturales [6]. Estos factores hacen que las tareas de RE sean desafiantes, lentas y propensas a errores, principalmente para proyectos grandes, ya que es necesario procesar, analizar y comprender grandes volúmenes de requisitos [7].

Se han llevado a cabo muchas investigaciones sobre la automatización de diferentes tareas de RE. Los enfoques propuestos generalmente comienzan aplicando un conjunto de pasos de Procesamiento del Lenguaje Natural (NLP) que extraen información y características lingüísticas de los textos de requisitos y construyen varias representaciones basadas en NLP.

Inspirándose en la estrecha relación entre el lenguaje natural (NL) y los requisitos, desde principios de la década de 1980, los investigadores han intentado desarrollar herramientas y métodos de NLP para procesar los textos de requisitos [8]. Sin embargo, durante casi tres décadas, esta investigación se vio obstaculizada por tecnologías de NLP inadecuadas, que no funcionaron lo suficientemente bien en ese momento para respaldar aplicaciones fuera de la investigación de la NLP [9]. La situación ha ido cambiando radicalmente desde finales del año 2000. Enormes mejoras y avances en las tecnologías de NLP, en particular la amplia disponibilidad de herramientas y recursos de NLP [10], hicieron posible que los investigadores exploraran una variedad de herramientas y recursos de NLP, habilitando herramientas y métodos para tareas de ER [11] [12]. Desde entonces, la investigación en NLP para ER, o NLP4RE para abreviar, ha crecido en un área de investigación activa y completa [11], que atrae a investigadores de la comunidad de RE en general. Hoy en día, la investigación en NLP4RE está prosperando y tiene un taller anual dedicado, llamado NLP4RE [12]. Ahora existe un gran potencial para desarrollar herramientas NLP4RE efectivas que puedan servir al mundo real para la práctica de RE. Esto es particularmente necesario, ya que las tareas de RE todavía se realizan manualmente en la práctica industrial, como lo demuestran los datos de la encuesta de “Naming the Pain in Requirements Engineering”(NaPiRE) que muestran que solo el 16% de las empresas utilizan técnicas automatizadas para el análisis de requisitos [13], pero la mayoría de estos trabajos son enfocados al procesamiento de texto en idioma inglés.

Por lo tanto, nuestra **situación problemática** es: La elicitación de requisitos es un trabajo mayoritariamente manual que tiene sus bases en la experiencia y capacidad de especialistas en llevar a cabo las diferentes técnicas, metodologías y métodos para el análisis de la información y su transformación en requisitos de software.

Como **objeto de estudio** para dar solución a este problema se tiene: Información textual no estructurada en idioma español.

Como **campo de acción** se tiene: Procesamiento de lenguaje Natural aplicado a la Ingeniería de Requisitos.

Se define como **objetivo general** Desarrollar un sistema para la extracción automática de requisitos de software, a partir de información textual no estructurada.

**Objetivos Específicos**

1. Caracterizar estándares y modelos de formalización de requisitos reportados en la literatura.

* Estudiar documentación sobre estándares y modelos de formalización de requisitos de software reportados en la literatura.
* Caracterizar estándares y modelos de formalización de requisitos, en cuanto a: elementos de información que intervienen, especificaciones, artefactos, elementos comunes, entre otros elementos.

1. Caracterizar trabajos reportados en la literatura.

* Caracterizar trabajos reportados en la literatura afines con la captura de requisitos de software a partir de información textual no estructurada.
* Identificar y caracterizar fuentes de información que será suministrada al modelo para la generación de requisitos.
* Determinar especificaciones de salidas del modelo a diseñar.
* Identificar recursos (bases de conocimiento u otros) que pueden aportar información/conocimiento al modelo

1. Implementar sistema para la extracción automática de requisitos de software.

* Implementar método basado en el análisis sintáctico a partir de patrones léxico – sintácticos.
* Extender implementación de los patrones léxicos-sintácticos.
* Implementar solución basa en el análisis de dependencias.
* Concebir mejoras al proceso de reducción de redundancias de requisitos capturados.
* Implementar mejoras concebidas al proceso de reducción de redundancias.

1. Evaluar la nueva versión de solución de capturas de requisitos.

* Construir colecciones de prueba para la experimentación y medición de resultados.
* Ejecutar una fase de ajuste del marco de evaluación diseñado.
* Refinar modelo e implementación según resultados preliminares de la evaluación realizada.
* Ejecutar marco de evaluación diseñado.
* Evaluar la nueva versión de solución de capturas de requisitos a partir de textos identificando debilidades y definiendo soluciones.

El resto del documento está organizado de la siguiente manera: Capítulo 1 Fundamentos teóricos. Capítulo 2 Extracción automática de requisitos de software. Capítulo 3 Evaluación de la solución propuesta.

# Capítulo 1. Fundamentos teóricos

En este capítulo se presentan las bases teóricas del trabajo, los aspectos necesarios para lograr la comprensión de conceptos que permitan el cumplimiento de los objetivos trazados. Se abordan los conceptos relativos a la Ingeniería de requisitos (RE) y el Procesamiento del lenguaje natural (NLP), así como la aplicación de técnicas, recursos y herramientas de NLP a la RE.

# Ingeniería de requisitos

Los requerimientos para un sistema son descripciones de lo que el sistema debe hacer: el servicio que ofrece y las restricciones en su operación. Tales requerimientos reflejan las necesidades de los clientes por un sistema que atienda cierto propósito, como sería controlar un dispositivo, colocar un pedido o buscar información. Al proceso de descubrir, analizar, documentar y verificar estos servicios y restricciones se le llama ingeniería de requisitos (RE) [14].

Por su parte [15] plantea que un requisito es una condición o capacidad necesitada por un usuario para resolver un problema o lograr un objetivo; una condición o capacidad que debe ser cumplida o procesada por un sistema o componente del sistema para satisfacer un contrato, estándar, especificación u otro documento formalmente impuesto; una representación documentada de una condición o capacidad como en las dos ideas previas expuestas.

Según [15] la Ingeniería de Requisitos es un enfoque disciplinado y sistemático a la especificación y gestión de requerimientos con las siguientes metas: conocer los requerimientos relevantes, logrando un conceso sobre estos entre los interesados, documentarlos de acuerdo a estándares establecidos y gestionarlos sistemáticamente; comprender y documentar los deseos y necesidades de los interesados, tenerlos a ellos especificando y gestionando los requerimientos minimizará el riesgo de entregar un sistema que no cumpla con sus deseos o necesidades.

Otra definición, planteada por [16] es que el espectro amplio de tareas y técnicas que llevan a entender los requerimientos se denomina ingeniería de requerimientos. Desde la perspectiva del proceso del software, la ingeniería de requerimientos es una de las acciones importantes de la ingeniería de software que comienza durante la actividad de comunicación y continúa en la de modelado. Debe adaptarse a las necesidades del proceso, del proyecto, del producto y de las personas que hacen el trabajo.

## Estándares y formalización de requisitos de software

La documentación de requisitos de software se puede estructurar en tres grupos fundamentales:

1. Utilizando lenguaje natural: Esta es la variante más comúnmente utilizada en la práctica, teniendo la ventaja de que los stakeholders no deberán aprender nuevas notaciones y que el lenguaje puede utilizarse de múltiples propósitos, desde estructuras formales hasta descripciones detalladas. En cambio, los requisitos en lenguaje natural pueden ser ambiguos, y perder perspectiva si no se redactan correctamente.
2. Modelos conceptuales: A diferencia del lenguaje natural los diferentes modelos conceptuales no pueden utilizarse de forma universal. Al documentar requisitos mediante modelos se deben utilizar lenguajes de modelado específicos, que deben ser utilizados de forma correcta para poder alcanzar el objetivo deseado. Entre los modelos más utilizados se encuentran:

* Diagrama de clases
* Diagramas de casos de uso
* Diagrama de actividades
* Diagrama de estados

1. Documentos híbridos: Actualmente es la variante que más fuerza va tomando. La utilización combinada de los dos puntos anteriores permite elaborar un documento enfocado a la audiencia objetivo, brindando diferentes perspectivas del sistema. La utilización combinada del lenguaje natural y modelos conceptuales arrastra las desventajas de ambas opciones, pero se complementa por la combinación de las fortalezas y su versatilidad.

Para determinar que estructuras de requisitos utilizar, los estereotipos adecuados en cada situación definen diferentes estándares desde los métodos tradicionales hasta las más actuales metodologías ágiles de desarrollo. Entre los estándares más utilizados se encuentran:

1. ***Rational Unified Process (RUP)***[17]*:* Este proceso es usualmente utilizado al desarrollar bajo el paradigma orientado a objeto. En este se utilizan modelos para describir el negocio y la solución mediante el uso de diferentes artefactos como: Reglas del negocio, casos de uso, combinando el lenguaje natural con modelos conceptuales. La especificación de requisitos normalmente es redactada utilizando la estructura *software requirements specification (SRS),* la cual está muy relacionada al estándar ISO/IEC/IEEE 29148:2011.

Entre los modelos que propone RUP se encuentran:

* Modelos de casos de uso
* Modelo de análisis
* Modelo de diseño
* Modelo de despliegue
* Modelo de implementación
* Modelo de prueba

1. **Estándar ISO/IEC/IEEE 29148:2011** [18]: Contiene normas establecidas para la documentación de requisitos de software. El estándar sugiere dividir el documento de especificación de requisitos en 5 capítulos utilizando el lenguaje natural como variante a utilizar.

* Información introductoria
* Referencias
* Requisitos del sistema (funcionales, rendimiento, interfaces)
* Medidas para la verificación
* Apéndice (Se aclaran los elementos que se asumen, identifican las dependencias)

1. **Modelo V**: Define diferentes estructuras a partir de quien es el creador:

* Especificación de requisitos del cliente: Elaborado por el cliente, incluye servicios, restricciones, explicación del proceso. Generalmente se escribe para responder el qué se hace y para qué se hace.
* Especificación de requisitos del sistema: Se basa en la especificación del cliente, pero tiene detalles del diseño e implementación que elabora el contratista. Se elabora a partir del proceso y restricciones establecidas por el cliente.

1. **CMMi** [19]: Este estándar plantea como propósito del desarrollo de requisitos de software, producir y analizar los requerimientos de cliente, de producto y de componente del producto.
2. **Estándar IEEE 830**: El estándar IEEE 830-1998 está enfocado en recomendaciones prácticas para la especificación de requerimientos, fue desarrollado por la IEEE y la IEEE-SA (Standards Association), indica la estructura y organización de toda la información que debe incluirse en un buen documento de especificación de requerimientos de software. Una de las características más distinguibles del estándar IEEE 830 es que los requisitos deben ser escritos de la siguiente manera: “El sistema debe…”.
3. Metodologías ágiles e historias de usuario: Las Historias de Usuario son un enfoque de requerimientos ágil que se focaliza en establecer conversaciones acerca de las necesidades de los clientes. Son descripciones cortas y simples de las funcionalidades del sistema, narradas desde la perspectiva de la persona que desea dicha funcionalidad, usualmente un usuario.

## Gestión de requisitos con soporte computacional

Ingenieros e investigadores de la ciencia de datos utilizan el aprendizaje automático para obtener nuevas perspectivas y soluciones de diversos desafíos, que estaban hasta este momento fuera de su alcance, aprovechando la infraestructura de hardware y disponibilidad de grandes volúmenes de datos. Algunas de las actividades que están siendo respaldadas por las tecnologías de aprendizaje automático son las relacionadas al desarrollo y gestión de requisitos.

En cuanto a las actividades de educción de requisitos, en [20] se propone un método para identificar automáticamente los requisitos de seguridad, combinando el análisis lingüístico con técnicas de aprendizaje automático.

En [21], proponen un framework de apoyo a la toma de decisiones, que emplea procesamiento de lenguaje natural y una ontología para procesar especificaciones de requisitos a modo de facilitar y brindar soporte a ingenieros y gerentes de proyectos.

Otras propuestas abordan los problemas de priorización de requisitos. En [22], presentan un método, denominado SNIPR, para ordenar y seleccionar los requerimientos de acuerdo a las prioridades de los interesados, mediante el empleo de técnicas de procesamiento del lenguaje natural.

En [23], proponen el uso de técnicas de minería de datos y aprendizaje automático para priorizar los requisitos de acuerdo con los intereses de las partes interesadas, los objetivos empresariales y las preocupaciones transversales, tales como los requisitos de seguridad y rendimiento. La eficacia del enfoque se evalúa a través de un caso de estudio basado en un conjunto de requisitos extraídos de un proyecto de código abierto denominado SugarCRM.

En [24], se aplica el aprendizaje automático para predecir el rendimiento de ciertas configuraciones del sistema que puedan afectar a los requerimientos. En [25], proponen una herramienta capaz de predecir los requerimientos que pueden fallar, dada la ambigüedad y escasez de información en la especificación de los requisitos. En [26], proponen el uso de las redes Bayesianas para predecir cuales son las especificaciones de requisitos que deben ser revisadas.

Otras propuestas resuelven los desafíos de la RE mediante las técnicas de clasificación. En [20], proponen un método para detectar ambigüedad en los documentos de especificación de requisitos, mediante el uso de redes bayesianas. En [27], proponen el uso de clasificadores bayesianos para clasificar los riesgos en los requerimientos. En [28], proponen clasificar automáticamente los requisitos de software en funcionales y no funcionales, con técnicas de aprendizaje automático supervisado. Paralelamente, se evalúa la precisión con la que se puede identificar varios tipos de requerimientos no funcionales, en particular los requisitos de usabilidad, seguridad, operación y rendimiento. En [29], consideran la aplicación de aprendizaje automático guiado por Tensorflow usando Word2vec para resolver el problema de clasificación de requerimientos.

Las técnicas de aprendizaje automático también son aplicadas para dar solución a los problemas de trazabilidad [30].

## Rol del lenguaje natural en los requisitos de software

El importante papel del lenguaje natural (NL) en la ingeniería de requisitos (RE) se ha establecido hace mucho tiempo [31] [32]. En una encuesta publicada en 1981, Abbott y Moorhead afirmaron que “el mejor lenguaje para los requisitos es el lenguaje natural” [33]. Si bien es difícil demostrar que NL es en realidad la mejor opción, la evidencia empírica a lo largo de los años ha demostrado que es al menos la notación más común para expresar requisitos en la práctica industrial. La encuesta en línea de 151 empresas de software a principios de la década de 2000 realizada por Mich et al. [34] concluyó que en el 95% por ciento de los casos los requisitos de los documentos se expresaron en alguna forma de NL. Este predominio de NL fue confirmado por un reciente encuesta de Kassab et al. [35], que involucró a 250 practicantes. La mayoría de los participantes (61%) en esa encuesta indicaron que NL se usaba normalmente en sus empresas para describir y especificar requisitos del software y del sistema. Por lo tanto, con base en la evidencia empírica pasada y actual, se puede suponer con seguridad que NL continuará sirviendo como la lengua franca para los requisitos en el futuro también. Inspirándose en la estrecha relación entre el NL y los requisitos, desde principios de la década de 1980, los investigadores han intentado desarrollar herramientas y métodos de procesamiento del lenguaje natural (NLP) para procesar los textos de requisitos [32].

### Construcción de un requisito en lenguaje natural.

Para la construcción de un requisito en lenguaje natural se pueden plantear 5 pasos fundamentales, con el objetivo de garantizar la calidad de los requisitos mediante el uso de una estructura definida y glosarios de términos [16].

Ilustración 1 Proceso de construcción de un requisito

Se obtiene la siguiente estructura como base para la construcción del requisito en lenguaje natural:

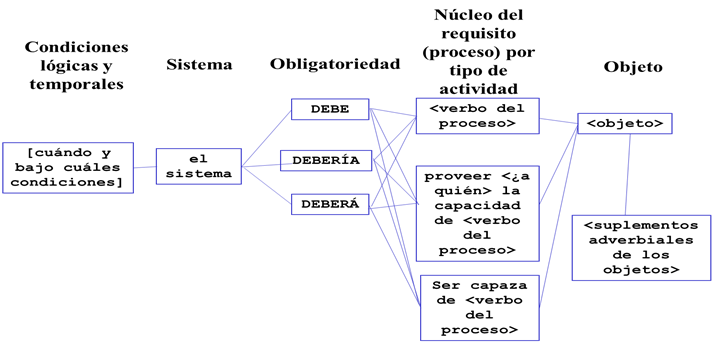


Ilustración 2 Estructura de un requisito en lenguaje natural

### Procesamiento de lenguaje natural

El procesamiento del lenguaje natural es una de las principales disciplinas de la inteligencia artificial. Su objetivo es permitir que los programas informáticos "entiendan" y procesen textos en lenguaje natural para lograr algunos objetivos específicos. Tradicionalmente, existen tres niveles principales de procesamiento en un enfoque basado en NLP [36]:

* Nivel léxico y morfológico: El nivel morfológico se enfoca en analizar palabras en sus morfemas como prefijos, sufijos, y palabras base. Incluye tareas comunes, como tokenización (proceso de dividir un texto en una lista de tokens) y lematización (proceso de encontrar la forma de diccionario, o el lema de cada palabra).
* Nivel sintáctico: El nivel sintáctico se enfoca en analizar la estructura gramatical de las oraciones. Este nivel generalmente incluye etiquetado de parte del discurso (etiquetado de POS(Part-of-speech): proceso de etiquetar cada token en una oración con su parte correspondiente de una etiqueta de voz basada en su contexto sintáctico), fragmentación (proceso de detección de constituyentes sintácticos como frases nominales y frases verbales en una oración), análisis de dependencia (proceso de analizar la estructura sintáctica de la oración, descubriendo las palabras relacionadas gramaticalmente, así como el tipo de relación entre ellas) y reconocimiento de entidad nombrada (localizar y clasificar las entidades nombradas mencionadas en la oración, en categorías predefinidas, como nombres de personas, organizaciones, ubicaciones)
* Nivel semántico: El nivel semántico se enfoca en comprender el significado del texto. El objetivo principal del procesamiento semántico es convertir automáticamente una oración del lenguaje natural en una representación formal de su significado, por ejemplo, la representación basada en ontologías (modelo de datos que representa un conjunto de conceptos dentro de un dominio y las relaciones entre esos conceptos), el modelo de espacio vectorial (modelo de representación básico que representa el texto como una matriz de término por documento), la representación basada en modelos de temas: (modelado estadístico utilizado para descubrir los temas latentes o abstractos que ocurren en un conjunto de textos) y las técnicas avanzadas de incrustación (método eficiente para aprender representaciones vectoriales de palabras de alta calidad a partir de grandes cantidades de datos de texto no estructurados)

Tipos de tecnología NLP:

Técnica NLP: Una técnica de NLP es un método, enfoque, proceso o procedimiento práctico para realizar una tarea de NLP en particular, como tokenización, segmentación (chunking), etiquetado POS (Part-of-speech), análisis sintáctico superficial y de dependencias, reconocimiento de entidades, desambiguación, análisis semántico.

Herramienta NLP: Una herramienta de NLP es un sistema de software o una biblioteca de software que admite una o más técnicas de NLP, como:

* Stanford CoreNLP7[[1]](#footnote-1): ¡CoreNLP es su ventanilla única para el procesamiento del lenguaje natural en Java!, CoreNLP permite a los usuarios derivar anotaciones lingüísticas para el texto, incluidos los límites de oraciones y tokens, partes del discurso, entidades nombradas, valores numéricos y de tiempo, análisis de dependencias y constituyentes, correferencia, sentimiento, atribuciones de citas y relaciones. CoreNLP actualmente admite 6 idiomas: árabe, chino, inglés, francés, alemán y español. La pieza central de CoreNLP es la tubería. Las canalizaciones toman texto sin formato, ejecutan una serie de anotadores de NLP en el texto y producen un conjunto final de anotaciones. Las canalizaciones producen CoreDocuments, objetos de datos que contienen toda la información de anotaciones, accesibles con una API simple y serializables en un búfer de protocolo de Google.
* NLTK[[2]](#footnote-2): NLTK es una plataforma líder para crear programas Python que funcionen con datos de lenguaje humano. Proporciona interfaces fáciles de usar para más de 50 corpus y recursos léxicos como WordNet, junto con un conjunto de bibliotecas de procesamiento de texto para clasificación, tokenización, derivación, etiquetado, análisis y razonamiento semántico, envoltorios para bibliotecas de NLP de nivel industrial, y un foro de discusión activo. NLTK ha sido llamado "una herramienta maravillosa para enseñar y trabajar en lingüística computacional usando Python" y "una biblioteca increíble para jugar con el lenguaje natural". El procesamiento del lenguaje natural con Python proporciona una introducción práctica a la programación para el procesamiento del lenguaje. Escrito por los creadores de NLTK, guía al lector a través de los fundamentos para escribir programas Python, trabajar con corpus, categorizar texto, analizar la estructura lingüística y más. La versión en línea del libro se ha actualizado para Python 3 y NLTK 3.
* Pattern.es: es una biblioteca para Python que contiene herramientas para la conjugación de verbos, la singularización o la pluralización de sustantivos, la división de chunks y permite realizar el etiquetado POS para español. En la versión para el idioma inglés adiciona una interfaz para WordNet. Su instalación y uso son muy sencillos. Para el etiquetado gramatical utiliza Penn TreeBank Tags. Penn TreeBank está basado en el corpus Brown, pionero en etiquetado POS para inglés. A pesar de haberse creado para el inglés, Pattern lo usa también para el español. Cada etiqueta está formada con dos o tres caracteres que indican la función que cumple cada palabra en una oración.
* SpaCy: SpaCy es una biblioteca para el procesamiento avanzado de lenguaje natural en Python. Incluye modelos estadísticos pre-entrenados y vectores de palabras, admite tokenización para más de 45 idiomas. Provee etiquetado, análisis y reconocimiento de entidades nombradas y una fácil integración de aprendizaje profundo. Es muy sencilla de utilizar, basta con incorporar desde un programa Python el modelo para el español. Devuelve un etiquetado POS completo, no sólo indica la función de la palabra en la oración, sino otros datos tales como tiempo verbal, persona, número, modo, género, entre otros. La tokenizacion y el etiquetado gramatical se basan en el corpus OntoNotes57 que sigue la sintaxis de Penn TreeBank.
* Freeling[[3]](#footnote-3): Freeling es una librería de código abierto para el análisis de texto (tokenización, análisis morfológico, detección de entidades nombradas, etiquetado POS, etc.) para una variedad de idiomas. Desde su primera versión, tiene soporte para el español. Fue desarrollado por el Centro de Tecnologías y Aplicaciones del Lenguaje y el Habla (TALP) de la Universidad Politécnica de Catalunya. Se puede utilizar y ampliar los recursos lingüísticos por defecto (diccionarios, lexicones, gramáticas, etc.) adaptándolos a dominios específicos. Su código fue escrito en C++ bajo una arquitectura cliente – servidor. Existen diversas APIs que permiten utilizar Freeling en otros lenguajes, incluido Python. Su instalación no es sencilla, sobre todo en entornos Windows. Sin embargo, subsanado este inconveniente, al importar la API desde el programa Python se puede invocar a todas las funciones implementadas. El analizador morfológico de Freeling realiza el etiquetado POS usando las etiquetas EAGLES.

Recurso PNL: Un recurso de PNL es un recurso de datos lingüísticos para respaldar técnicas o herramientas de PNL, que puede ser un léxico del lenguaje (es decir, un diccionario) o un corpus (es decir, una colección de textos). Los léxicos existentes incluyen WordNet10 y FrameNet11, mientras que los ejemplos de corpus incluyen British National Corpus12 y Brown Corpus13.

# Procesamiento de lenguaje natural aplicado a la Ingeniería de Requisitos

El Procesamiento del lenguaje natural (NLP) se usa ampliamente para respaldar la automatización de diferentes tareas de ingeniería de requisitos (RE). La mayoría de los enfoques propuestos comienzan con varios pasos de NLP que analizan las declaraciones de requisitos, extraen su información lingüística y las convierten en representaciones fáciles de procesar, como listas de características o representaciones vectoriales basadas en incrustaciones. Estas representaciones basadas en NLP generalmente se usan en una etapa posterior como entradas para técnicas de aprendizaje automático o métodos basados en reglas. Por lo tanto, las representaciones de los requisitos juegan un papel importante en la determinación de la precisión de los diferentes enfoques.

Se han publicado muchas revisiones sobre la relación entre las tareas de NLP y RE, tales como la identificación de defectos de calidad y ambigüedad, clasificación y agrupación de grandes colecciones de requisitos, extracción de abstracciones clave, generación de modelos y trazabilidad entre los requisitos de lenguaje natural (NL) [37].

Hasta hace poco, muchas de estas aplicaciones de la NLP se han limitado al mundo académico debido a la inaccesibilidad de las herramientas de la NLP y la pronunciada curva de aprendizaje. Afortunadamente, los avances en el aprendizaje profundo y la disponibilidad de grandes corpus de NL han reducido significativamente las barreras de entrada al uso de NLP. Esto crea oportunidades sin precedentes para aplicar técnicas de NLP a la práctica de RE y ayudar a analizar automáticamente los documentos relacionados con los requisitos [37].

## Análisis de trabajos relacionados

Centrándose específicamente en la extracción automática de requisitos de software:

La solución reportada por [38] describe: Cargar el diseño del documento y analizarlo en oraciones. Dentro de las oraciones, también convertir las oraciones en palabras individuales y recuperar las etiquetas de parte del discurso para cada palabra. Se probaron dos enfoques para extraer requisitos de software, (1) extraer oraciones que contienen verbos en su forma base y (2) extraer frases que contienen palabras clave que se utilizan a menudo para indicar requisitos. Se verificó la ambigüedad semántica, si existe, se construye el árbol estructural de la oración para identificar el sujeto de la oración anterior: se desarrolló y definió una gramática y se analizó la oración en un árbol de acuerdo con esta gramática; el sujeto se identificó como el sintagma nominal principal en el árbol.

En [39] la metodología destinada es a mejorar la detección de los requisitos no funcionales (NFR) en los documentos de requisitos. Usa el Stanford Parser (equipado con el etiquetador POS de Brill y un lematizador morfológico) para derivar morfológicamente las palabras y extraer cinco características sintácticas de cada una de las instancias de entrenamiento (oraciones) del corpus.

En [40] el procedimiento que se desarrolla es el enfoque L'Ecritoire, con una relación bidireccional. Así como los objetivos pueden ayudar en el descubrimiento de escenarios, los escenarios pueden ayudar en el descubrimiento objetivo. La solución total está en dos partes, se crean escenarios textuales que es el que produce objetivo. La correspondencia entre un patrón semántico y el modelo de escenario define la relación entre la forma textual de un escenario y su forma conceptual.

En [41] una tarea importante para lograr este objetivo es construir una ontología que consista en un conjunto de conceptos, es decir, entidades, atributos y relaciones basadas en el dominio de aplicación de interés. La ontología construida aquí representa el conocimiento del dominio y los requisitos son el subconjunto especializado del mismo.

En este documento [42], se propone un enfoque semiautomático llamado NFR-Specifier, cuyo objetivo es generar especificaciones precisas a partir de requisitos informales, incluidos los NFR. El enfoque consta de cinco módulos, a saber. Pre-procesamiento, resolución de ambigüedades, formación de ontologías SRS, generación de diagramas UML y clasificación de NFR. Inicialmente, el ingeniero de requisitos recopila el conocimiento del dominio de los usuarios por medio de varios enfoques de comunicación, a saber, cuestionarios, entrevistas, checklist, prototipado, reuniones, entre otros. Una vez finalizada la fase de comunicación, el ingeniero de requisitos representa la información recopilada por medio de archivos de texto, documentos, gráficos o modelos UML (es decir, caso de uso, clase, diagrama de secuencia).

Este trabajo [43] explora cómo la cantidad y el tipo de conocimiento afectan la calidad de obtención de requisitos en dos simulaciones consecutivas. Se calcula con las salidas del automatismo con el patrón oro introducido en la última palabra. La recuperación puede verse como una medida de completitud, comparando el número de requisitos identificados con el número total de requisitos existentes en un documento.

El documento [44] propone un método para obtener los requisitos del usuario en la industria de maquinaria basado en la regla de asociación de texto. El primer paso es el pre-procesamiento de datos de los requisitos del usuario. El modelo de espacio vectorial se utiliza para describir los requisitos del usuario. En segundo lugar, se utiliza una teoría mejorada de la regla de asociación gris para calcular el grado de correlación entre las palabras características y los nombres propios de la industria de la maquinaria. Luego se construye la matriz de candidatos a nombres propios seleccionando una palabra de mayor grado de correlación. Finalmente, el requerimiento del usuario se obtiene utilizando la matriz ponderada.

El método de extracción propuesto por [45] consta de tres pasos organizados como una cadena de comandos en una canalización: (1) Análisis de dependencias, cada frase del texto de entrada se analiza mediante el analizador basado en dependencias DepPattern.(2) Constituyentes de la cláusula, para cada oración analizada, se descubren las cláusulas verbales que contiene y, luego, para cada cláusula, se identifican los participantes verbales, incluidas sus funciones: sujeto, objeto directo, atributo y complementos preposicionales. (3) Reglas de extracción, se aplica un conjunto de reglas sobre los componentes de la cláusula para extraer los triples objetivo.

Como conclusión de este análisis se identificaron varias limitaciones, tales como, la no existencia de disminución de redundancias entre las frases extraídas, no existe procesamiento para el idioma español y poco uso de la técnica de extracción de información basada en el análisis de dependencias.

Tabla 1 Resumen de revisión bibliográfica

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Trabajos relacionados*** | ***Tipo de requisito*** | ***Idioma a procesar*** | ***Herramientas PLN*** | ***Análisis sintáctico*** | | | ***Reducción de redundancias*** | | |
|  | | | | ***Análisis de dependencias*** | ***Patrones sintácticos*** | **Otros** | **Similitud de coseno** | **Algoritmo de agrupamiento** | **Otros** |
| [38] | funcional | ingles | NLTK | X |  |  | X |  |  |
| [39] | no funcional | ingles | Stanford |  |  | clasificación |  |  |  |
| [40] | funcional | ingles | NLTK |  | X |  |  |  |  |
| [41] | funcional | ingles |  |  | X |  |  |  |  |
| [42] | no funcional | ingles | Stanford |  | X |  |  |  | desambiguación |
| [43] | funcional | ingles | Stanford |  | X |  | X |  |  |
| [44] | funcional | ingles | Stanford |  | X |  |  |  |  |
| [45] | funcional | ingles |  | X |  |  |  |  |  |

# Métodos de evaluación

Las tareas de evaluación de la calidad se ocupan de detectar defectos en las especificaciones de los requisitos del software

## Métodos de evaluación reportados en la literatura

En el artículo [38] plantean como método para evaluar sus resultados:

Las oraciones extraídas, que se han identificado como requisitos probables, se envían a un archivo. Se crean vectores TF-IDF para cada una de las frases extraídas y cada uno de los requisitos en DOORS. Luego se calcula la similitud del coseno para cada posible par de vectores de oraciones y vectores de requisitos de DOORS. Si la similitud del coseno para cualquier par dado supera un umbral, aquí 0,95 en una escala de 0 a 1, se dice que este requisito es una coincidencia con una oración extraída. Se usa el vectorizador TF-IDF y la métrica de similitud de coseno por pares de scikitlearn (paquete de código abierto para herramientas de aprendizaje automático en Python).

## Métricas de análisis de resultados

Las métricas seleccionadas para la evaluación de la propuesta son precisión, cobertura y medida-F. La mayoría de los métodos reportados en el estado del arte hacen uso de estas métricas para evaluar sus resultados, esto permite comparar los resultados del método propuesto con los reportados por otras soluciones reportadas.

**Precisión (P)** permite evaluar con que precisión los requisitos extraídas se pueden tomar realmente como los adecuados. La precisión brinda la proporción de requisitos funcionales extraídos correctamente (*requisitos\_extraidos\_correctos*) del total de los requisitos extraídos, y se calcula como se muestra en la fórmula:

**Cobertura (C)** permite evaluar la medida en la que se cubren los requisitos extraídos automáticamente en comparación con los requisitos identificados manualmente (*requisitos\_correctos*) y se calcula como se muestra en la fórmula:

**Medida-F (F)** permite otorgarle una evaluación general a la propuesta a partir de las dos métricas definidas anteriormente. Un mayor valor de Medida-F significa un valor razonablemente mayor de la Precisión y la Cobertura, dado que se corresponde con la media harmónica de estas dos, y se calcula como se muestra en la fórmula:

# Análisis de tecnologías de implementación

La implementación de la solución se desarrolla en el lenguaje de programación Python el cual es un lenguaje de programación multiparadigma que soporta varios paradigmas de programación como orientación a objetos, estructurada, programación imperativa y, en menor medida programación funcional. Se utiliza el marco de trabajo PyCharm por su gran rendimiento y flexibilidad, lo que permite un desarrollo ágil y reutilizable. La herramienta de procesamiento de lenguaje natural SpaCy se utiliza por su fácil acceso y todo el potencial que posee.

## Python

**Python[[4]](#footnote-4)** es un lenguaje de programación interpretado, orientado a objetos de alto nivel y con semántica dinámica. Su sintaxis hace énfasis en la legibilidad del código, lo que facilita su depuración y, por tanto, favorece la productividad. Ofrece la potencia y la flexibilidad de los lenguajes compilados con una curva de aprendizaje suave. Aunque Python fue creado como lenguaje de programación de uso general, cuenta con una serie de librerías y entornos de desarrollo para cada una de las fases del proceso de Data Science. Esto, sumado a su potencia, su carácter open source y su facilidad de aprendizaje le ha llevado a tomar la delantera a otros lenguajes propios de la analítica de datos por medio de Machine Learning como pueden ser SAS (software comercial líder hasta el momento). Python fue creado por Guido Van Rossum en 1991 y, como curiosidad, debe su nombre a la gran afición de su creador por las películas del grupo Monty Python. Además de librerías de herramientas científicas, numéricas, de herramientas de análisis y estructuras de datos, o de algoritmos de Machine Learning como NumPy, SciPy, Matplotlib, Pandas o PyBrain, Python ofrece entornos interactivos de programación orientados al Data Science.

## PyCharm (IDE)

**PyCharm[[5]](#footnote-5)**  es un [entorno de desarrollo integrado](https://en.wikipedia.org/wiki/Integrated_development_environment) (IDE) utilizado en [la programación de computadoras](https://en.wikipedia.org/wiki/Computer_programming) , específicamente para el lenguaje de programación [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) . Está desarrollado por la empresa [checa](https://en.wikipedia.org/wiki/Czech_Republic)[JetBrains](https://en.wikipedia.org/wiki/JetBrains) (anteriormente conocida como IntelliJ). Proporciona análisis de código, un depurador gráfico, un probador de unidades integrado, integración con [sistemas de control](https://en.wikipedia.org/wiki/Revision_control) de versiones (VCS) y es compatible con el desarrollo web con [Django](https://en.wikipedia.org/wiki/Django_(web_framework)) , así como con [la ciencia de datos](https://en.wikipedia.org/wiki/Data_science) con [Anaconda](https://en.wikipedia.org/wiki/Anaconda_(Python_distribution)).

PyCharm es [multiplataforma](https://en.wikipedia.org/wiki/Cross-platform) , con versiones de [Windows](https://en.wikipedia.org/wiki/Windows) , [macOS](https://en.wikipedia.org/wiki/MacOS) y [Linux](https://en.wikipedia.org/wiki/Linux) . La edición comunitaria se publica bajo la [licencia Apache](https://en.wikipedia.org/wiki/Apache_License), y también hay una versión educativa, así como una edición profesional con características adicionales (publicada bajo una [licencia propietaria](https://en.wikipedia.org/wiki/Proprietary_software)[financiada por suscripción](https://en.wikipedia.org/wiki/Subscription_business_model) ).

***PyCharm vs Jupyter***

**Jupyter[[6]](#footnote-6)**

Jupyter notebook es un IDE de código abierto que se utiliza para crear documentos Jupyter que se pueden crear y compartir con códigos activos. Además, es un entorno computacional interactivo basado en la web. El cuaderno Jupyter puede admitir varios lenguajes que son populares en la ciencia de datos, como Python, Julia, Scala, R, etc.

**Diferencias entre Jupyter y PyCharm[[7]](#footnote-7):**

Tabla 2 Diferencias entre IDE Python

|  |  |
| --- | --- |
| **Jupyter** | **PyCharm** |
| El portátil Jupyter es una plataforma informática interactiva basada en web. | Pycharm es un editor de código inteligente. |
| El cuaderno combina código en vivo, ecuaciones, texto narrativo, visualizaciones, paneles interactivos y otros medios. | El editor proporciona soporte de primera clase para Python, JavaScript, CoffeeScript, TypeScript, CSS, lenguaje de plantillas popular y más. |
| Se puede clasificar como una herramienta en los "Cuadernos de ciencia de datos" | PyCharm se agrupa en “Entorno de desarrollo integrado(IDE)”. |
| Proporciona ejecución de código en línea mediante bloques. | Proporciona finalización automática inteligente. |
| Puede tener un tema y es compatible con kernel y látex. | Es una poderosa refactorización, integración virtual e integración Git |

## SpaCy

**SpaCy[[8]](#footnote-8)** es una librería de software para  [procesamiento de lenguajes naturales](https://es.wikipedia.org/wiki/Procesamiento_de_lenguajes_naturales) desarrollado por Matt Honnibal y programado en lenguaje [Python](https://es.wikipedia.org/wiki/Python). Fue lanzado en febrero de 2015 estando su desarrollo activo y siendo utilizado en distintos entornos. Es software libre con [Licencia MIT](https://es.wikipedia.org/wiki/Licencia_MIT) su repositorio se encuentra disponible en [Github](https://es.wikipedia.org/wiki/Github).

Características principales:

* Tokenización no destructiva.
* Compatibilidad con tokenización alfa para más de 65 idiomas.
* Soporte integrado para componentes de canalización entrenables, como reconocimiento de entidades nombradas, etiquetado de parte de la voz, análisis de dependencias, clasificación de texto, vinculación de entidades, entre otros.
* Modelos estadísticos para 17 idiomas.
* Aprendizaje multitarea con transformadores previamente entrenados como BERT.

La Universidad Católica de Salta realizó una comparación de herramientas de procesamiento de textos en español para Python y explica lo siguiente [46]:

De cada herramienta se tuvieron en cuenta algunos aspectos como la facilidad en su instalación, sobre todo para una persona con conocimientos básicos o nulos en programación; la necesidad de utilizar una interfaz de comunicación; la presencia de funciones específicas para segmentación y tokenización; el tipo de etiquetado POS que realizan y la implementación de lematizador para el lenguaje español. A modo de resumen se presenta un cuadro comparativo de algunas funciones básicas de procesamiento de texto y las herramientas mencionadas que fueron objeto de evaluación.

Tabla 3 Comparación entre herramientas NLP

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Herramienta** | **Código** | **Segmentación y tokenización** | **Etiquetado POS** | **Lematización** |
| NLTK | Nativo Python | Si | Eagles | No |
| Freeling | API | Si | Eagles | Si |
| Pattern.es | Nativo Python | Si | Penn TreeBank | Si |
| SpaCy | Nativo Python | Si | Penn TreeBank | Si |
| Stanford NLP | API | Si | Eagles | Si |

En la segunda columna de la Tabla se muestra la relación de las herramientas con el lenguaje Python. NLTK, Pattern.es y SpaCy se pueden instalar muy fácilmente de los repositorios de Python a través de la línea de comando. No ocurre lo mismo con Stanford NLP y Freeling. Ambos paquetes no son nativos de Python; fueron escritos en Java y C++, respectivamente. La instalación de Stanford es más sencilla que la de Freeling, aunque ambos necesitan sendos servidores que se estén ejecutando para funcionar. Sin embargo, si no se requiere de todo el potencial de ambas herramientas, existe otra opción para su uso: Stanford se puede utilizar a través de NLTK y Freeling provee un ejecutable que realiza la segmentación, la tokenización, el etiquetado POS y la lematización recibiendo un archivo como entrada. Todas las herramientas poseen funciones para segmentación y tokenización. NLTK, Freeling y Stanford NLP realizan el etiquetado POS con etiquetas Eagles mientras que Pattern.es y SpaCy lo hacen con etiquetas de Penn TreeBank. Salvo NLTK, las demás herramientas, implementan un lematizador para el idioma español. El lematizador disponible en NLTK, WordNet, está disponible para el idioma inglés.

NLTK requiere que se hagan una serie de operaciones sobre el texto de segmentación, tokenizacion, etiquetado y Chunking antes de hacer el reconocimiento de entidades nombradas como tal, esto más control al usuario sobre el proceso. Por otro lado, en SpaCy se permite trabajar sobre un texto a nivel de entidad directamente, lo que simplifica mucho el proceso.

# Conclusiones parciales

Al terminar el capítulo de estudio teórico se puede concluir que:

* El lenguaje natural constituye la principal fuente de información para la ingeniaría de requisitos, pero sus características dificultan el proceso ocasionando elevados valores en las estadísticas de fallos de los proyectos.
* Existen numerosos estándares y formalizaciones de requisitos, pero el lenguaje natural y la utilización de modelos de forma independientes o combinadas en una estructura son las más comunes.
* En el mercado se encuentran presentes varias herramientas de NLP como parte del proceso de RE, en este estudio se incluyeron solo aquellas a las cual se tuvo acceso por ser Open Source.
* Se justificó el uso del lenguaje de programación Python para implementar nuestra propuesta de solución, así como de las librerías de SpaCy y IDE PyCharm que es la plataforma líder para crear programas Python que funcionen con datos de lenguaje humano.

# Capítulo 2. Extracción automática de requisitos de software

En el presente capítulo se documenta toda la propuesta de solución. Se exponen los elementos que forman parte de la solución. Se presentan los diagramas en Lenguaje Unificado de Modelado (UML, Unified Modeling Language) necesarios para explicar el flujo de las actividades que forman parte de la arquitectura de la solución propuesta, como los casos de uso y modelo del dominio. Se muestra para un mejor entendimiento de la solución las principales etapas de desarrollo en un diagrama de flujo.

# Solución teórica

En esta se sección describe la solución propuesta en el trabajo. Las principales etapas del proceso de desarrollo se describen en la Ilustración 3. En primer lugar, el texto a analizar es pre-procesado, se hace una limpieza del texto, eliminando signos de puntuación y convirtiéndolo en su totalidad a letras minúsculas. A continuación, se lleva acabo el análisis sintáctico, a partir del etiquetado de cada tokens del texto según su clasificación sintáctica, se realiza un mapeo con patrones basados en reglas definidos previamente, extrayendo así frases que dan lugar a sentencias de requisitos de software. Tomando como base las frases extraídas en la fase anterior se realiza un refinamiento del volumen de sentencias extraídas, reduciendo a una, aquellas que sean completamente iguales. Luego se agrupan aquellas que tengan un 90% de similitud entre ellas. Finalmente se exporta un archivo txt donde se muestran los resultados obtenidos.

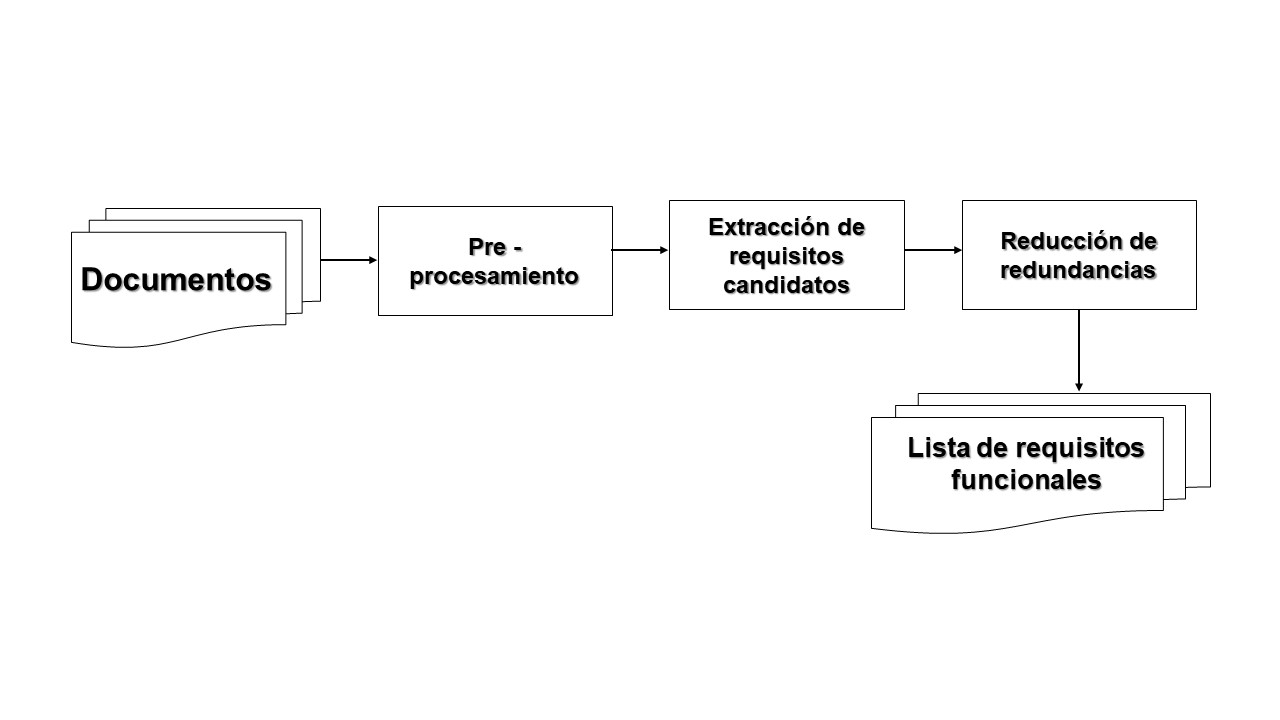


Ilustración 3 Flujo de trabajo de la solución propuesta

## Pre – procesamiento

Como se planteaba en la subsección anterior la solución contará fundamentalmente con un componente para la preparación o pre-procesamiento de la información haciendo uso de la biblioteca de Procesamiento de Lenguaje Natural para español, SpaCy.

El componente de pre-procesamiento se encargará de limpiar el texto de ruido y estandarizar para que sea fácil de procesar. Algunos de estos pasos incluyen quitar las puntuaciones, transformar el texto a minúscula, o eliminar palabras frecuentes usadas en el lenguaje como preposiciones, artículos, entre otros, que por sí mismas no aportan mucha información, además de usar la raíz de las palabras. En la Ilustración 4 se muestra un diagrama de actividades del pre - procesamiento.

Dentro de las principales actividades dentro del módulo de pre-procesamiento se encuentra:

* Tokenización: esta actividad consiste en dividir el texto sin procesar en pequeños trozos. La tokenización dividirá el texto en bruto en palabras u oraciones llamadas tokens. Estos tokens permitirán comprender el contexto o desarrollar el modelo para la NLP, ayudando a interpretar el significado del texto al analizar la secuencia de las palabras.
* Tras la tokenización, se procede al stopwords. Se trata de palabras muy habituales en cualquier idioma que, sin embargo, aportan poco valor tales como artículos, signos de puntuación, palabras de enlace, etc. Por ello, es interesante identificarlas y «excluirlas». Es una forma de «limpiar» el texto.
* Etiquetado PoS (Part-of-speech) tagging: Se encarga de clasificar las partes de las oraciones en verbo, sustantivo, adjetivo, preposición entre otras.
* Lematización: es el proceso mediante el cual las palabras de un texto que pertenecen a un mismo paradigma flexivo o derivativo son llevadas a una forma normal que representa a toda la clase. En este caso solo se les realiza a los verbos conjugados.

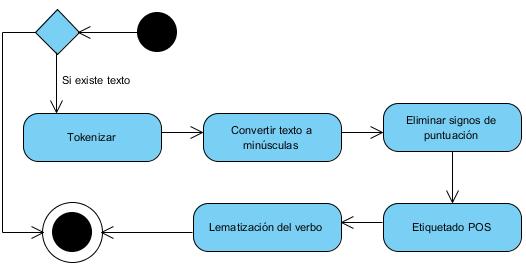


Ilustración 4 Diagrama de actividades del pre - procesamiento

## Extracción de requisitos candidatos

El análisis sintáctico tiene como función etiquetar cada uno de los componentes sintácticos que aparecen en la oración y analizar cómo las palabras se combinan para formar construcciones gramaticalmente correctas. El resultado de este proceso consiste en generar la estructura correspondiente a las categorías sintácticas formadas por cada una de las unidades léxicas que aparecen en la oración.

### Análisis sintáctico basado en patrones léxicos – sintácticos

Un **patrón** es una descripción de la forma que pueden tomar los lexemas de un token. En el caso de una palabra clave como token, el patrón es sólo la secuencia de caracteres que forman la palabra clave. Para los identificadores y algunos otros tokens, el patrón es una estructura más compleja que se relaciona mediante muchas cadenas.

Para extraer las frases que dan lugar a los requisitos candidatos, se definieron previamente un conjunto de patrones léxicos – sintácticos que fueron generados a partir de un proceso estadístico realizado a 40 tesis del curso 2020-2021 de la facultad de Ingeniería Informática de la CUJAE, donde fueron tomados cada uno de los requisitos funcionales de los diagramas de casos de uso, dando lugar a un total de 555 requisitos funcionales analizados. Estos requisitos fueron procesados por el analizador sintáctico de la biblioteca de SpaCy y a partir de la etiqueta gramatical de cada tokens fueron formados los patrones.

Luego de realizado el pre-procesamiento del texto, partiendo de la clasificación gramatical de cada tokens se realiza un mapeo con los patrones predefinidos y se generan las frases que dan lugar a las sentencias de requisitos.

En la tabla se muestran cada uno de estos patrones léxico – sintácticos.

Tabla 4 Patrones léxicos - sintácticos

|  |  |
| --- | --- |
| ***Patrón léxico – sintáctico*** | ***Ejemplo de frase*** |
| VERB NOUN | analizar muestras, recetar medicamentos |
| VER NOUN ADJ | analizar muestras bilógicas |
| VERB DET NOUN | establecer un diagnóstico, entregar las muestras |
| VERB NOUN ADP NOUN | incluir detectores de humo |
| VERB NOUN ADP DET NOUN | recoger datos sobre su estructura |
| VERB NOUN ADP NOUN ADJ | obtener reportes en tiempo real |
| VERB NOUN ADP NOUN ADP NOUN | exportar base de datos del día |
| VERB NOUN ADJ ADP NOUN | gestionar tratamientos asociados al paciente |
| VERB NOUN VERB DET NOUN ADJ | reproducir voz alertando el billete reconocido |
| VERB ADJ PRON VERB ADJ DET NOUN ADJ | ejecutar nodo que tiene implementado el algoritmo RSKkNN |
| VERB ADJ PRON VERB DET NOUN AUX | ejecutar nodo que crea el árbol IUR-tree |
| VERB CCONJ VERB DET NOUN ADP NOUN | salvar y restaurar la base de datos |
| VERB ADV ADP NOUN ADP DET NOUN ADJ | organizar No. de lista de los estudiantes matriculados |
| VERB NOUN CCONJ NOUN | enviar reclamación o protesta |
| VERB DET NOUN ADJ | explicar la propuesta diseñada |
| VERB DET NOUN CCONJ DET NOUN | visualizar las delegaciones y sus atletas |
| VERB ADP NOUN | apelar a sanción |
| VERB NOUN ADJ ADP NOUN ADJ | cifrar código binario con contraseña modulada |
| VERB NOUN ADJ ADJ ADP DET NOUN ADP NOUN | obtener código binario original en el buffer de memoria |
| VERB NOUN ADP NOUN ADP NOUN | enviar correo de petición de acceso |

### Análisis sintáctico basado análisis de dependencias

La idea fundamental de dependencia está basada en que la estructura sintáctica de una frase, consiste en relaciones binarias asimétricas entre las palabras de esa frase [47]. Por tanto, han de establecerse criterios para definir qué relaciones de dependencia existen, para distinguir de qué forma están relacionadas dos palabras en una frase y si esas relaciones están etiquetadas o no. Por tanto, a partir de las relaciones asimétricas y los diferentes criterios; dada una frase, en cualquier lenguaje, se puede establecer un árbol sintáctico de dependencias etiquetado, como el del ejemplo que se muestra en la Figura. En este ejemplo se observa el análisis de dependencias de la frase en español: “Las universidades consideradas cobran tarifas elevadas”. Se puede razonar que dada esa frase y conociendo cierta información sintáctica de cada uno de los tokens que aparecen en la misma, se pueden establecer distintas relaciones de dependencia, donde la raíz de todas ellas es la acción principal de la frase, dígase el verbo. Con la información contenida en un árbol de dependencias se pueden realizar múltiples tareas, como simplificación de textos [48], reconocimiento de la implicación textual [49], detección de conceptos negados en una frase [50], etc.

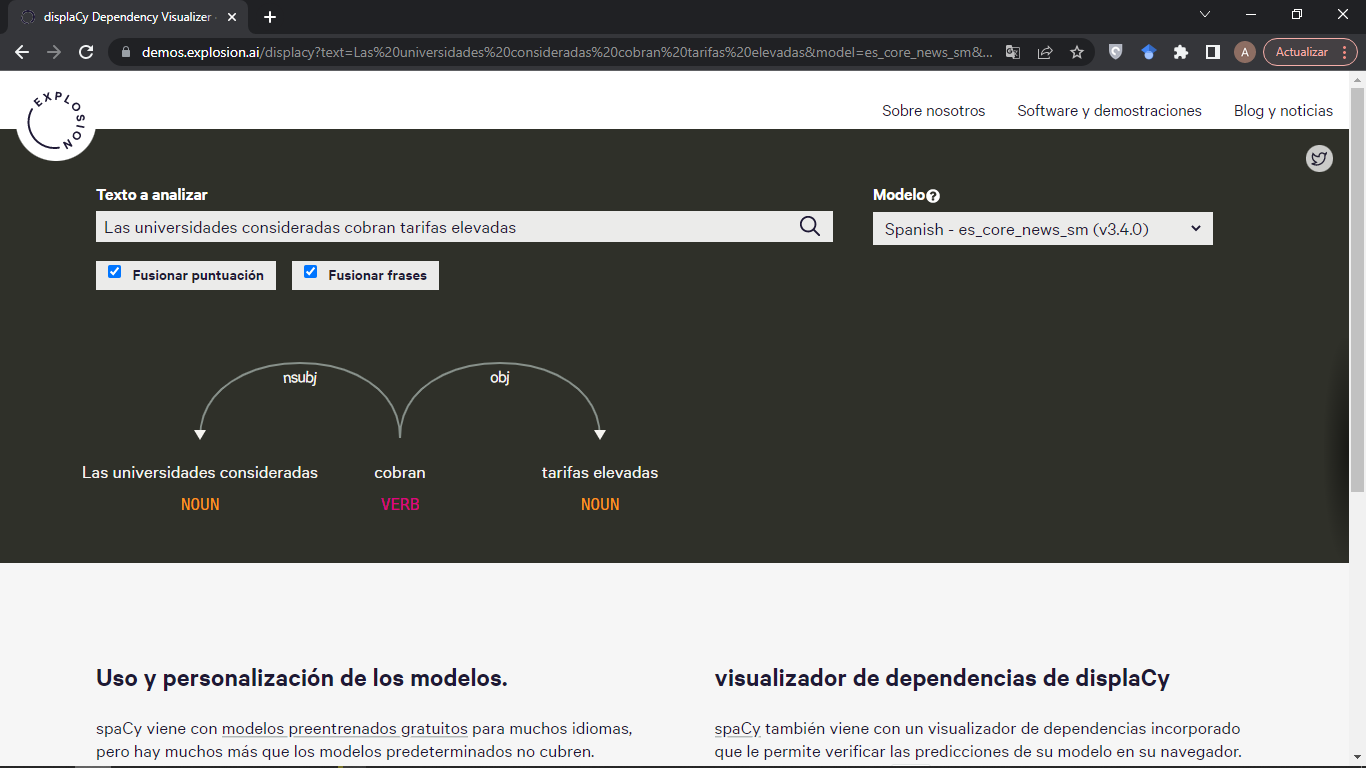


Ilustración 5 Árbol de dependencias generado por el analizador sintáctico SpaCy

En el trabajo se usará esta técnica de extracción de información como una segunda vía para la generación de los requisitos candidatos. Basándose en lo planteado por [45], se estará utilizando un patrón para extraer todos las frases con una misma estructura, la combinación del verbo en conjunto con el sujeto de la oración y el objeto al que se realiza la acción. El patrón se forma con las etiquetas de dependencias de la biblioteca SpaCy. Se recorre todo el árbol de dependencias generado por la biblioteca y se extraen las frases cumpliendo con dicha estructura.

Reutilizando el ejemplo anteriormente mencionado la frase generada por este método de extracción quedaría de la siguiente forma:

Tabla 5 Ejemplo de extracción usando análisis de dependencias

|  |  |
| --- | --- |
| **Patrón** | **Frase** |
| VERB + nsubj + obj | cobran las universidades consideradas tarifas elevadas |

## Reducción de redundancias

Como estrategia de reducción de redundancias se definió la siguiente:

* Reducir a una, aquellas frases que sean completamente iguales, almacenando las restantes en otro archivo independiente a la solución, debido a que el objetivo de la solución propuesta es asistir el trabajo del analista – diseñador del software, se decidió no eliminar ninguna frase generada y que sea el especialista encargado quien decida la relevancia de la misma.
* Agrupar aquellas frases que tengan un 90% de similitud sintáctica entre ellas. Se utilizan dos métricas de similitud para agrupar:
* Métrica Levenshtein : La distancia de [Levenshtein](https://es.wikipedia.org/wiki/Distancia_de_Levenshtein) es quizá la más común y simple de las métricas de edición basadas en caracteres, por lo que generalmente (y quizá también por la dificultad de pronunciación) se le conoce simplemente como “distancia de edición”. Esta métrica es, en cierto sentido, una forma de cuantificar la diferencia entre dos cadenas de texto, sin embargo, aunque por su descripción parezca sencilla, la forma de calcularla no es trivial como en el caso de la distancia de Hamming, además, a diferencia de ésta última, la distancia de Levenshtein tradicional es capaz de comparar cadenas de diferente tamaño.
* Métrica Similitud de coseno: En [el análisis de datos](https://en.wikipedia.org/wiki/Data_analysis) , la similitud del coseno es una [medida de similitud](https://en.wikipedia.org/wiki/Measure_of_similarity) entre dos secuencias de números. Para definirlo, las sucesiones se ven como vectores en un [espacio de producto interior](https://en.wikipedia.org/wiki/Inner_product_space) , y la similitud de coseno se define como el [coseno](https://en.wikipedia.org/wiki/Cosine) del ángulo entre ellos, es decir, el [producto escalar](https://en.wikipedia.org/wiki/Dot_product) de los vectores dividido por el producto de sus longitudes. De ello se deduce que la similitud del coseno no depende de las magnitudes de los vectores, sino solo de su ángulo. La semejanza del coseno siempre pertenece al intervalo [1, -1]. La similitud del coseno se usa particularmente en el espacio positivo, donde el resultado está claramente delimitado en [0, 1].
* Al calcular la similitud entre las frases se generan sublistas con cada uno de los grupos y se devuelve una lista general con cada uno de los grupos.

# Desarrollo de la solución de extracción de requisitos de software

Explicación de casos de uso del sistema, actores del sistema y patrones presentes en el proyecto.

## Diagrama de casos de uso y descripción de casos de uso

Un Modelo de Casos de Uso o Modelo de Vista de Casos de Uso constituye un modelo donde se perciben las funcionalidades del sistema por usuarios externos llamados actores. Un caso de uso es una unidad coherente de funcionalidad expresada como una transacción entre los actores y el sistema. El propósito de esta vista es listar los actores y los casos de usos y mostrar que actores participan en cada caso de uso. En la Ilustración 6 se muestra el diagrama de Casos de Uso del Sistema que se diseñó como parte de la propuesta.

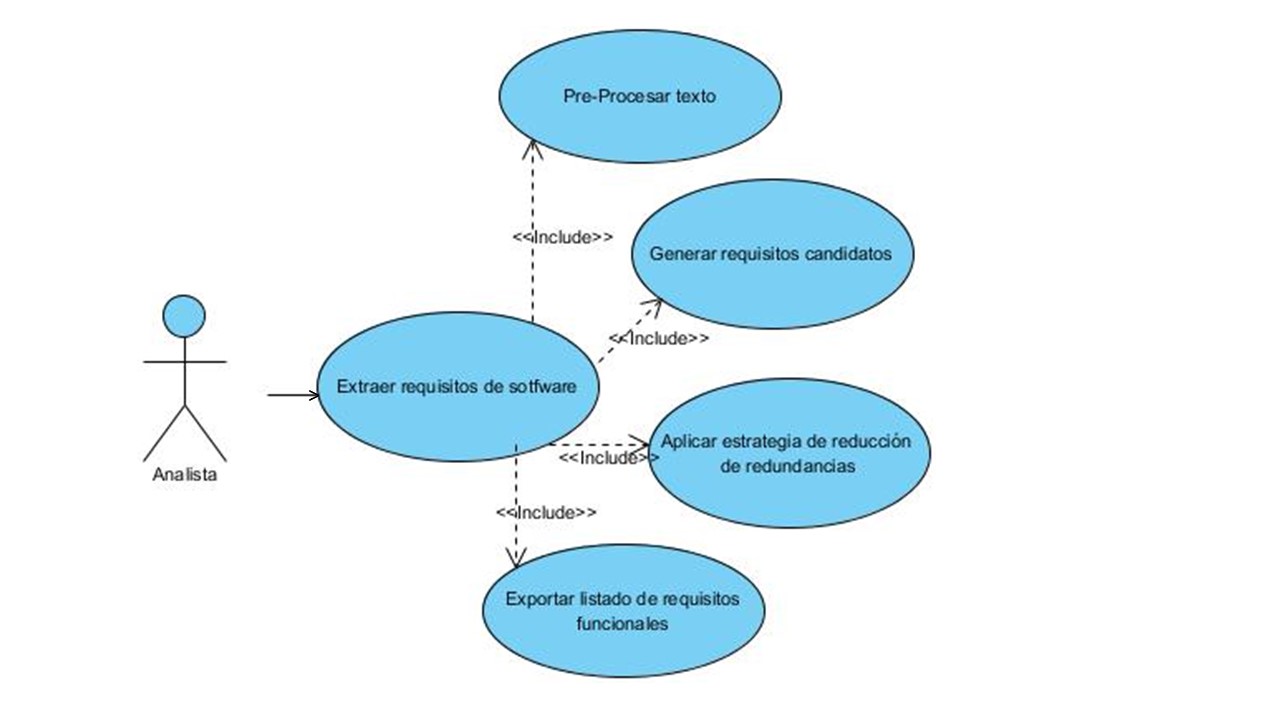


Ilustración 6 Diagrama de caso de uso

### Actores del sistema

En la Tabla se muestra los actores del sistema y una descripción.

Tabla 6 Actores del sistema y una descripción

|  |  |
| --- | --- |
| **Actores del sistema** | **Descripción** |
| Analista | Es el actor que interactúa directamente con el sistema. Es el encargado de iniciar todos los procesos del sistema. Utiliza el sistema para generar los requisitos de software. |

### Especificación de alto nivel de caso de uso

La especificación de cada caso de uso se muestra en la Tabla 7.

Tabla 7 Descripción del caso de uso Extraer requisitos de software.

|  |  |
| --- | --- |
| **Caso de uso** | Extraer requisitos de software |
| **Actores** | Analista |
| **Propósito** | Generar las sentencias que dan lugar a requisitos funcionales de software. |
| **Descripción** | Para la extracción de los requisitos el analista debe, inicialmente, cargar el documento a procesar. |
| **Responsabilidad** | Cargar el archivo con el texto a procesar. |
| **Precondiciones** | Archivo en formato de texto |
| **Poscondiciones** | Se obtiene listado de requisitos funcionales |

## Arquitectura del modelo

La implementación de la solución se lleva a cabo con el lenguaje de programación Python, con la versión 3.9.0, y PyCharm como repositorio de los paquetes y librerías. Python tiene muchas librerías para el desarrollo de diversos tipos de aplicaciones, en este trabajo en particular se emplean la librería SpaCy famosa en el empleo de análisis de fuentes textuales de información, con las que se pueden realizar tareas propias de la minería de texto como el pre - procesamiento de los datos, el agrupamiento, entre otras.

El modelo se divide en 5 paquetes principales: cargar\_varios\_archivos, pre\_procesamiento, reducción\_redundancias, exportar\_resultados y SpaCy. La estructura de estos paquetes y la interacción entre ellos se representa en la Ilustración 7, usando un diseño multicapas.

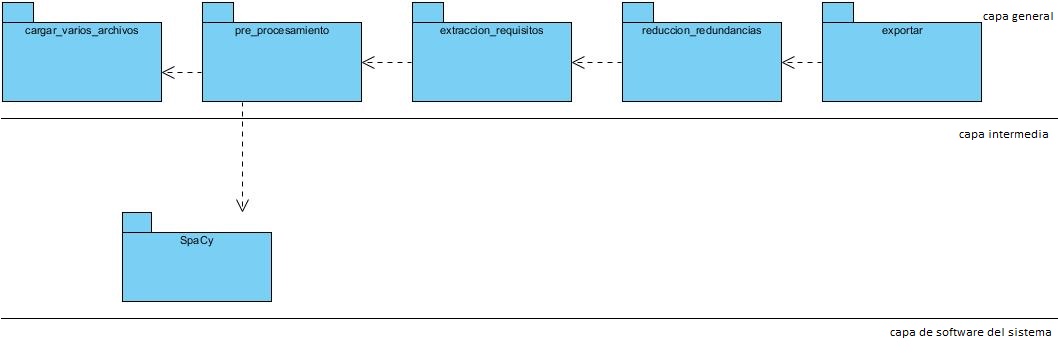


Ilustración 7 Estructura en capas del modelo.

## Estilos y patrones de arquitectura

**Arquitectura basada en componentes**:

Representa una división del sistema en módulos más pequeños o subsistemas, que son unidades de software los cuales contienen un conjunto de funciones que tienen alguna relación entre ellas, realizan las acciones del sistema y que se implementan en archivos físicos.

Descripción detallada de los paquetes:

cargar\_varios\_archivos: paquete donde se implementa la funcionalidad de cargar el archivo a procesar.

pre\_procesamiento: paquete donde se implementan las diferentes funcionalidades encargadas de la preparación del texto a procesar (tokenizacion, etiquetado, análisis sintáctico).

reducción\_redundancias: paquete donde se implementan la funcionalidad encargada de eliminar freses repetidas (eliminar\_redundancias) y la funcionalidad de agrupar dichas frases por un 90% de similitud entre ellas (agrupar\_por\_similitud)

exportar\_resultados: paquete donde se exportan en archivo .txt las sentencias de requisitos generadas.

extraccion\_requisitos: modulo general donde se invocan cada uno de los paquetes y sus respectivas funciones.

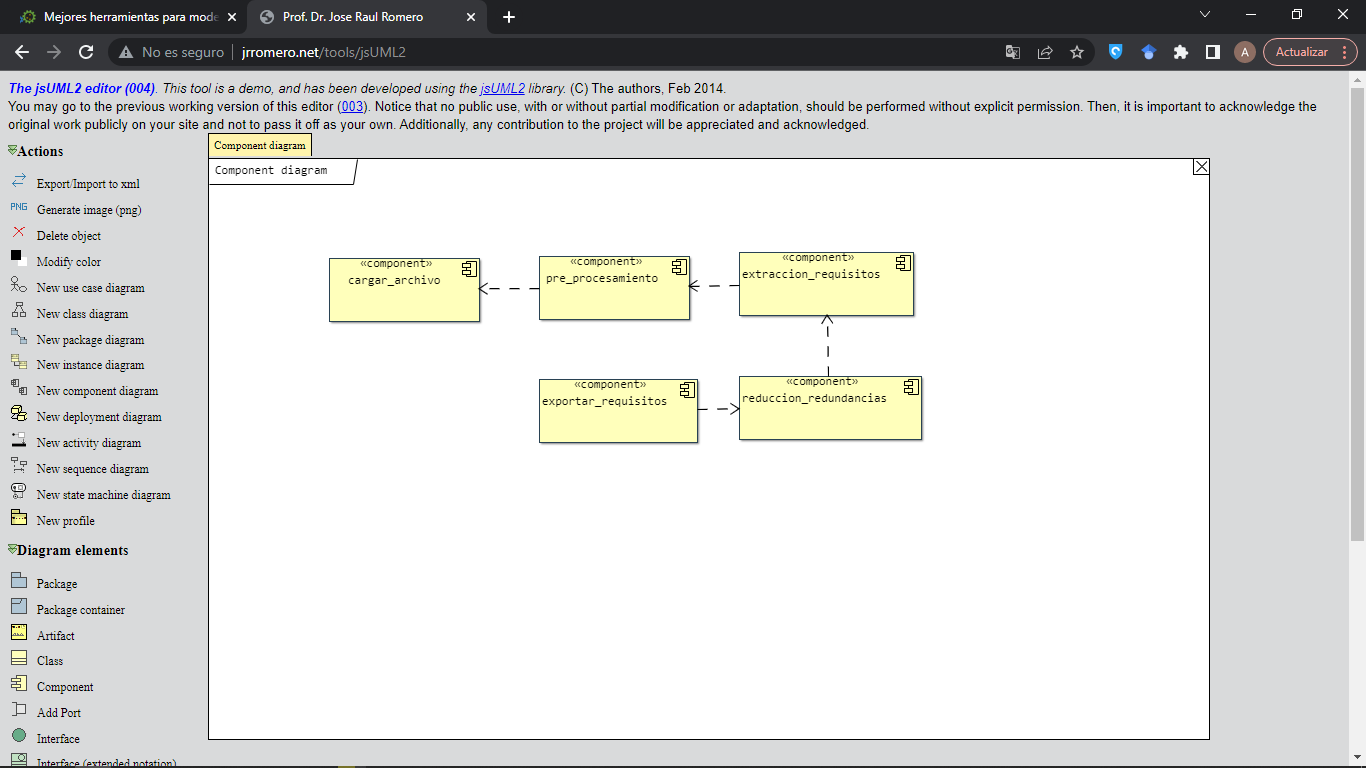


Ilustración 8 Estructura modular del sistema. Arquitectura basada en componentes.

**Patrón Filtros y Tuberías**:

Representa el flujo de trabajo de la solución propuesta mencionado al inicio del capítulo.

# Conclusiones parciales

Como conclusiones del capítulo se puede afirmar que:

* En el estudio bibliográfico realizado la mayor parte de trabajos se basan en la extracción de información en patrones o reglas léxicos sintácticas, la mayoría procesa textos en idioma español y no se aplica una estrategia de reducción de redundancias.
* Se desarrolló un método para la extracción automática de requisitos de software
* Se aplicaron dos técnicas de extracción de información en el método desarrollado, análisis sintáctico basado en patrones léxicos sintácticos y análisis sintáctico basado en análisis de dependencias.
* Se aplicaron métricas de similitud sintáctica (Coseno y Levenshtein).

# Capítulo 3. Evaluación de la solución propuesta

En el presente capítulo se presenta el estudio experimental de la solución propuesta. Se evalúan los requisitos extraídos de manera automática de un grupo de caso de estudios definidos, comparándolos con los requisitos extraídos de forma manual, por un especialista en la materia, de dichos casos de estudios. Se llevó a cabo la evaluación de los mismos empleando las métricas habituales en la clasificación de texto, llamadas Precisión (P), Cobertura (C) y Medida-F (F).

Tabla 8 Descripción de casos de estudio.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Casos de estudio** |  | **Características** | | |
| **oraciones** | **palabras** | **requisitos** | **dominio** |
| Sistema de seguridad vivienda | 34 | 614 | 17 | seguridad |
| Sistema de seguridad evento deportivo | 34 | 629 | 17 | seguridad |
| Decoración de interiores | 22 | 503 | 18 | decoración |
| Inversiones en telefonía fija | 21 | 583 | 18 | telecomunicaciones |
| Distribución de combustible | 26 | 593 | 18 | administración |
| Gestión cursos de posgrado | 67 | 1074 | 54 | docencia |
| Gestión de reservas de casas de campo | 23 | 607 | 18 | recreación |
| Préstamo de libros | 16 | 408 | 16 | docencia |
| Préstamo de video | 16 | 405 | 16 | docencia |
| Salud mental | 13 | 302 | 12 | salud |
| Atención a niños | 13 | 296 | 6 | salud |
| Reserva de habitaciones de un hotel | 71 | 2342 | 25 | turismo |
| Agencias de viajes | 96 | 2224 | 31 | turismo |
| Sistema de control para una bomba de insulina | 18 | 376 | 6 | salud |

# Marco de evaluación

La solución propuesta fue evaluada tomando como referencias la extracción manual de requisitos de software de un conjunto de casos de estudio que se definieron previamente por el equipo de trabajo. Estos requisitos fueron comparados sintácticamente con los requisitos extraídos de forma automática, obteniendo una tercera lista de requisitos. Posteriormente se evalúan los resultados usando las métricas de evaluación recisión, cobertura y medida-F.

## Ejemplo de extracción automática

Entrevista: Sistema de seguridad de vivienda

**Empresa:** ¿Por qué vendría a nuestra empresa a solicitar servicios?

**Cliente:** Mi casa tiene varias fuentes de entrada y necesita un sistema de seguridad que me permita detectar movimientos cuando no hay nadie en la casa, que, si alguien intenta entrar por alguna ventana o puerta, me alerte, y si se produce algún escape de gas o de humo, también me alerte.

**Empresa:** Nosotros tenemos lo que Ud. necesita, nuestra propuesta incluye detectores de humo y gas, sensores de ventana y puertas y detectores de movimiento. Además, si a Ud. le interesa cuando se produce alguna incidencia, se puede llamar automáticamente al cuerpo de bomberos (por escape de gas o emisión de humo) o a la policía (por actuación de los sensores de movimiento, puerta o ventana).

**Cliente:** Perfecto,¿cómo podríamos acceder a sus servicios?

**Empresa:** Ud.se entrevista con el comercial de nuestra empresa suministrándole la dirección de su vivienda y el nombre del propietario. En este momento, el comercial elabora un contrato que recoge el estudio de factibilidad que respalda el análisis que hará el especialista de seguridad sobre la propuesta de sensores y paneles de control adecuados para proteger su vivienda de situaciones indeseables tales como irrupciones ilegales, fuego y escape de gas. Una vez firmado el contrato por el cliente, el comercial pasa al especialista de seguridad, esta última visita la instalación para recoger datos sobre su estructura y elaborar una propuesta de diseño del sistema de seguridad. Esta propuesta se pasa a un especialista en economía que cuantifica el costo de instalación, de mantenimiento de la infraestructura instalada y de atención a las incidencias que se produzcan.

Requisitos extraídos automáticamente

Tabla 9 Requisitos extraídos usando patrones léxicos sintácticos

|  |  |
| --- | --- |
| **Requisitos** | **Patrón al que responde** |
| solicitar servicios | VERB + NOUN |
| tener varias fuentes | VERB + DET + NOUN |
| necesitar un sistema | VERB + DET + NOUN |
| detectar movimientos | VERB + NOUN |
| producir algún escape | VERB + DET + NOUN |
| necesitar nuestra propuesta | VERB + DET + NOUN |
| incluir detectores de humo | VERB + NOUN + ADP + NOUN |
| producir alguna incidencia | VERB + DET + NOUN |
| suministrar él la dirección | VERB + DET + NOUN |
| elaborar un contrato | VERB + DET + NOUN |
| recoger el estudio | VERB + DET + NOUN |
| respaldar el análisis | VERB + DET + NOUN |
| hacer el especialista | VERB + DET + NOUN |
| proteger su vivienda | VERB + DET + NOUN |
| firmar el contrato | VERB + DET + NOUN |
| pasar al especialista | VERB + DET + NOUN |
| recoger datos sobre su estructura | VERB + NOUN + ADP + DET + NOUN |
| elaborar una propuesta | VERB + DET + NOUN |
| cuantificar el costo | VERB + DET + NOUN |

Tabla 10 Requisitos extraídos usando análisis de dependencias

|  |
| --- |
| **Requisitos** |
| tiene entrevista sistema de seguridad de vivienda |
| empresa por qué vendría a nuestra empresa a solicitar servicios |
| cliente mi casa fuentes, |
| tenemos |
| empresa nosotros ud, |
| necesita, |
| además, |
| llamar cuerpo, |
| cliente, |
| entrevista comercial, |
| elabora el comercial contrato, |

# Resultados y discusión

En esta sección se sintetizan y analizan los resultados obtenidos en los experimentos realizados con la colección de casos de estudio para satisfacer los objetivos de evaluación planteados. Las métricas de Precisión y Cobertura son computadas comparando el requisito que se obtiene extrayendo información del texto, con el elaborado manualmente por el experto. En esa comparación se empleó la distancia Levenshtein, usando como umbral de aceptación el 60%. Los resultados obtenidos en los experimentos realizados con la colección de prueba se muestran en la siguiente Tabla.

Tabla 11 Resultados de la evaluación del método con documentos de entrevistas a usuarios.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | ***Usando patrones léxico-sintáctico*** | | | ***Usando análisis de dependencias*** | | | ***Extracción basada en enfoque híbrido*** | | |
| ***P*** | ***C*** | ***F*** | ***P*** | ***C*** | ***F*** | ***P*** | ***C*** | ***F*** |
| **Entrevista 1** | 30.55 | 55 | 39.28 | 19.35 | 30 | 23.52 | 25.75 | 85 | 39.53 |
| **Entrevista 2** | 32.55 | 82.35 | 46.66 | 29.03 | 52.94 | 37.5 | 51.11 | 67 | 31.5 |
| **Entrevista 3** | 27.77 | 55.55 | 37.03 | 15.78 | 16.66 | 16.21 | 23.63 | 72.22 | 35.61 |
| **Entrevista 4** | 25.64 | 50 | 33.89 | 30 | 30 | 30 | 27.11 | 80 | 40.5 |
| **Entrevista 5** | 29.72 | 61.11 | 40 | 26.08 | 33.33 | 29.26 | 43.58 | 94.44 | 28.33 |
| **Entrevista 6** | 35 | 51.85 | 41.79 | 45.16 | 51.85 | 48.27 | 39.71 | 73.7 | 57.43 |
| **Entrevista 7** | 41.37 | 35.28 | 38.09 | 5 | 2.94 | 3.7 | 26.53 | 38.23 | 31.32 |
| **Entrevista 8** | 59.25 | 82 | 74.41 | 35.71 | 31.25 | 33.33 | 50 | 96 | 71.42 |
| **Entrevista 9** | 67.85 | 96.75 | 86.36 | 50 | 43.75 | 46.66 | 61.9 | 93.5 | 89.65 |
| **Entrevista 10** | 16.66 | 33.33 | 22.22 | 8.33 | 8.33 | 8.33 | 13.88 | 41.66 | 20.83 |
| **Entrevista 11** | 16 | 26.66 | 20 | 16.66 | 13.33 | 14.81 | 16.21 | 40 | 23.07 |
| **Entrevista 12** | 24.5 | 50 | 39.37 | 22.22 | 56 | 31.81 | 26.22 | 65 | 41.93 |
| **Entrevista 13** | 45.34 | 95.8 | 66.66 | 19.75 | 51.61 | 28.57 | 31.28 | 64.51 | 52.57 |
| **Entrevista 14** | 21.73 | 83.33 | 34.48 | 11.11 | 33.33 | 16.66 | 17.07 | 97.66 | 29.78 |
| **Promedio** | ***33.85*** | ***61.36*** | ***44.30*** | ***23.87*** | ***32.52*** | ***26.33*** | ***32.43*** | ***72.07*** | ***42.39*** |

Los resultados obtenidos muestran, en la mayoría de casos, valores relativamente altos, resaltando el valor de la cobertura que se muestra alto en comparación con la precisión. Se demostró que la mejor combinación para medir resultados es aplicando métrica de similitud sintáctica Levenshtein a un 60% de similitud entre los requisitos candidatos extraídos. En comparación entre patrones y análisis de dependencias la extracción por patrones léxicos sintácticos arroja resultados relativamente más elevados.

# Conclusiones

Con la culminación del presente trabajo y el análisis de los resultados obtenidos se puede concluir que:

# Recomendaciones

Para darle continuidad a la investigación realizada se proponen las siguientes recomendaciones:

# Referencias Bibliográficas

|  |  |
| --- | --- |
| [1] | S. Group, «Chaos Report,» 2018. |
| [2] | E. S. Calisaya, «Construcción de una herramienta para el análisis de requisitos de software descritos en lenguaje natural,» Universidad Nacional de San Agustín de Arequipa, Perú, 2019. |
| [3] | B. W. Boehm, «Software Engineering Economics,» *IEEE Transactions on Software Engineering,* vol. 10, pp. 4 - 21, 1984. |
| [4] | B. Boehm, «Verifying and Validating Software Requirements and Design Specifications,» *IEEE,* vol. 1, nº 1, p. 75–88, 1984. |
| [5] | A. Lamsweerde, R. Darimont y E. Letier, «Managing conflicts in goal-driven requirements engineering,» *IEEE transactions on Software engineering,* vol. 24, nº 11, p. 908–926, 1998. |
| [6] | C. Denger, D. Berry y E. Kamsties, «Higher quality requirements specifications through natural language patterns,» de *Proceedings 2003 Symposium on Security and Privacy*, Herzlia, Israel, 2003. |
| [7] | R. Vlas y W. N. Robinson, «A Rule-Based Natural Language Technique for Requirements Discovery and Classification in Open-Source Software Development Projects,» de *44th Hawaii International Conference on System Sciences*, Kauai, HI, USA, 2011. |
| [8] | K. Ryan, «The role of natural language in requirements engineering,» de *Proceedings of the IEEE International Symposium on Requirements Engineering*, San Diego, CA, USA, 1993. |
| [9] | J. Hirschberg y C. D. Manning, «Advances in natural language processing,» *Science,* vol. 349, nº 6245, p. 261–266, 2015. |
| [10] | G. Goth, «Deep or shallow, NLP is breaking out,» *Communications of the ACM,* vol. 19, nº 3, p. 13–16, 2016. |
| [11] | A. Ferrari, F. Dell’Orletta, A. Esuli, V. Gervasi y S. Gnesi, «Natural language requirements processing: A 4D vision.,» *IEEE,* p. 28–35, 2017. |
| [12] | F. Dalpiaz, A. Ferrari, X. Franch y C. Palomares, «Natural Language Processing for Requirements Engineering: The Best Is Yet to Come,» *IEEE Software,* vol. 35, nº 5, p. 115–119, 2018. |
| [13] | D. M. Fernández, «Naming the pain in requirements engineering,» *Empirical Software Engineering,* vol. 22, p. 2298–2338, 2017. |
| [14] | I. Sommerville, Ingeniería de Software, Ciudad de México: Pearson Education, 2011. |
| [15] | C. Rupp y K. Pohl, Requirements Engineering Fundamentals, California, 2017. |
| [16] | R. S. Pressman, Ingeniería del Software. Un enfoque Práctico. Séptima Edición, Mexico, 2010. |
| [17] | P. Kruchten, The Rational Unified Process: An Introduction, Addison-Wesley, 2001. |
| [18] | International Organization for Standardization, *[ISO/IEC/IEEE 29148:2011] Systems and software engineering Life cycle processes – Requirements engineering,* Geneva, 2011. |
| [19] | M. K. S. S. Mary Beth Chrissis, CMMI for Development, Guidelines for Process Integration and Product Improvement, Boston, MA: Addison-Wesley, Carnegie Mellon Software Engineering Institute (SEI) , 2011. |
| [20] | S. Richa, B. Jaspreet y B. K.K., «Automated identification of business rules in requirements documents,» de *IEEE International Advance Computing Conference (IACC)*, Gurgaon, India, 2014. |
| [21] | B. Ebrahim, E. Faezeh y G. Dragan, «Decision support for the software product line domain engineering lifecycle,» *Automated Software Engineering,* vol. 16, pp. 335-377, 2012. |
| [22] | M. Jason, S. Shahryar, H. Thomas y E. Timothy, «Software requirements prioritization and selection using linguistic tools and constraint solvers—a controlled experiment,,» *Empirical Software Engineering,* vol. 20, p. 1721–1761, 2015. |
| [23] | C. Duan, P. Laurent, J. Cleland-Huang y C. Kwiatkowski, «Towards automated requirements prioritization and triage,» *Requirements Engineering,* vol. 14, pp. 73-89, 2009. |
| [24] | J. Guo y D. Yang, «Data-efficient performance learning for configurable systems,» *Empirical Software Engineering,* vol. 23, p. 1826–1867, 2018. |
| [25] | C. Fitzgerald, E. Letier y A. Finkelstein, «Early failure prediction in feature request management systems: an extended study,» *Requirements Engineering,* vol. 17, p. 117–132, 2012. |
| [26] | J. d. Sagrado y I. d. Águila, «Stability prediction of the software requirements specification,» *Software Quality Journal,* vol. 26, pp. 585-605, 2018. |
| [27] | I. M. d. Águila y J. d. Sagrado, «Requirement risk level forecast using Bayesian networks classifiers,» *International Journal of Software Engineering and Knowledge Engineering,* vol. 21, nº 02, pp. 167-190, 2011. |
| [28] | Z. Kurtanović y W. Maalej, «Automatically Classifying Functional and Non-functional Requirements Using Supervised Machine Learning,» de *IEEE 25th International Requirements Engineering Conference (RE)*, Lisbon, Portugal, 2017. |
| [29] | A. Dekhtyar y V. Fong, «RE Data Challenge: Requirements Identification with Word2Vec and TensorFlow,» de *IEEE 25th International Requirements Engineering Conference (RE)*, Lisbon, Portugal, 2017. |
| [30] | L. Zeheng y C. Mingrui, «Tracing requirements in software design,» de *International Conference on Software and System Process*, 2017. |
| [31] | C. Rolland y C. Proix, «A natural language approach for requirements engineering,» *In Advanced Information Systems Engineering,* vol. 593, p. 257–277, 2005. |
| [32] | K. Ryan, «The role of natural language in requirements engineering,» de *Proceedings of the IEEE International Symposium on Requirements Engineering*, San Diego, CA, USA, 1993. |
| [33] | R. J. A. a. D. K. Moorhead, «Software requirements and specifications: A survey of needs and languages,» *Journal of Systems and Software,* vol. 2, nº 4, pp. 297-316, 1981. |
| [34] | M. Luisa, F. Mariangela y N. I. Pierluigi, «Market research for requirements analysis using linguistic tools,» *Requirements Engineering,* vol. 9, p. 40–56, 2004. |
| [35] | M. Kassab, C. Neill y P. Laplante, «State of practice in requirements engineering: contemporary data,» *Innovations in Systems and Software Engineering,* vol. 10, p. 235–241, 2014. |
| [36] | J. H. Martín y D. Jurafski, «Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition,» *Prentice Hall,* 2000. |
| [37] | F. Dalpiaz, A. Ferrari, X. Franch y C. Palomares, «Natural Language Processing for Requirements Engineering,» *IEEE Software,* vol. 35, pp. 115-119, 2018. |
| [38] | W. Shira y B. Paul, «A Fully Automated Approach to Requirement Extraction from Design Documents,» de *IEEE Aerospace Conference (50100)*, Big Sky, MT, USA, 2021. |
| [39] | I. Hussain, L. Kosseim y O. Ormandjieva, «Using Linguistic Knowledge to Classify Nonfunctional Requirements in SRS documents,» *Lecture Notes in Computer Science,* vol. 5039, pp. 287-298, 2008. |
| [40] | C. Rolland y C. Salinesi, «Supporting Requirements Elicitation through Goal/Scenario Coupling,» *Conceptual Modeling: Foundations and Applications,* vol. 5600, p. 398–416, 2009. |
| [41] | S. Murugesh y A. Jaya, «Construction of Ontology for Software Requirements Elicitation.,» *Indian Journal of Science and Technology,* vol. 8, nº 29, 2015. |
| [42] | U. Shah, S. Patel y D. Jinwala, «Specification of non-functional requirements: A hybrid approach,» de *In 22nd International Working Conference on Requirements Engineering.* , Gothenburg, Sweden, 2016. |
| [43] | H. Meth, A. Maedche y M. Einoeder, «Is Knowledge Power? The Role of Knowledge in Automated Requirements Elicitation,» de *International Conference on Advanced Information Systems Engineering*, 2013. |
| [44] | Lili, «Research on User Requirements Elicitation Using Text Association Rule,» de *In International Symposium on Intelligence Information Processing and Trusted Computing.*, Huanggang, China, 2010. |
| [45] | G. Pablo y G. Marcos, «Dependency-Based Open Information Extraction,» de *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, Avignon, France, 2012. |
| [46] | A. C. M. A. Lorena Talamé, «Comparación de herramientas de procesamiento de textos en español extraídos de una red social para Python,» de *ASAI, Simposio Argentino de Inteligencia Artificial*, 2019. |
| [47] | S. Kübler, R. McDonald y J. Nivre, «Dependency Parsing. Synthesis Lectures on Human Language Technologies,» *Morgan & Claypool Publishers.,* p. 127, 2009. |
| [48] | H. Caseli, T. Pereira, L. Specia, T. Pardo, C. Gasperin y S. Aluisio, «Building a Brazilian Portuguese parallel corpus of original and simplified texts,» de *Center of Computational Linguistics*, Brasil, 2009. |
| [49] | J. Herrera, A. Peñas y F. Verdejo, «Textual Entailment Recognition Based on Dependency Analysis and WordNet,» *Part of the Lecture Notes in Computer Science book series,* vol. 3944, p. 231–239, 2005. |
| [50] | M. Ballesteros, R. Martín y B. D. Agudo, «JadaWeb: A CBR System for Cooking Recipes,» de *En Proceedings of Workshop on Computer Cooking Contest (ICCBR 2010)*, Italy, 2010. |
| [51] | V. Teller, « Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition,» 2000. |

1. https://stanfordnlp.github.io/CoreNLP/ [↑](#footnote-ref-1)
2. https://www.nltk.org/ [↑](#footnote-ref-2)
3. https://nlp.lsi.upc.edu/freeling/node/1 [↑](#footnote-ref-3)
4. https://www.python.org/ [↑](#footnote-ref-4)
5. https://www.jetbrains.com/help/pycharm/quick-start-guide.html [↑](#footnote-ref-5)
6. https://jupyter.org/ [↑](#footnote-ref-6)
7. https://es.acervolima.com/diferencia-entre-jupyter-y-pychar [↑](#footnote-ref-7)
8. https://spacy.io/ [↑](#footnote-ref-8)