UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 4

SIDARES: herramienta de procesamiento del lenguaje natural para la detección de ambigüedad léxica y sintáctica en requisitos de software

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

Autores:

Jonathan Ramírez Reyes

Samira de las Mercedes Enríquez González

Tutores:

Dr. C. Dunia María Colomé Cedeño

Dr. C. Héctor R. González Diez

M.Sc. Reiman Alfonso Azcuy

La Habana, noviembre 2022

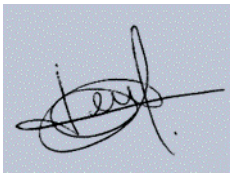
“Año 63 de la Revolución”

"El lenguaje ha de ser matemático, geométrico, escultórico. La idea ha de encajar exactamente en la frase, tan exactamente que no pueda quitarse nada de la frase sin quitar eso mismo de la idea".

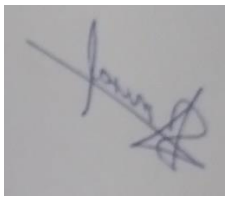
José Martí

Declaración de Autoría

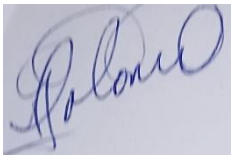
Declaramos ser autores de la presente tesis que tiene por título "SIDARES: herramienta de procesamiento del lenguaje natural para la detección de ambigüedad léxica y sintáctica en requisitos de software" y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo. Para que así conste firmamos la presente a los 16 días del mes de noviembre del año 2022.



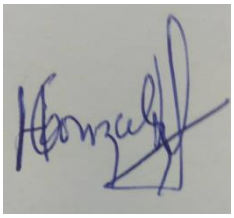
Jonathan Ramírez Reyes



Samira de las Mercedes Enríquez González



Dr. C. Dunia María Colomé Cedeño



Dr. C. Héctor R. González Diez

MS.c Reiman Alfonso Azcuy

Datos de Contacto

Nombre y apellidos del tutor: Dunia María Colomé Cedeño
Institución: Universidad de las Ciencias Informáticas
Título: Doctora en ciencias
Correo electrónico: dcolome@uci.cu

Nombre y apellidos del tutor: Héctor R. González Diez
Institución: Universidad de las Ciencias Informáticas
Título: Doctor en ciencias
Correo electrónico: hglez@uci.cu

Nombre y apellidos del tutor: Reiman Alfonso Azcuy
Institución: Universidad de las Ciencias Informáticas
Título: Master en ciencias
Correo electrónico: razcuy@uci.cu

Agradecimientos

A mis padres por su comprensión, cariño y apoyo incondicional, los amo.

A mi familia (abuelas, tíos, hermanos, primos) por su preocupación y el amor que me han dado.

A mis amistades y compañeros en especial a los amigos del aula que me apoyaron siempre.

A mi novio por hacerme feliz y ser mi gran amigo.

A mis tutores, profesores, mi oponente y profesora Yadira por apoyarme en todo este tiempo de desarrollo del trabajo de diploma y por ser comprensivos.

Muchas gracias a todos.

Samira

A mis padres por ser un ejemplo a seguir y hacerme una persona de bien.

A mi familia por quererme y apoyarme.

A mis amistades y compañeros en especial a los amigos del aula.

A mi novia Patricia por hacerme feliz.

A mis tutores por apoyarme en todo este tiempo de desarrollo del trabajo de diploma y por ser comprensivos.

Muchas gracias a todos.

Jonathan

Dedicatoria

A nuestros padres, familiares y amigos.

A nuestros tutores y profesores por confiar en nosotros y por todo el tiempo que nos dedicaron.

Samira y Jonathan.

Resumen

La ingeniería de requisitos es una de las etapas más importantes del ciclo de vida del desarrollo de software. El éxito de cualquier producto de software depende de la calidad de sus requisitos. Los requisitos de software suelen estar escritos en lenguaje natural. La ambigüedad en los requisitos escritos en lenguaje natural es un problema que ha sido estudiado por la comunidad de ingeniería de requisitos durante más de dos décadas. La resolución manual de la ambigüedad en los requisitos es trabajosa y requiere mucho tiempo. Existen varias herramientas de procesamiento del lenguaje natural para automatizar el análisis de la ambigüedad; sin embargo, la mayoría de ellas no están ampliamente disponibles, son obsoletas y poco seguras; las pocas herramientas públicas sólo permiten el análisis de requisitos en lengua inglesa. Esta investigación pretende desarrollar una herramienta de procesamiento del lenguaje natural para detectar la ambigüedad léxica y sintáctica presente en los requisitos de software, utilizando el lenguaje de programación Python y herramientas de procesamiento del lenguaje natural como NLTK. Como resultado de este trabajo, se presenta un conjunto de datos que contiene 19 357 requisitos pertenecientes a los proyectos de desarrollo de software de la Universidad de Ciencias Informáticas; el conjunto de datos obtenidos constituye una línea de base para futuras investigaciones. Se utilizó la metodología XP para guiar el desarrollo de la herramienta propuesta. Se evaluó el enfoque en un conjunto de datos de 100 requisitos, logrando un 98% de precisión y un 91% de exhaustividad.

Palabras clave: *ambigüedad; conjunto de datos; procesamiento del lenguaje natural; requisitos de software; técnicas.*

Abstract

Requirements engine of the most important stages of the software development life cycle. The success of any software product depends on the quality of its requirements. Software requirements are usually written in natural language. Ambiguity in requirements written in natural language is a problem that has been studied by the requirements engineering community for more than two decades. Manual resolution of ambiguity in requirements is tedious and time consuming. There are several natural language processing tools to automate ambiguity analysis, however, most of them are not widely available, outdated and unsafe; the few public tools only allow the analysis of requirements in the English language. This research aims to develop a natural language processing tool for detecting lexical and syntactic ambiguity present in software requirements, using the Python programming language and natural language processing tools such as NLTK. As a result of this work, a dataset is presented that contains 19,357 requirements belonging to the software development projects of the University of Informatics Sciences; the set of data obtained constitutes a baseline for future research. The XP methodology was used to guide the development of the proposed tool. We evaluated our approachon a suite of dataset of 100 requirements and achieved 98% precision and 91% recall on average.

Keywords: *ambiguity; dataset; natural language processing; software requirements; techniques.*

Índice de Contenido

Índice de Figuras

Índice de Tablas

Introducción

En un estudio reciente de Standish Group[1] se analizaron más de 50 000 proyectos y se descubrió que el 50% de ellos se cuestionaron, porque eran entregados tarde, archivados por encima del presupuesto o las características y funcionalidades no eran las requeridas por el cliente. Solo el 31% tuvo éxito y el 19% fallaron. Una de las principales causas de los fracasos es la inadecuada realización de la Ingeniería de Requisitos (IR).

Llevar a cabo de manera adecuada el proceso de IR disminuye la probabilidad de fracaso de un proyecto. Los requisitos bien definidos permiten conocer de un modo conciso lo que debe ser capaz de realizar el software a desarrollar además de orientar las actividades, recursos y esfuerzos de manera eficiente permitiendo la disminución de costos y retrasos. Dentro de la IR se realizan las tareas educción, documentación, validación y gestión de requisitos *(Aschauer et al. 2018)*. La obtención de requisitos es la actividad de extraer, de cualquier fuente de información disponible (documentos, aplicaciones existentes, entrevistas, etc.), las necesidades de los interesados para desarrollar el sistema. El proceso de captura de requisitos puede resultar complejo, debido a esto existen un conjunto de técnicas que permiten hacer este proceso de una forma más eficiente y precisa, obteniéndose necesidades y modelos del sistema. Dentro de las técnicas utilizadas para esta actividad se encuentran: las entrevistas, tormenta de ideas, los cuestionarios y mapa conceptual *(Durand 2017).*

La documentación es otra de las tareas de la IR. Documentar los requisitos tal y como lo describe el cliente, de modo que todas las partes interesadas comprendan el contexto exacto de los requisitos forma parte de su tarea fundamental *(Sabriye 2018)***.**

La especificación de requisitos es una de las actividades que se realiza dentro de la documentación. Se debe elaborar una especificación de calidad porque constituye la base del desarrollo de software y la misma trae consigo un software de calidad. Una mala especificación conlleva a una pérdida de tiempo o al fracaso del producto, esto ocurre porque la especificación de requisito generalmente se realiza en lenguaje natural (LN).

[1] https://hennyportman.wordpress.com/2021/01/06/review-standish-group-chaos-2020-beyond-infinity/

Debido a ello se hace probable la aparición de varios defectos como la ambigüedad en los requisitos. Para abordar el tema de la ambigüedad en los requisitos, primeramente, se debe conocer las definiciones de requisito de software que brindan las distintas fuentes.

"Declaración abstracta de alto nivel de un servicio que debe proporcionar el sistema o una restricción de este" (Sommerville et al.2011)

"Necesidad percibida por un interesado. Capacidad o propiedad que debe tener un sistema" (Durand 2017)

"Un requisito no es más que una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo" (Jane 2018)

Analizando las definiciones anteriores, un requisito es una descripción de una condición o capacidad que debe cumplir un sistema, ya sea derivada de una necesidad de usuario identificada o bien estipulada en un contrato estándar, especificación u otro documento formalmente impuesto al inicio del proceso.

Un requisito ambiguo es un requisito que tiene diferentes significados y se puede interpretar de múltiples formas dependiendo de la ambigüedad que posea. La ambigüedad es como una anomalía, pero también, como un fenómeno natural que penetra todo el lenguaje. Hay que relacionarla y a la vez diferenciarla de la vaguedad y del lenguaje metafórico. En todas estas manifestaciones existen multiplicidad de significados y a la vez la exigencia de seleccionar uno y prescindir de los restantes. La ambigüedad supone la existencia de dos o más significados excluyentes entre sí, pero bien definidos, mientras que la vaguedad se basa en la indeterminación del referente y el lenguaje metafórico establece una relación entre el plano real y el metafórico. *(Osama et al. 2020)*
Dentro de los tipos de ambigüedad está la lingüística, que se puede definir como la posibilidad de encontrar dos o más significados para una misma serie de estímulos grafémicas, palabras, enunciado o texto*(Osama et al. 2020).* Algunas de las ambigüedades lingüística que presentan los requisitos de software son ambigüedad pragmática, semántica y sintáctica ponderadas, polisemia *(Osama et al. 2020).*

Para la detección y reducción de la ambigüedad en la documentación de requisitos, se identificaron distintas soluciones que pueden mejorar la calidad de la especificación de requisitos. Estas soluciones están agrupadas en las seis categorías siguientes *(Hayman Oo et al. 2018)*

- Placas de caldera
- Lenguajes controlados
- Inspecciones
- Orientada a objeto y basada en UML
- Ontología
- Procesamiento del lenguaje natural (PLN)

La mayor parte de las herramientas automatizadas se basan en soluciones de ontología y PLN. Las soluciones basadas en ontologías son una conceptualización de un dominio que permite la definición de relaciones semánticas entre entidades y la inferencia de conocimiento a través del razonamiento. Esta solución se utiliza para la detección de ambigüedades léxicas, además de ser adecuada para la elicitación de requisitos y puede ayudar a reducir el esfuerzo *(Kumar et al. 2019).*

El desarrollo de aplicaciones o sistemas de PLN hoy en día es un gran reto para la humanidad debido a que los cómputos requieren de un habla en lenguaje de programación preciso o un LN claro y altamente estructurado. Sin embargo, los humanos están lejos de eso, porque el LN no es siempre preciso, sino todo lo contrario, es ambiguo y los niveles lingüísticos dependen de muchos factores que hacen que su estructura no sea siempre la adecuada.

La IR es una de las disciplinas de suma importancia dentro del desarrollo de software y detectar errores en etapas tempranas como esta, reduce el costo y esfuerzo. Además, en estas etapas es donde se documentan los requisitos en LN, que trae consigo posibles ambigüedades, inconsistencia e incluso especificaciones incompletas de los requisitos, importantes para el desarrollo de un producto de calidad según las necesidades del cliente o usuario final.

Estos defectos que se encuentran en la fase de documentación son difícilmente corregibles si pasan a etapas posteriores, para ello es importante el uso de técnicas de PLN capaces de detectar los requisitos dentro de un documento escrito en este lenguaje, para reducir inconsistencias y ambigüedades al redactar un documento de especificación de requisito *(Hayman Oo et al. 2018)*. El PLN forma parte de la inteligencia artificial (IA) y tiene como objetivo aplicar sistemas o métodos informáticos que faciliten la comunicación entre humanos y máquinas, logrando una comprensión de ambos más factible y eficaz *(Ferrari 2019).*

El desarrollo de software en Cuba va en crecimiento y en los últimos años se ha evidenciado su auge, pero aún existen pocos avances en herramientas y técnicas para la detección de ambigüedades en los requisitos de software. Es decir, no se cuenta en el contexto cubano con una herramienta que permita el entrenamiento, la prueba y validación de técnicas para eliminar la ambigüedad de requisitos, por lo que no hay una disminución de este problema en los requisitos expresados por los implicados de cualquier producto lo que conlleva al problema común de no satisfacer las necesidades del cliente luego del desarrollo de este. El desarrollo de software en Cuba y por tanto en la Universidad de las Ciencias Informáticas (UCI) presenta los siguientes inconvenientes.

- En la Dirección de Calidad de la UCI, el análisis de requisitos de software se realiza manualmente al no disponer de una herramienta para ello.
- Los conjuntos de datos de requisitos de software público están en idioma inglés y las técnicas de desambiguación para este lenguaje son completamente diferentes a las aplicadas en idioma español.
- Los datos expuestos en los conjuntos de datos disponibles están enfocados en proyectos específicos.
- Las técnicas del PLN son en su gran mayoría estadísticas por lo que necesitan grandes bases de datos de requisitos en LN para soportar el entrenamiento, prueba y validación de técnicas de desambiguación de requisitos.
- La mayoría de las herramientas para el análisis de requisitos existentes en el mundo no están disponibles públicamente.
- Las herramientas públicas solo realizan el análisis de requisitos en idioma inglés.

Debido a la situación anteriormente expuesta se define el siguiente **problema investigativo** ¿Cómo contribuir a la detección de ambigüedades en los requisitos de software en español escritos en lenguaje natural para favorecer el análisis de su calidad?

Para solucionar el problema anterior se define como **objeto de estudio** el procesamiento del lenguaje natural para la ingeniería de requisitos.

Se determinó como **objetivo general:** desarrollar una herramienta de procesamiento del lenguaje natural para la detección de ambigüedad léxica y sintáctica en requisitos de software escritos en idioma español.

El objetivo de estudio está enmarcado en el **campo de acción** del procesamiento del lenguaje natural asociado a la ambigüedad de requisitos de software escritos en español.

Para ello se proponen las siguientes **tareas investigativas:**

- Elaboración del diseño teórico metodológico de la investigación.
- Determinación del objetivo a cumplir con la aplicación de las técnicas de procesamiento del lenguaje natural.
- Determinación del flujo del proceso de procesamiento del lenguaje natural.
- Selección de los datos que van a ser utilizados para la detección de ambigüedad.
- Limpieza de los datos seleccionados.
- Construcción de un conjunto de datos como caso de estudio de la investigación.
- Construcción de algoritmos que contribuyan a la detección de ambigüedades.
- Diseño y elaboración de pruebas para la validación de los requisitos de la solución.

Para la realización de dichas tareas se tuvieron en cuenta **métodos científicos como:**

Método histórico lógico: para analizar a nivel nacional e internacional el uso de conjuntos de datos de requisitos y herramientas similares a la que se implementará.

Análisis documental: este método se utilizó para la selección de fuentes bibliográficas relevantes para el estudio. Los principales documentos analizados se obtuvieron de la búsqueda realizada en base de datos como ACM, IEEE, Springer y ScienceDirect. La

bibliografía recuperada de todas las fuentes es para obtener el conjunto de bibliografía necesaria que se utiliza para encontrar una solución al problema de la investigación.

Análisis síntesis: se utilizó con el objetivo de analizar la información y arribar a conclusiones relacionadas con la investigación. A partir de las bibliografías y literaturas estudiadas, se realizó una síntesis de los elementos significativos de la investigación.

Preguntas Científicas: durante el trayecto de la investigación se pretende dar respuesta a las siguientes preguntas de investigación.

¿Cuál es el estado del desarrollo de herramientas para el procesamiento del lenguaje natural para la ingeniería de requisitos?
¿Cuáles son los criterios de calidad de los requisitos de software?
¿Cuáles son los tipos de ambigüedades de requisitos?
¿Qué soluciones técnicas existen para la desambiguación de requisitos?
¿Cuáles son las características de los conjuntos de datos de requisitos de software?

El contenido de esta investigación se encuentra distribuido en tres capítulos de la siguiente manera.

Capítulo I. Procesamiento del lenguaje natural para IR: se describen un conjunto de conceptos relacionados con la problemática antes descrita. Además, se realiza un estudio de las herramientas informáticas y técnicas para la detección de la ambigüedad en los requisitos de software, así como las técnicas de PLN existentes. Se selecciona la metodología, las tecnologías y el lenguaje de programación utilizado en el desarrollo de la investigación.

Capítulo II. Análisis y diseño de la propuesta solución: se describen las características de la herramienta propuesta y el conjunto de datos construido. Así como su diseño y el proceso de construcción del conjunto de datos, utilizando como guía para el proceso de desarrollo la metodología XP. Se definen el patrón arquitectónico, el plan de iteraciones y de entrega. Se analizan los elementos necesarios para garantizar el éxito en el proceso de desarrollo, tales como: los patrones de diseño y estándares de codificación.

Capítulo III. Implementación, prueba y validación de la herramienta: se da continuidad a las fases codificación y prueba de la metodología XP, con la realización de pruebas unitarias, funcionales y de aceptación con el cliente a la herramienta propuesta utilizando para la validación el conjunto de datos de requisitos desarrollado. En el capítulo también se realiza una validación de los resultados de la herramienta en relación al cumplimiento del objetivo general de la investigación.

Capítulo I. Procesamiento del Lenguaje Natural para Ingeniería de Requisitos

Introducción

El LN en la IR tiene un importante papel, principalmente durante la descripción y especificación de requisitos del software; para entenderlo no se requiere una notación específica, lo que ha popularizado su uso. En este capítulo se presentan los principales conceptos y definiciones estudiados sobre esta temática, que resultaran de utilidad para una mejor comprensión de esta área del conocimiento. Además, se aborda el análisis de las tareas asociadas al PLN para la IR, se presenta un resumen de herramientas que emplean técnicas de PLN, un estudio de las principales tecnologías y métodos, que serán empleadas para desarrollar la propuesta solución. Por último, se presenta la metodología para llevar a cabo el desarrollo del software que contribuye a la desambiguación de requisitos y solución a la problemática expuesta.

1.1 Ingeniería de requisitos

El proceso de recopilar, analizar y verificar las necesidades del cliente o usuario para un sistema es llamado IR. La meta de la IR es crear, mantener y entregar una especificación de requisitos de software correcta y completa *(Sommerville et al. 2011)*. Por otra parte, varios autores *(Pressman 2020) (Aschauer et al. 2018)* definen la IR como las tareas y técnicas que llevan a entender los requisitos relevantes con un nivel de detalle adecuado en cualquier momento del desarrollo del sistema. La IR tiene diferentes actividades dependiendo de la literatura *(Sommerville et al. 2011) (Pressman 2020) (Aschauer et al. 2018)*, ver Figura 1.

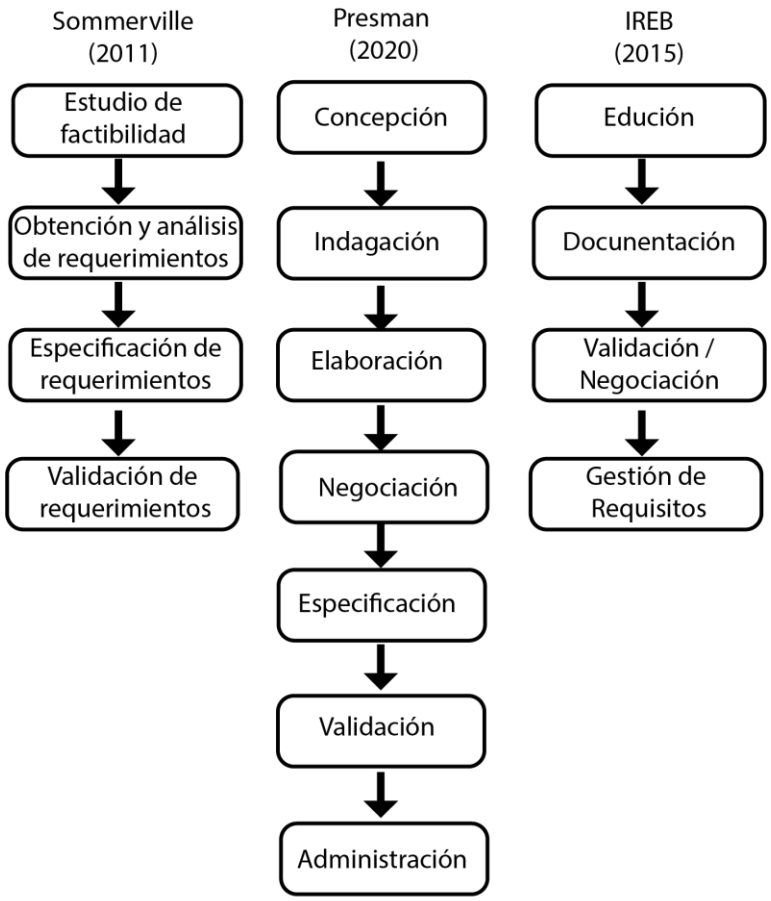


Figura 1: Actividades de la IR. Elaboración propia

Esta investigación se guía por las actividades propuesta por el Consejo Internacional de IR (IREB, por sus siglas en inglés), las que se detallan a continuación:

- **Educción:** es una de las actividades más cruciales en la IR, se recopilan los requisitos, se analizan y priorizan según las necesidades del cliente, el análisis incluye comprobar la existencia de ambigüedades o si se omitieron requisitos importantes para el cliente.

- **Documentación:** en esta fase se documentan los requisitos tal cual los describe el cliente, para que el resto de los implicados comprendan lo que se desea desarrollar. El documento debe ser claro e inequívoco para que sea considerado un documento de uso eficaz posteriormente.

- **Validación:** la validación se basa en asegurar el cumplimiento de las necesidades del cliente, para ello deben estar bien descritas, documentadas y completas.

- **Gestión de requisitos:** esta actividad está presente en cada fase del desarrollo de software, el seguimiento, planificación y análisis de los requisitos correctamente evita que el producto final se vea afectado.

Las actividades de educción y documentación son las más tratadas en esta investigación, teniendo en cuenta su objetivo.

1.1.1 Documentación de requisitos de software

La documentación, también reconocida como especificación, es una de las actividades fundamentales del proceso de IR, tal como se muestra en el epígrafe anterior. Esta actividad contempla tres tipos de documentación de requisitos: 1) *en LN*, 2) *con modelos conceptuales* tales como diagramas de casos de uso, diagramas de clase, diagramas de actividad y diagramas de estado y 3) *formas combinadas* de documentación de requisitos (LN y modelos).

En esta actividad, el documento de requisitos es relevante, pues en este se describen las características del producto, se brinda información sobre el contexto del sistema, las condiciones de aceptación y las características de la implementación técnica; este documento es reconocido como el de especificación de requisito del software *(Glinz et al. 2020)*. Una especificación es un documento que define, de forma completa, precisa y verificable, los requisitos, el diseño, el comportamiento u otras características de un sistema o componente de este *(Durand 2017)*.

Generalmente, en este tipo de documento se aprecian las tres formas de documentación, pero son los requisitos en LN del sistema en consideración, los elementos esenciales del documento. La especificación de requisitos de software produce artefactos como el diseño del sistema, codificación y prueba para el desarrollo del software y el resultado exitoso del proyecto. Estos documentos ayudan como vínculo en el inicio del desarrollo hasta el punto principal del control de calidad y como se mencionaba anteriormente el lenguaje recomendado para realizar la especificación es el natural *(Hayman Oo et al.2018)*.

El documento de especificación de requisitos y los requisitos de software comparten criterios de calidad, como son: la claridad, la comprensibilidad y la ausencia de ambigüedad *(Gnesi et al. 2019)*. La documentación de requisitos de software de calidad

constituye un reto, teniendo en cuenta que, esencialmente su especificación se realiza empleando el LN *(Osama et al.2020)*, que, a pesar de ser expresivo, intuitivo y universal, a la vez puede resultar ambiguo. Diversas alternativas se han desarrollado para la descripción de los requisitos *(Durand 2017)*, pero ninguna se ha generalizado, por lo que el LN continúa siendo la forma más empleada para especificar los requisitos del software. *(Sommerville et al. 2011)*

La calidad de un producto de software y del documento de especificación de requisitos dependen en gran medida de la calidad de la especificación de sus requisitos. En el siguiente epígrafe se presentan los principales criterios de calidad de los requisitos, así como aquellos elementos del lenguaje a tener en cuenta para su especificación en LN.

1.2 Criterios de calidad de requisitos de software

La evaluación de la calidad de requisitos en LN sigue un modelo de calidad específico. Los modelos de calidad son la base de las herramientas automatizadas para el análisis de requisitos de software. *(Gnesi et al.2019)*. Un modelo de calidad es una guía de referencia con la que se puede evaluar un determinado artefacto; se define mediante un conjunto de atributos de calidad, que son propiedades de calidad de alto nivel. *(Ferrari et al 2019)*

En las distintas bibliografías se aprecian los criterios de calidad utilizados en herramientas analizadoras de requisitos. Algunas de estas herramientas *(Kumar Gupta et al. 2019)*, *(Gnesi et al. 2019*, *(Ferrari et al.2019), (Kumar Gupta et al. 2019)* siguen diferentes normas como la ISO/IEC/IEEE 29148:2018 e ISO/IEC/IEEE152888:2015, mientras que pocas optan por la comprobación de errores gramaticales. Los criterios de calidad y las características de los requisitos varían; existiendo herramientas de PLN para la IR que pretenden mejorar la calidad de los requisitos centrándose en una o más de esas características, tanto para la calidad del requisito individualmente como de un conjunto de estos.

Entre los criterios de calidad abordados por diversos autores para la evaluación de requisitos, se encuentran: ***claridad*** *(Gnesi et al. 2019), (Naeem, et al. 2019), (Ferrari, et al. 2019),* ***correcto*** *(ISO/IEC/IEEE29148:2018), (Ferrari, et al. 2019), (Naeem, et al. 2019), (Kumar, et al. 2019)*, ***no ambigüedad*** *(ISO/IEC/IEEE29148:2018), (Gnesi et al.*

2019), (Ferrari, et al. 2019), (Arendse et al. 2016), (Kumar, et al. 2019), ***completitud*** *(ISO/IEC/IEEE 29148:2018), (Kumar, et al. 2019), (Ferrari, et al. 2019),* ***singular*** *(ISO/IEC/IEEE 29148:2018), (Ferrari, et al. 2019)* y otras como: ***atomicidad, factible, verificable, apropiado, comprensible*** **y** ***coherente****.* Ver Tabla 1

Tabla 1: Criterios de calidad de los requisitos de software. Elaboración propia

Autores	**Criterio de calidad**
ISO/IEC/IEEE29148:2018	necesario, adecuado, sin ambigüedades, completo, singular, factible, verificable, correcto, conforme
(Gnesi et al. 2019)	no ambigüedad, claridad, comprensibilidad
(Ferrari, et al. 2019)	claridad, no ambigüedad, simplicidad, exhaustividad, concisión, corrección, coherencia
(Naeem, et al. 2019)	claridad, correcto
(Kumar, et al. 2019)	completo, correcto, no ambigüedad
(Arendse et al. 2016)	ambigüedad, atomicidad

Además, el estándar ISO/IEC/IEEE29148:2018 reconoce como criterios de calidad para un conjunto de requisitos los siguientes: completo, consistente, factible, comprensible, capaz de ser validado. El criterio de no ambigüedad es de interés significativo en los requisitos de software, razón por la que es el criterio que se aborda en esta investigación.

1.2.1 Ambigüedad de requisitos de software

En la literatura consultada se aprecian diversas definiciones que aportan un mejor entendimiento de estos criterios de calidad de requisitos de software. Seguidamente se presentan algunas definiciones de ambigüedad.

> *"Ambigüedad es el término que hace referencia a aquellas estructuras gramaticales que pueden entenderse de varios modos o admitir distintas interpretaciones y dar, por consiguiente, motivo a dudas, incertidumbre o confusión" (Ortuño et al. 2016).*

> *"La ambigüedad consiste en los múltiples significados que tiene una palabra, tal como puede quedar reflejado en un diccionario; la multiplicidad de significados se llama polisemia" (Yadav et al. 2021).*

En esta investigación se asume que la ambigüedad es la posibilidad de admitir diferentes interpretaciones a partir de la representación de una oración; está presente cuando existe confusión al tener diversas estructuras asociadas a la misma oración y no tener los elementos necesarios para eliminar las incorrectas *(Ortuño et al. 2016).* Existen distintos tipos de ambigüedades, las que han sido tratadas por variados autores. *(Bustos 2018) (Yadav, et al. 2021)*, *(Zhao et al. 2021)*. A continuación, en la Figura 2 se muestran los tipos de ambigüedades más relevantes que pueden estar presente en los requisitos de software.

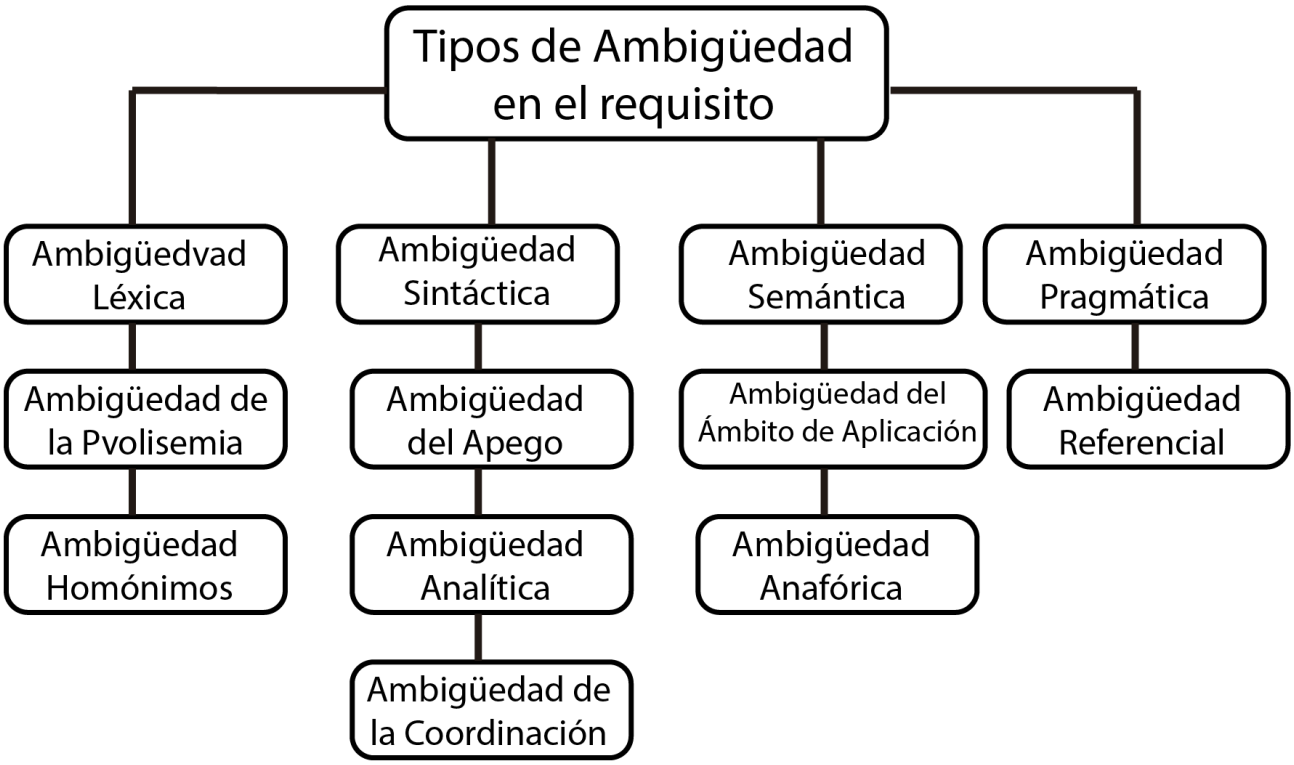


Figura 2:Tipos de ambigüedad. Fuente: (Zait et al 2018) (Osama et al. 2020)

En el Anexo 1 se encuentra las definiciones y ejemplos de los tipos de ambigüedad que se muestran en la Figura 2. En este trabajo se tratan las ambigüedades sintáctica y léxica; ambigüedades como la semántica y la pragmática, que requieren un nivel más profundo de análisis del significado al nivel de discurso y contexto; serán incorporadas a la

propuesta de solución en trabajos futuros. A continuación, se detallan las ambigüedades léxica y sintáctica.

La **ambigüedad léxica** se refiere a las palabras o frases que tienden a tener más de un significado en dependencia del contexto, como es el caso de cubo-Figura geométrico y cubo-recipiente. Este tipo de ambigüedad puede clasificarse en homónima y polisémica. La homonimia se produce cuando varias palabras tienen la misma ortografía o fonética, pero significados diferentes. Por ejemplo, lo que sucede con llama-fuego y llama-animal, que derivan del latín. Por otro lado, la polisemia se produce cuando una misma palabra da diferentes significados en distintos contextos. Esto se evidencia con la palabra periódico puede significar 1) publicación física impresa "el periódico se mojó con la lluvia", 2) la ilustración editora "el periódico firmo a su personal de edición *(Osama et al. 2020) (Zhao et al.2022).*

Mientras la **ambigüedad sintáctica** se manifiesta cuando una expresión o frase tiene más de una estructura gramatical. Por ejemplo, tomo un autobús rápido, esta frase tiene más de una salida porque hace referencia a tomo el autobús rápidamente o el autobús que tomo es muy rápido. Existen tres tipos de ambigüedad: sintáctica coordinativa, de apego y analítica *(Osama et al. 2020*). La coordinativa *(López 2019)* se puede presentar cuando una oración contiene más de una palabra de tipo conjunción; puede ser copulativa, disyuntiva o mixta. La ambigüedad de apego ocurre cuando existe la duda de la unión de una frase con otra. Por último, la ambigüedad analítica, se produce cuando el papel de los constituyentes dentro de una oración o frase es ambiguo, esto es común en las oraciones sustantivas compuestas *(Osama et al. 2020).*

Además de los criterios de calidad, diversos autores se refieren a criterios del lenguaje de requisitos ver Anexo 2. Estos criterios constituyen consideraciones a tener en cuenta durante la escritura de los requisitos para obtener requisitos bien formados. En el próximo epígrafe se presentan los criterios del lenguaje de requisitos más relevantes.

1.2.2 Criterios del lenguaje de requisitos de software

Como se mencionó en epígrafes anteriores algunas herramientas hacen uso de los criterios de calidad o características de calidad y del lenguaje de requisitos. A

continuación, queda resumido el estudio bibliográfico realizado sobre los criterios del lenguaje utilizados por distintas herramientas de PLN para la IR.

Entre los criterios de lenguaje resaltan el uso de ***superlativos*** *(ISO/IEC/IEEE 29148:2018), (Arendse et al. 2016),* ***términos subjetivos*** *(ISO/IEC/IEEE 29148:2018), (Gnesi et al. 2019), (Arendse et al.2016), (Ferrari, et al. 2019),* ***pronombres vagos*** *(ISO/IEC/IEEE 29148:2018), (Gnesi et al., 2019), (Arendse et al. 2016), (Naeem, et al. 2019),* ***frases o términos comparativos*** *(ISO/IEC/IEEE 29148:2018), (Arendse et al, 2016),* ***imperativos*** (*Gnesi et al.2019), (Arendse et al.2016), (Naeem, et al. 2019),* ***opcional*** *(Gnesi et al. 2019), (Naeem, et al. 2019), (Ferrari, et al. 2019),* ***cuantificadores*** *(Arendse et al. 2016), (Naeem, et al. 2019),* ***términos ambiguos*** *(ISO/IEC/IEEE 29148:2018), (Arendse et al.2016),* ***términos negativos*** *(Arendse et al., 2016), (Naeem, et al. 2019),* entre otros. En el Anexo 2 se presentan estos y otros criterios del lenguaje de requisitos de software.

Basado en el significado de las ambigüedades léxica y sintáctica y en sus relaciones con varios de los criterios del lenguaje de requisitos, presentados anteriormente, se ha definido el siguiente modelo de calidad para la evaluación de los requisitos en LN, el cual será el empleado en la herramienta a implementar.

Criterio de calidad: No ambigüedad.

Indicadores según el tipo de ambigüedad:

Ambigüedad léxica aparece cuando una palabra presenta diferentes significados.

- El uso de frases múltiples provocará diversas interpretaciones.
- No debe haber términos ni frases con múltiples interpretaciones.
- Palabras o frases con más de un significado que provoque confusión.

Ambigüedad sintáctica aparece cuando una frase u oración se le asocia más de una estructura gramatical.

- No deben aparecer conjunciones disyuntivas (o, u, sea, bien) y copulativas (y, e, ni, que).
- No deben existir preposiciones separables (a, con, de, en).

Estos indicadores serán utilizados para desarrollar las reglas heurísticas en la que se basarán los algoritmos de detección de ambigüedad léxica y sintáctica.

1.3 Procesamiento del lenguaje natural para la ingeniería de requisitos

Procesamiento en lenguaje natural

El PLN es una tecnología que ha ido evolucionando en los últimos años. El campo de acción se encuentra dentro de la IA y la lingüística, para interpretar el LN en la relación de los seres humanos y las máquinas. A continuación, se presentan algunas definiciones sobre esta tecnología.

> *"Este campo de estudio trata sobre cómo las máquinas/ordenadores procesan las órdenes del LN, ya sean escritas o verbales, y luego realizan la acción solicitada. La salida debe ser a través de la respuesta escrita o de voz, en el LN" (Paul et al. 2018).*

> *"El PLN es una gama de técnicas computacionales teóricamente motivadas para analizar y representar textos naturales en uno o más niveles de análisis lingüístico con el propósito de lograr un procesamiento del lenguaje similar al humano para una variedad de tareas o aplicaciones" (Guadalupe 2019).*

> *"El PLN consiste en comprender e interpretar el lenguaje humano, hablado o escrito, mediante el procesamiento de máquinas" (Guadalupe 2019).*

> *"La capacidad de que las máquinas entiendan e interpreten a los humanos se denomina PLN" (Guadalupe 2019).*

Como se puede apreciar existen diferentes definiciones para el PLN dependiendo del autor o contexto. Con relación a la problemática que da origen a esta investigación, se decidió tomar en todo el trayecto de esta, la primera definición. La razón de esta decisión es que el PLN se esfuerza en conseguir un procesamiento del lenguaje similar al humano, pero más que eso los autores de la presente investigación pretenden apoyar a los ingenieros de requisitos a realizar la tarea de detección de defectos, que implican el procesamiento y el análisis de documentos textuales de requisitos.

Existen niveles de análisis lingüísticos como los fonéticos, morfológico, léxico, sintáctico, semántico, discursivo y pragmático del lenguaje a los que se hace alusión en la definición dos *(Guadalupe 2019)*. Se conoce además que los humanos suelen usar esos niveles para comprender el lenguaje. Por tanto, los sistemas o máquinas pueden admitir, combinar o manejar los niveles y mientras más adquieran más capaces y potentes son estos sistemas o cómputos.

En la Figura 3 puede apreciarse la relación entre técnicas, tareas, recursos y herramientas del PLN, presentada por *(Zhao et al. 2022).* Las tareas se pueden automatizar, las técnicas basadas en PLN apoyan la ejecución de las tareas y las herramientas emplean las técnicas, mientras que los recursos emplean las tareas y las herramientas.

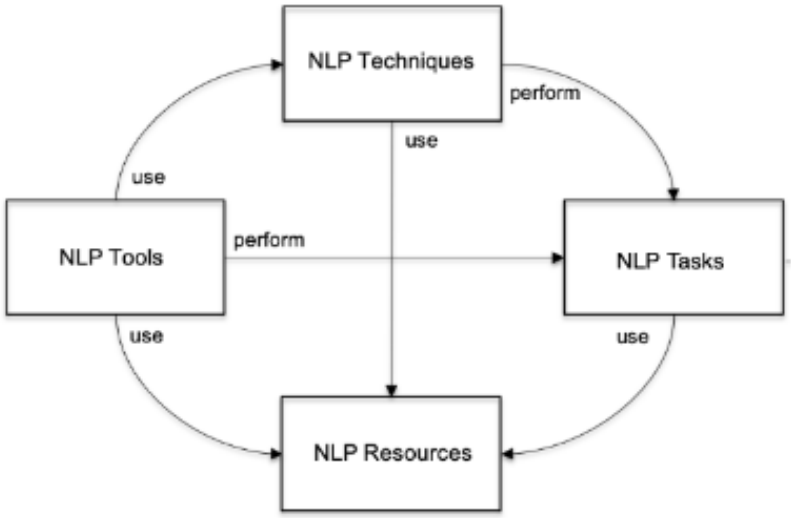


Figura 3: Relación entre técnicas y tareas de PLN. Fuente Zhao et al.2022)

Como se puede apreciar en la Figura 3 las herramientas PLN emplean técnicas y recursos, a su vez permiten el desarrollo de tareas de PLN. Las técnicas PLN emplean recursos y permiten desarrollar tareas, mientras que las tareas utilizan recursos.

Procesamiento del lenguaje natural para ingeniería de requisitos

La relación entre el PLN y la IR está establecida y ampliamente discutida *(Zhao et al 2022)*. El PLN para la IR (PLN4IR) o el NLP4RE (por sus siglas en inglés) es un área de investigación y desarrollo que busca aplicar tecnologías de PLN (técnicas, herramientas y recursos) a diferentes tipos de documentos de requisitos para respaldar una variedad de tareas de análisis lingüístico realizadas en varias etapas de la IR. *(Guzmán-Luna et al. 2018)*

Se conoce que los primeros trabajos sobre la aplicación de técnicas de PLN a tareas de IR comenzaron en los años noventa, con valiosas experiencias orientadas principalmente a la síntesis de modelos *(Ferrari 2019).* Luego de un período de experimentación de las técnicas en entornos de investigación para diferentes tareas, en los últimos años se ha observado un creciente número de estudios de casos sólidos sobre aplicaciones de PLN a problemas de IR del mundo real. La madurez de las herramientas actuales de PLN ha contribuido a este gran avance *(Zhao et al. 2021)*.

1.3.1 Tareas del procesamiento del lenguaje natural para la ingeniería de requisitos

Una tarea de PLN se define como "*un trabajo de procesamiento de texto que se puede realizar mediante una o más técnicas de PLN, con el apoyo de algunas herramientas y recursos de PLN*" *(Zhao et al. 2022)*. Mientras que, las tareas del PLN para IR son tareas de análisis lingüístico llevadas a cabo durante las actividades de la IR *(Guzmán-Luna et al. 2018)***.** En (Zhao et al. 2021) presentan seis tipos de tareas de PLN para IR, con un enfoque orientado a la IR, las que se presentan en la Tabla 2

Tabla 2: Tareas de PLN4IR (Zhao et al. 2022)

Tarea PLN4IR	Significado	Explicación
Detección *(Tjong et al. 2018) (Falessi et al. 2018)*	Detectar problemas lingüísticos en los documentos de requisitos	Esta tarea suele servir de apoyo a las actividades de revisión manual para que los requisitos, o los artefactos relacionados con ellos, sean claros e inequívocos.
Extracción *(Zhao et al. 2022)*	Identificar abstracciones y conceptos clave del	Esta tarea normalmente tiene como objetivo extraer términos de una o varias palabras de los textos de requisitos para establecer glosarios específicos del

	dominio	dominio y del proyecto.
Clasificación ***(Casamayor, et al. 2018)***	Clasificar los requisitos en diferentes categorías	Esta tarea tiene como objetivo clasificar los requisitos en diferentes tipos, basándose en el propósito para el que se aplica la tarea.
Modelado ***(Ambriola et al. 2018)***	Identificar los conceptos de modelado y construir modelos conceptuales	Esta tarea suele hacer uso de la tarea de extracción, y puede tomar diferentes formas de generación de modelos.
Rastreo y relación ***(Zhao et al. 2022)***	Establecer vínculos de trazabilidad o relaciones entre los requisitos, o entre los requisitos y otros artefactos de software.	Esta tarea tiene como objetivo principal apoyar las actividades de rastreo manual orientadas a hacer cumplir y demostrar la coherencia del proceso.
Búsqueda y recuperación (Morales-Ramirez, Kifetew, y Perini 2019) (Zhao et al. 2022)	Buscar y recuperar requisitos de los repositorios existentes	El objetivo de esta tarea es reutilizar los activos de requisitos existentes para que coincidan con las necesidades de nuevos clientes, así como apoyar el alcance del dominio hacia el desarrollo de un nuevo producto.

Cada una de las tareas mencionadas tienen un objetivo específico, que aporta facilidad a la hora de dar cumplimiento a distintas actividades de la IR. Para resolver la problemática presente en esta investigación se decidió profundizar en la tarea de **detección**.

Además de las tareas mencionadas en la tabla 2, en *(Zhao et al. 2022)* se listan otras tareas, en este caso con enfoque marcado hacia el PLN, que son frecuentemente ejecutadas en IR, y que se presentan a continuación.

- **Análisis sintáctico:** Analizar la estructura sintáctica de una frase para representar la relación entre sus componentes. Se pueden utilizar diferentes estructuras de representación, como el árbol de análisis sintáctico, o el gráfico de análisis de dependencias.
- **Análisis semántico:** Identificar y etiquetar componentes y relaciones semánticamente relevantes en el texto. Supone identificar el significado de una determinada palabra o frase en un contexto y la relación entre palabras o términos.
- **Análisis de frecuencias:** Analizar las frecuencias de palabras o términos en un determinado contexto y producir datos probabilísticos.
- **Análisis de similitud:** Para calcular las estimaciones numéricas de la similitud entre los elementos del texto, (ej. para identificar la relación semántica, los sinónimos o para apoyar el tema modelado).
- **Análisis basado en reglas:** Utilizar reglas gramaticales, reglas semánticas o patrones para analizar la sintaxis de un texto.
- **Normalización del texto:** Convertir las palabras en su forma original y eliminar las palabras o caracteres innecesarios del texto.
- **Segmentación de textos:** Descomponer un texto en una secuencia de frases o palabras individuales.
- **Etiquetado de parte del discurso:** Asociar palabras con etiquetas de parte del discurso para distinguir entre sustantivos, verbos, adjetivos, adverbios, etc. La unidad de entrada es una frase, ya que las palabras del contexto (es decir, las vecinas) se utilizan normalmente para inferir el etiquetado de una palabra.
- **Etiquetado semántico:** Extraer del texto fragmentos de información útiles (palabras, términos, relaciones, etc.).

En la solución a la problemática expresada en este trabajo, se decide la realización de las tareas **segmentación y normalización del texto**, **etiquetado semántico** y **de parte del discurso**, además de los **análisis sintáctico** y **basado en reglas**. Como se mencionaba las tareas de PLN se realizan a través de técnicas de PLN, en el siguiente epígrafe se presentan las principales técnicas de PLN.

1.3.2 Técnicas de procesamiento del lenguaje natural para la ingeniería de requisito

Una técnica de PLN es "*un método práctico, un enfoque, un proceso o un procedimiento para realizar una tarea de PLN concreta" (Zhao et al.2022)*, *(Zhao et al. 2021)*. Las tareas de PLN tienen una estrecha relación con las técnicas de PLN, en el Anexo 3 se puede evidenciar esta relación. A continuación, se describen en la Tabla 3, algunas de las técnicas de PLN más recientes, publicadas en los últimos dos años, utilizadas con frecuencia en la IR *(Zhao et al.2022)*.

Tabla 3: Técnicas de PLN más empleadas en la IR (Zhao et al.2022).

Nombre	Explicación
Etiquetado de parte del discurso (clasificación)	El etiquetado de parte del discurso procesa una secuencia de palabras, y adjunta una etiqueta a cada palabra. Las partes de la oración también se conocen como clases de palabras o categorías léxicas.
Chunking	Es un tipo de análisis sintáctico superficial que analiza una frase, identificando primero sus partes constituyentes (sustantivos, verbos, adjetivos, etc.) y luego los relaciona con unidades de orden superior que tienen significados gramaticales discretos (grupos de sustantivos o frases, grupos de verbos, etc.).
Análisis del sentimiento	Es el proceso de identificar y categorizar computacionalmente las opiniones expresadas en un texto.
Indexación semántica latente	Una práctica matemática que ayuda a clasificar y recuperar información sobre determinados términos clave y conceptos utilizando la descomposición del valor singular.
Patrones semánticos	Los patrones semánticos se generan a partir de los conceptos comunes que coinciden. Los conceptos más coincidentes de cada palabra, son considerados. Un patrón semántico puede

	relacionarse con varios conceptos y una sola estructura semántica puede contener varios patrones semánticos.
Patrones léxicos	Palabras o fragmentos de texto que aparecen en el lenguaje con alta frecuencia y el significado de las partes del texto o la palabra son a veces diferentes del significado del texto o la palabra en su conjunto.
Detección de homónimos	Detectar las palabras que se pronuncian igual entre sí (por ejemplo, "caza" y "casa") o tienen la misma grafía (por ejemplo, "llama de fuego" y "llama de animal")
Análisis basado en reglas	Utiliza reglas gramaticales, reglas semánticas o patrones para analizar la sintaxis de un texto.
Normalización del texto	Convertir las palabras en su forma original y eliminar las palabras o caracteres innecesarios del texto.
Segmentación de textos	Descomponer un texto en una secuencia de frases o palabras individuales.
Lematización	Utilizar un análisis de vocabulario y morfológico de las palabras, normalmente con el objetivo de eliminar las inflexiones y devolver la forma base o de diccionario de una palabra, que se conoce como lema.
Eliminación del ruido	Eliminación de caracteres, dígitos y trozos de texto que puedan interferir en el análisis del texto.
Tokenización	El proceso de dividir una cadena de texto en palabras, frases, símbolos u otros tokens significativos.
Frecuencia de las palabras	La frecuencia con la que aparece una palabra en un documento, dividida por el número de palabras que tiene.

Teniendo en cuenta el análisis de las características y el uso de las técnicas descritas para la IR en la tabla 3, se decide utilizar en la implementación de los algoritmos de detección de ambigüedades léxicas y sintácticas, la **tokenización**, la **eliminación del ruido** y el **etiquetado de parte del discurso**. El uso de estas técnicas es muy común en el PLN para el preprocesamiento. La tokenización es la técnica más importante y obligatoria a la hora del tratamiento de texto (Ramírez 2018) (Ricardo Javier. et al. 2021), porque separa los fragmentos de texto en pequeñas unidades. Esto permite un

procesamiento y limpiado del texto más eficiente. La eliminación de ruido permite la limpieza adecuada del texto de gran ayuda para dar solución a la problemática planteada. El objetivo del etiquetado de parte del discurso es asignar una clasificación a cada token como se describe en la tabla anterior. Una clasificación para un token puede ser por ejemplo sustantivo o verbo. Dentro de los indicadores de calidad sintácticos se encuentran la no existencia de conjunciones ni preposiciones por eso esta técnica sería de gran utilidad.

1.3.3 Herramientas de procesamiento del lenguaje natural

Una herramienta de PLN es “un sistema de software o una librería de software que apoya una o más técnicas de PLN” *(Guzmán-Luna et al. 2018)*. A continuación, se presentan las principales herramientas empleadas para el PLN.

- **Standford CoreNLP:** grupo de herramientas de análisis de lenguaje natural escritas en Java. Es un marco integrado lo que hace que sea muy fácil aplicar a un montón de herramientas de análisis de texto. Utiliza componentes de aprendizaje profundo, probabilístico y basado en reglas. Permite la clasificación, normalización y tokenización**.** *(Yang et al. 2022)*

- **Stanford NLP:** es un paquete de análisis del lenguaje natural en Python. Contiene herramientas que se pueden utilizar en una línea de producción para convertir una cadena que contenga texto en lenguaje humano en listas de frases y palabras. Además, se utiliza para generar formas base de palabras, oraciones y características morfológicas, para dar un análisis de dependencia de la estructura sintáctica. Está diseñado para ser paralelo entre más de 70 idiomas. Este paquete está construido con componentes de redes neuronales de alta precisión que permiten un entrenamiento y una evaluación eficientes con sus propios datos anotados. Los módulos están construidos sobre PyTorch. Algunas de las características de este paquete de análisis es que requiere un esfuerzo mínimo de configuración. Contiene una red neuronal completa para el análisis de texto robusto, incluyendo la tokenización y el análisis de dependencias. Además, presenta una interfaz de Python estable. *(StanfordNLP 2022),* (Peng Qi et.al 2018)

- **GATE:** es un marco de procesamiento de texto y procesamiento del lenguaje natural implementado en Python. Proporciona representaciones muy flexibles de documentos, anotaciones separadas por características. Brinda la capacidad de usar las herramientas de PLN existentes para representar documentos listos para usarse. Además, ofrece sus propias herramientas de anotación como diccionarios basados en cadenas y tokens, anotadores basados en expresiones regulares y reglas. Permite opcionalmente el seguimiento de todos los cambios realizados en el documento. *(Robert 2022)*

- **NLTK:** es una plataforma para crear programas de Python que trabaja con datos en lenguaje natural. Tiene un diseño modular que la hace ideal para adaptarse a distintas problemáticas por medio de la composición de módulos. NLTK ha ayudado a resolver varios problemas asociados a diferentes aspectos del PLN. Proporciona interfaces fáciles de usar, junto a un conjunto de bibliotecas de procesamiento de texto para la clasificación, tokenización, lemmatización y análisis semántico. Es un proyecto gratuito de código abierto. *(Samuel 2019)*

- **WEKA:** es una plataforma de software destinada al aprendizaje automático y la minería de datos escrita en Java. Weka contiene herramientas para diferentes tareas como reproceso, clasificar, asociación y visualizar. la herramienta puede ser utilizada en cualquier maquina con la máquina virtual de java. Una de las características del Weka es que tiene implementados algoritmos de clasificación, agrupamiento y reglas de asociación. Además, es de código abierto bajo licencia pública general y brinda como resultado un desarrollo más rápido de modelos de aprendizaje automático. (Espinoza 2018)

- **Apache OpenNLP:** conjunto de herramientas basado en aprendizaje automático para el procesamiento de texto en lenguaje natural. Está escrito totalmente en Java y brinda soporte para tareas comunes de NLP, como tokenización, etiquetado de parte del discurso, fragmentación entre otras. Proporciona un grupo de modelos prediseñados para una variedad de idiomas y clasificadores comunes como Entropy, Perceptron y Naive bayes. *(Reese et al. 2018)*

- **Spacy:** herramienta que permite construir aplicaciones de procesamiento de lenguaje natural (NLP). Proporciona modelos pre entrenados de diferentes lenguajes, lo cual es ideal para los principiantes en el campo del PLN. Además, permite crear modelos nuevos y modificar los pre entrenados con datos propios a campos específicos. Su velocidad y precisión es muy eficiente pese a utilizar modelos de redes neuronales. Permite extraer y analizar información de un texto. *(Duygu 2021)*

- **Scikit-learn:** herramienta simple y eficiente para el análisis de datos. Está escrita en Python y basada en Numpy, Scipy y matplotlib. Es de código abierto accesible y reutilizable en varios contextos. Proporciona herramientas para el aprendizaje automático y el modelado estadístico incluida la clasificación, regresión, agrupamiento y reducción de la dimensionalidad. Ofrece además una documentación que posibilita una mayor flexibilidad del uso de la herramienta. *(Jolly 2018)*

Tabla 4:Análisis comparativo de herramientas PLN

Herramienta	**Funcionalidades**	**Lenguaje /Disponibilidad**
Standford CoreNLP	Clasificación Normalización Tokenización	Java
Stanford NLP	Análisis de texto robusto incluyendo la tokenización Análisis de dependencias	Python
GATE	Diccionarios basados en cadenas y tokens Anotadores basados en expresiones regulares y reglas	Python
NLTK	Clasificación Tokenización Lemmatización Análisis semántico	Python / código libre

WEKA	Clasificación Agrupamiento Reglas de asociación	Java/ código libre
Apache OpenNLP	Tokenización Etiquetado de parte del discurso Fragmentación	Java
Spacy	Extraer y analizar información de un texto	Python
Scikit-learn	Clasificación Regresión Agrupamiento Reducción de la dimensionalidad	Python/ código libre

Luego del estudio de herramientas de PLN realizado se decide utilizar para el desarrollo de la solución NLTK debido a que es de código libre y escrita en Python ver tabla 4. Además, esta es una de las herramientas usadas para el PLN por las funcionalidades que brinda. Las bibliotecas que ofrece NLTK para realizar la tokenización, clasificación y lematización hacen de esta herramienta la más adecuada; para lograr un procesamiento y análisis profundo de los requisitos de software y a su vez en ellos detectar defectos.

1.4 Herramientas de procesamiento del lenguaje natural para la ingeniería de requisitos

Las herramientas de PLN para la validación de requisitos, no reemplazan el razonamiento humano, pero si disminuyen el esfuerzo y el tiempo de los ingenieros de requisitos en el desarrollo de esta actividad. De esta manera el ingeniero de requisito puede actuar, a partir de los comentarios o recomendaciones derivadas de estas herramientas y reescribir los requisitos de manera adecuada.

En los últimos 20 años, más de 50 herramientas han sido desarrolladas, ya sean automatizadas o semiautomatizadas, que emplean técnicas de PLN para mejorar la calidad de los requisitos (Naeem et.al. 2019). Existen cuatro tipos de clasificación de los

enfoques de las herramientas de PLN para la IR (Soren Lauesen 2020), los cuales se mencionan a continuación:

1. Resaltar los defectos y desviaciones en el documento de requisitos escrito en lenguaje natural.
2. Generación de modelos a partir de la descripción del lenguaje natural.
3. Determinar los vínculos de rastreo entre la descripción de lenguaje natural.
4. Encontrar las abstracciones clave de los documentos en lenguaje natural.

Muchas de estas herramientas están obsoletas, entre ellas SHREE, QUARS, SAT y AQUSA (Naeem et.al. 2019), otras no están disponibles. En la Tabla 5 se muestra un resumen de las herramientas de PLN para IR propuesto por Afrah Naeen, Zeeshan Aslam y Munam Ali Shah (Naeem et.al. 2019), el cual se ha tomado como base para la selección de las herramientas a analizar según el objetivo de este trabajo.

Tabla 5: Resumen de herramientas de PLN para IR (Naeem et.al. 2019)

Nombre de la herramienta	**Año**	**Disponibilidad**
Qvscribe	2016	Licenciada
Innoslate	2014	Libre
QUOD	2014	Libre
RQA	2011	Licenciada
Requirement Assesment tool (RAT)	2018	Libre
TACTILE check	2017	Libre
Tiger Pro	2004	Libre para estudiantes
QUALICEN	2014	Licenciada
Visure quality analyzer	2011	De pago
NASA reconstructed ARM	2011	No disponible
Pragmatic Ambiguity Detector	2012	Libre
IntelliReq	2014	Bloqueada
NARCIA	2015	Instalable en la PC
AQUSA	2016	Libre solo para su comunidad

Teniendo en cuenta la información de la Tabla 5, han sido seleccionadas las siguientes herramientas para el análisis: QVscribe, RAT, RQA y QUOD.

QVscribe: es la herramienta existente que se utiliza con Microsoft Word para analizar los requisitos en tiempo de ejecución y comprobar su calidad. El documento de esta herramienta investiga la causa y los efectos de los errores en los requisitos de lenguaje natural en los proyectos basados en el tiempo y el presupuesto. Además, analiza la forma en que una nueva clase creciente de herramientas de análisis de requisitos automatizados, se basan en el PLN computacional. Debido a que este análisis puede ser utilizado por los ingenieros de requisitos en el momento del ciclo de vida de desarrollo de software para que puedan retomar el control y asegurarse de que los proyectos exitosos están en el tiempo y dentro del presupuesto*. (Vaz 2018)*

El analizador de calidad de requisitos (RQA): es una herramienta comercial desarrollada por The Reuse Company. La herramienta proporciona indicadores morfológicos, léxicos, analíticos y relacionales (Gnesi et al. 2019). RQA es una herramienta de software que ayuda a definir, medir, gestionar y mejorar la calidad de las especificaciones de los requisitos. RQA utiliza un amplio rango de métricas basadas en corrección, consistencia y completitud para así evaluar la calidad de las especificaciones de los requisitos. Mientras las métricas individuales son adecuadas para evaluar la calidad de requisitos individuales, las métricas globales pueden evaluar la calidad de una especificación completa. Está basado en el PLN y técnicas semánticas, permitiendo así una comprensión del significado real de las necesidades en lugar de sólo las palabras claves analizadas. *(Kasser et al.2018)*

Asistente para la elaboración de requisitos (RAT): es el asistente para analistas e ingenieros de requisitos. RAT utiliza un grupo de patrones y guías para ayudar al auto completamiento, atendiendo siempre una gramática correcta. Asegurando el patrón correcto la herramienta brinda beneficios como la completitud, corrección, consistencia, reutilización y formalización. Este asistente incluye la detección de inconsistencia, requisitos no atómicos y ambiguos, uso del verbo en tiempo erróneo. *(Overti 2022)*

Analizador de calidad para documentos oficiales (QuOD): Herramienta basada en el PLN orientada a detectar defectos lingüísticos en las descripciones de los procesos de negocios y proporcionar recomendaciones de mejora. Es la primera herramienta que aborda el problema de la identificación de defectos lingüísticos en las declaraciones de los procesos de negocios. QuOD analiza cuatro atributos de calidad como son la sencillez, la no ambigüedad, la claridad de contenido y la corrección. *(Ferrari et al. 2019)*

A continuación, se presentan los Indicadores (I) a tener en cuenta para la comparación de las herramientas seleccionadas y una breve explicación de cada uno. (Naeem et.al. 2019)

I1. Nivel de evaluación: la herramienta evalúa la calidad de cada requisito por separado, de un conjunto de requisitos o de un documento de especificación de requisitos.

I2. Criterios de calidad/parámetros: parámetros sobre la base de los cuales las herramientas evalúan la calidad de los requisitos o documento de especificación de requisitos.

I3. Formato de entrada de los requisitos: trata sobre el formato de los archivos que la herramienta admite, puede ser .doc, .pdf, .xls, .xml, .txt, .csv, texto plano, entre otros

I4. Empleo de estándares: estándar o estándares de calidad de requisitos para requisitos individuales o documentos de especificación de requisitos.

I5. PLN: la herramienta procesa lenguaje natural o comprueba errores gramaticales.

I6. Retroalimentación para palabras resaltadas: herramientas que ofrecen descripción adicional o brindan recomendaciones para el ingeniero de requisitos.

En la Tabla 6 se muestra el resultado del estudio comparativo de las herramientas analizadas.

Tabla 6: Resumen de análisis de las herramientas seleccionadas

Herramientas	I1	I2	I3	I4	I5	I6
QVscribe[2]	Evalúa en escala de (1-5) 1 para peor 5 para excelente	Calidad Similitud Consistencia	Entrada en formato .doc Genera informes en .pdf	ISO/IEC/IEEE 29148	procesa lenguaje natural	Proporciona una descripción
RAT[3]	Evalúa en porciento	Completitud Corrección	Entrada en formato. doc,dox, .pdf, .text, Genera una tabla resultante en contenido estático	ISO/IEC/IEEE 29148 IEEE 830	procesa lenguaje natural	Describe detalladamente las mejoras en forma de tabla
RQA[4]	Evalúa en malo medio bueno	Corrección Integridad Coherencia	Entrada en formato Excel	ISO 15288 ISO 12207	procesa lenguaje natural	Proporciona una descripción
QuOD[5]	Evalúa en porciento	Corrección Sencillez Claridad de contenido No	Entrada en texto plano (copiar y pegar)	No utiliza ninguna norma o estándar para el análisis de	procesa lenguaje natural y errores gramaticales	Proporciona una retroalimentación limitada

[2] https://gracorp.com/
[3] https://github.com/Zeeshan89/Requirement-Assessment-Tool-RAT-
[4] https://www.reusecompany.com/rqa-quality-studio
[5] https://narwhal.it/quod/

		ambigüedad		calidad		

El estudio realizado a las herramientas anteriores evidencia la necesidad de desarrollar una herramienta analizadora de requisitos en español, debido a que las existentes para este fin se encuentran en inglés y las pocas en lenguaje español son para un dominio o contexto específico. Aunque el idioma de las herramientas constituye un inconveniente para una posible integración, este estudio demuestra que la mayoría de estas herramientas son obsoletas, privadas y otras no se encuentran disponibles, ver tabla 5. Las herramientas seleccionadas sirven de referencia para las distintas funcionalidades de la propuesta de solución.

1.5 Conjunto de datos de requisitos de software en lenguaje natural

El PLN consiste de dos ingredientes fundamentales: los algoritmos y los datos *(Ferrari et al. 2019)*. La comunidad de PLN puede proporcionar a la comunidad de los ingenieros de requisitos avanzados algoritmos para el procesamiento de textos. Sin embargo, no es suficiente la disponibilidad de conjuntos de datos, para probar estas poderosas herramientas.

La forma más simple de los conjuntos de datos suele ser una tabla de bases de datos. En las tablas aparecen variables de las cuales se representan todos sus datos o valores. Además, pueden relacionarse datos de una tabla con otras tablas entre sí.

Existen conjunto de datos para el procesamiento de requisitos en lenguaje natural, entre los que se encuentran los que se presentan a continuación:

Kaggle (Software Requirements Dataset): el conjunto de datos contiene varios requisitos funcionales y no funcionales para diferentes tipos de productos de software. El etiquetado especifica los siguientes requisitos. [6]

- Functional (F)
- Availability (A)
- Fault Tolerance (FT)
- Legal (L)
- Look & Feel (LF)
- Maintainability (MN)
- Operational (O)
- Performance (PE)
- Portability (PO)
- Scalability (SC)
- Security (SE)
- Usability (US)

PURE (PUblic REquirements Dataset): este es un conjunto de datos de 79 documentos de requisitos en lenguaje natural disponibles públicamente y recogidos de la web. El conjunto de datos incluye 34.268 frases y puede utilizarse para tareas de PLN típicas de la IR, como la síntesis de modelos, la identificación de abstracciones y la evaluación de la estructura de los documentos. Además, puede anotarse para que sirva de referencia para otras tareas, como la detección de ambigüedades, la categorización de requisitos y la identificación de requisitos equivalentes. *(Soren 2020)*[7]

[6] https://www.kaggle.com/datasets/iamsouvik/software-requirements-dataset 2020

[7] https://zenodo.org/record/1414117

ReqEval: conjunto de datos elaborado por cinco anotadores con experiencia en ingeniería de software y lingüística computacional. La tarea principal de este conjunto de datos involucra a la ambigüedad que se encuentra en las especificaciones de requisitos. El conjunto de datos contiene 200 oraciones de las cuales 102 están marcadas como ambiguas. Está dividido un 75% para entrenamiento y un 25% de prueba.*(Abualhaija et al. 2020)*[8]

La mayoría de los conjuntos de datos de requisitos de software que se encuentran disponibles actualmente, son en idioma inglés como los descritos en este epígrafe. Las técnicas y herramientas de PLN para el tratamiento de la ambigüedad en lenguaje español son completamente distintas a las aplicadas en el idioma inglés atendiendo a los principios de cada idioma. Además, estas técnicas de PLN necesitan de grandes bases de datos para ser aplicadas, pero aún son escasos los conjuntos de datos en la IR. Las pocas bases de datos disponibles suelen ser de empresas o de proyectos específicos, lo que dificulta su accesibilidad. El desarrollo de un conjunto de datos de requisitos en español y público, como línea base para la propuesta solución de la investigación, constituye una manera de validar y probar las técnicas de PLN que se definieron, para la implementación de la herramienta.

Para la construcción del conjunto de datos de requisitos de software, que será utilizado como caso de estudio en esta investigación y por tanto dar solución a una de las tareas planteadas se hace necesario emplear el algoritmo kdd, del inglés (Knowledge Discovery in Data bases).

El **descubrimiento de conocimiento en bases de datos** es básicamente un proceso automático en el que se combinan descubrimiento y análisis. El proceso consiste en extraer patrones en forma de reglas o funciones, a partir de los datos, para que el usuario los analice. Esta tarea implica generalmente procesar los datos, hacer minería de datos y presentar resultados. Kdd se relaciona con aprendizaje de máquina y reconocimiento de patrones en el estudio de teorías y algoritmos de minería de datos para modelamiento de datos y extracción de patrones. Así mismo, se enfoca en la extensión de estas teorías y

[8] https://zenodo.org/record/4471411/2020

algoritmos al problema de encontrar patrones entendibles que puedan ser interpretados como conocimiento útil o interesante, y hace un fuerte énfasis sobre el trabajo con grandes conjuntos de datos del mundo real. (Timarán et al. 2018)

El proceso Kdd trae consigo una serie de pasos ETL (extracción, transformación y limpieza) que se le debe realizar al conjunto de datos para una mayor comprensión de los datos. Estos pasos se encuentran agrupados en las etapas que se describen en la tabla 7.

Tabla 7:Resumen de análisis de las herramientas seleccionadas

Etapa	**Descripción**
Selección/Extracción	Se identifica el conocimiento relevante y se define las metas del proceso kdd desde el punto de vista del cliente seleccionando todos los datos sobre el cual se realiza el descubrimiento del conocimiento.
Procesamiento/Limpieza	Analiza la calidad de los datos, se aplica opciones básicas como la remoción de datos ruidoso, se seleccionan estrategias para el manejo de datos desconocidos, nulos y duplicados. En esta etapa es de suma importancia la interacción de los implicados.
Transformación/Reducción	Se buscan características útiles para representar los datos dependiendo de la meta del proceso. Se utilizan métodos de reducción de dimensiones o de transformación para disminuir el número efectivo de variables bajo consideración o para encontrar representaciones invariantes de los datos.
Minería de Datos	El objetivo de la etapa minería de datos es la búsqueda y descubrimiento de patrones

	insospechados y de interés, aplicando tareas de descubrimiento como clasificación, patrones secuenciales y asociaciones.
Interpretación/Evaluación de Datos	se interpretan los patrones descubiertos y posiblemente se retorna a las anteriores etapas para posteriores iteraciones. Esta etapa puede incluir la visualización de los patrones extraídos, la remoción de los patrones redundantes o irrelevantes y la traducción de los patrones útiles en términos que sean entendibles para el usuario. Por otra parte, se consolida el conocimiento descubierto para incorporarlo en otro sistema para posteriores acciones o, simplemente, para documentarlo y reportarlo a las partes interesadas; también para verificar y resolver conflictos potenciales con el conocimiento previamente descubierto.

Kdd es un proceso iterativo e interactivo como se puede observar en la Figura 4, además cada fase o etapa contiene un conjunto de pasos específicos. Los cuales deben cumplirse con eficiencia, para lograr la calidad del conocimiento que se espera al finalizar todo el proceso.

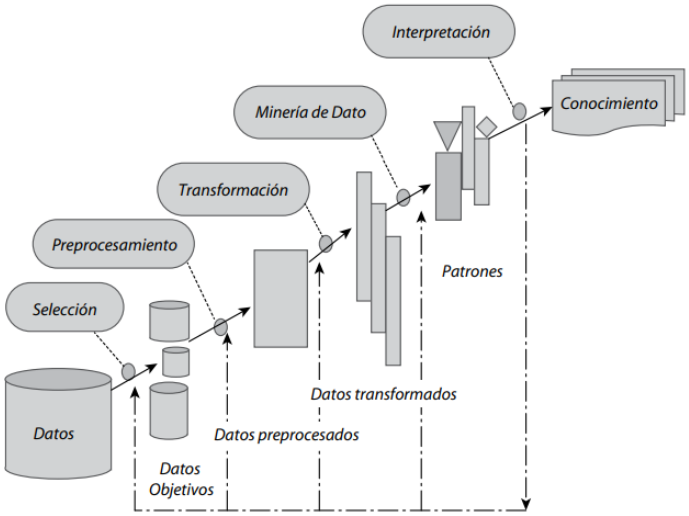


Figura 4: Etapas del proceso KDD (Timarán et al. 2018)

1.6 Herramientas y tecnologías

El uso de herramientas para mejorar tanto la productividad como la calidad en el desarrollo de un proyecto de software ha motivado aún más su uso. Para el desarrollo de la propuesta solución se tendrán en cuenta un conjunto de herramientas y tecnologías lenguajes, las cuales a continuación se describen y se fundamenta su elección.

1.6.1Herramientas de Ingeniería del Software Asistida por Computadora

Las herramientas CASE por sus siglas en inglés (Computer Aided Software Engineering). Ayudan a todos los encargados de la ingeniería del software de un proyecto en las actividades destinadas al proceso de software *(Pressman 2020).*

Dentro de las herramientas y tecnologías de desarrollo a utilizar se encuentra el Visual Paradigm la cual es una herramienta CASE. La misma propicia un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación. Se caracteriza por la disponibilidad en múltiples plataformas (Windows, Linux). Además, el diseño centrado en casos de uso y enfocado al negocio garantiza un software de mayor calidad (Dr. Henderi et.al.2022).

Otra herramienta CASE que emplea el lenguaje de modelado UML (Lenguaje Unificado de Modelado) es Rational Rose es muy potente y soporta la generación de diversos

diagramas (Hung et.al. 2022). Debido a que Visual Paradigm permite guardar todo el modelado en un único archivo se decide la utilización de esta herramienta para el modelado del proceso de desarrollo de la propuesta solución. Además, existe una licencia disponible en la UCI y es la herramienta usada en el desarrollo de software en la universidad.

1.6.2 Lenguajes de programación

Según el ranking de los lenguajes de programación más demandados en el 2022, se destacan como más usados, Java, Java Script, Python, C++ (José 2022). A continuación, se muestra una descripción de los lenguajes más demandados luego de un previo análisis.

Java es un lenguaje de programación ampliamente utilizado para codificar aplicaciones web. Ha sido una opción popular entre los desarrolladores durante más de dos décadas, con millones de aplicaciones Java en uso en la actualidad. Java es un lenguaje multiplataforma, orientado a objetos y centrado en la red que se puede utilizar como una plataforma en sí mismo. Es un lenguaje de programación rápido, seguro y fiable para codificar todo, desde aplicaciones móviles y software empresarial hasta aplicaciones de macro datos y tecnologías del lado del servidor. Java es popular porque ha sido diseñado para facilitar su uso. Algunas razones por las que los desarrolladores siguen eligiendo Java sobre otros lenguajes de programación incluyen recursos de aprendizaje de alta calidad, funciones y bibliotecas incorporadas, seguridad y plataforma independiente. (Alexandra 2022)

Java Script es un lenguaje de programación ligero o interpretado. Es más conocido como un lenguaje de scripting (secuencias de comandos) para páginas web, y es usado en muchos entornos fuera del navegador, tal como Node.js, Apache CouchDB y Adobe Acrobat. No necesita un compilador para traducir su código como C o C ++. Se ejecuta directamente en un navegador web y es compatible con la mayoría de los navegadores web y móviles como Google Chrome, Android, iPhone, etc. Java Script es un lenguaje de programación completo que puede hacer la mayoría de las cosas que puede declarar variables, definir e invocar funciones, incluidas las funciones de flecha y definición de clases y objetos de Java Script entre otras funcionalidades que realizan cualquier otro lenguaje como Python. Cabe destacar que Java y Java Script no son lo mismo debido a

que Java es mucho más complejo, aunque también más potente y robusto *(Haverbeke 2018*). En el Anexo 4 se encuentra una tabla comparativa de ambos leguajes. Este leguaje brinda una librería que se describe a continuación, la cual se empleara en el desarrollo de la solución.

React es una biblioteca de Java Script una de las más populares, no es un framework a diferencia de Angular, que si lo es. Es un proyecto de código abierto creado por Facebook que se utiliza para construir interfaces de usuario en el Frontend. Uno de los aspectos más importantes de React es el hecho de que se pueden crear componentes, que son como elementos HTML personalizados y reutilizables, para construir rápida y eficazmente interfaces de usuario. React también agiliza la forma en que se almacenan y manejan los datos *(Boduch et al. 2020).* Luego de un estudio previo se decidió el uso React porque no deja fuera al resto de las herramientas tecnológicas, para desarrollar nuevas características sin necesidad de volver a escribir el código existente. Ayuda a crear interfaces de usuario interactivas de forma sencilla y diseña vistas simples para cada estado de la aplicación. Se encarga de actualizar y renderizar de manera eficiente los componentes correctos cuando los datos cambian.

C++ constituye la evolución del lenguaje C. La intención de su creación fue extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. El alto rendimiento es una de sus principales características. Esto es debido a que puede hacer llamadas directas al sistema operativo, es un lenguaje compilado para cada plataforma, posee gran variedad de parámetros de optimización y se integra de forma directa con el lenguaje ensamblador. A pesar de que ya tiene muchos años, el lenguaje se ha ido actualizando, permitiendo crear, relacionar y operar con datos complejos y ha implementado múltiples patrones de diseño. *(Reyes 2018) (Miguel Casado et al.2021)*

Python es un lenguaje de programación de alto nivel que se utiliza para desarrollar aplicaciones de todo tipo. A diferencia de otros lenguajes como Java o .NET, se trata de un lenguaje interpretado, es decir, que no es necesario compilarlo para ejecutar las aplicaciones escritas en Python, sino que se ejecutan directamente por el ordenador utilizando un programa denominado interpretador, por lo que no es necesario “traducirlo” a lenguaje máquina. Python es un lenguaje sencillo de leer y escribir debido a su alta

similitud con el lenguaje humano. Además, se trata de un lenguaje multiplataforma de código abierto y, por lo tanto, gratuito, lo que permite desarrollar software sin límites. Con el paso del tiempo, Python ha ido ganando adeptos gracias a su sencillez y a sus amplias posibilidades, sobre todo en los últimos años, ya que facilita trabajar con inteligencia artificial, big data, aprendizaje automático y la ciencia de los datos*. (Mckinney 2018)*

La elegante sintaxis y tipado dinámico de Python entre otras características y funciones que brinda, hizo que los autores de la presente investigación se decidieran por el uso de este lenguaje en el desarrollo de la herramienta propuesta. A continuación, se abordará más sobre este lenguaje.

Python es un lenguaje de programación potente y fácil de aprender. Tiene estructuras de datos de alto nivel eficientes y un simple pero efectivo sistema de programación orientado a objetos. Su naturaleza interpretada lo convierten en un lenguaje ideal para scripting y desarrollo rápido de aplicaciones en muchas áreas, para la mayoría de las plataformas. Python también es apropiado como un lenguaje para extender aplicaciones modificables. *(Cuevas 2020)*

En resumen, Python 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. 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 la ciencia de los datos. Esto, sumado a su potencia, su carácter de fuente abierta y su facilidad de aprendizaje le ha llevado a ocupar el puesto número uno en lenguajes propios de la analítica de datos. Además de librerías de herramientas científicas, numéricas, de herramientas de análisis y estructuras de datos, o de algoritmos de aprendizaje automático (Machine Learning) como NumPy, SciPy, Matplotlib, Pandas o PyBrain, Python ofrece entornos interactivos de programación orientados al Data Science. 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. *(Sergio 2022)*

1.6.3 Librerías de Python

Al profundizar en las características del lenguaje de programación Python se logró conocer las facilidades que este brinda, con el propósito de ser utilizadas a lo largo de la investigación. A continuación, se presentan algunas de sus librerías.

- **Pandas** proporciona estructuras de datos de alto nivel y funciones diseñadas para trabajar con datos estructurados o tabulares de forma rápida, fácil y expresiva. Pandas combina las ideas de alto rendimiento de NumPy con las capacidades flexibles de manipulación de datos de las hojas de cálculo y las bases de datos relacionales (como SQL). Proporciona una sofisticada funcionalidad de indexación para facilitar la remodelación de datos, realizar agregaciones y seleccionar subconjuntos de datos. *(Harrison 2020)*

- **NumPy** abreviatura de Numerical Python, ha sido durante mucho tiempo una piedra angular de la computación numérica en Python. Proporciona las estructuras de datos, los algoritmos y las bibliotecas necesarias para la mayoría de las aplicaciones científicas que implican datos numéricos en Python. NumPy contiene, entre otras cosas:

    - Un objeto de matriz multidimensional rápida y eficiente.
    - Funciones para realizar cálculos por elementos con matrices u operaciones matemáticas entre matrices.
    - Herramientas para leer y escribir conjuntos de datos basados en *arrays* en el disco.
    - Operaciones de álgebra lineal, transformación de Fourier y generación de números aleatorios.
    - Una API de C madura para permitir que las extensiones de Python y el código nativo de C o C++ accedan a las estructuras de datos de NumPy y a sus funciones. *(Oliphant 2018)*

- **Matplotlib** es la biblioteca más popular de Python para producir gráficos y otras visualizaciones de datos bidimensionales. Fue creada originalmente por John D. Hunter y ahora es mantenida por un gran equipo de desarrolladores. Está diseñada para crear

gráficos aptos para su publicación. Aunque hay otras bibliotecas de visualización disponibles para los programadores de Python, matplotlib es la más utilizada y, como tal, tiene generalmente una buena integración con el resto del ecosistema. Es una elección segura como herramienta de visualización por defecto. *(Jhon hunter et al.2018)*

- **Tensor Flow** es una plataforma de código abierto de extremo a extremo para el aprendizaje automático. Cuenta con un ecosistema integral y flexible de herramientas, bibliotecas y recursos de la comunidad que permite que los investigadores innoven con el aprendizaje automático y los desarrolladores creen e implementen aplicaciones con tecnología de aprendizaje automático (AA) fácilmente. *(Miguel de Icaza 2022)*

- **PyTorch** es una librería de código abierto basada en Python, enfocada a la realización de cálculos numéricos mediante programación de tensores, lo que facilita su aplicación al desarrollo de aplicaciones de aprendizaje profundo. La sencillez de su interfaz, y su capacidad para ejecutarse en GPUs (lo que acelera el entrenamiento de los modelos), lo convierten en la opción más asequible para crear redes neuronales artificiales. *(Marcos 2022)*

- **Text Blob** esta librería no puede considerarse como una librería NLP por sí misma. En su lugar, se trata de una interfaz montada sobre NLTK y que agrega componentes tales como analizadores de sentimiento. Su uso permite resolver parte de la complejidad de NLTK, al mismo tiempo que permite crear rápidamente prototipos. *(Cassanelli et al. 2019)*

- **RE** es un módulo de Python que se basa en la sintaxis utilizada para las expresiones regulares. Este módulo permite realizar varias funciones como compilar una expresión regular y comprobar si una determinada cadena coincide con una expresión regular dada. Es principalmente utilizado para la búsqueda y manipulación de cadenas. Además, se utiliza con frecuencia para extraer una gran cantidad de datos de un sitio web.(Bahit 2020)

- **Googletrans** es una biblioteca de Python gratuita e ilimitada que implementó la API de Google Translate. Esta biblioteca utiliza la API Ajax de Google Translate para realizar llamadas a métodos tales como detectar y traducir. Además, permite realizar varias funciones como la detección automática de idioma, traducciones masivas y URL de servicio personalizable. Debido a las limitaciones de la versión web de Google Translate, esta API no garantiza que la biblioteca funcione correctamente en todo momento.[9]

Luego de este estudio se puede resumir que las librerías de Python son amplias y cuentan con gran variedad de módulos que permiten el acceso a funcionalidades específicas de cada una, lo que facilita el desarrollo a la hora de programar. Las librerías descritas brindan características necesarias para el desarrollo de la propuesta solución, es por ello que se pretende usar Re, Numpy, Pandas, Pandas, Text Blob y Googletrans para la implementación de los algoritmos pertenecientes a la herramienta propuesta.

1.6.4 Framework de desarrollo

Existen diversos framework de desarrollo, pero en este epígrafe solo se hará énfasis en los de desarrollo web, debido a que la herramienta a desarrollar es para la web. Los frameworks hacen que el desarrollador se centre en el problema que quiere resolver y no en la implementación de funcionalidades que normalmente son de uso común y que ya están implementadas por otros *(Saul 2019).* Entre los framework existentes que ayudarían al desarrollo de esta aplicación de manera más consistente y eficaz se decidió hacer uso de los siguientes.

Bootstrap ofrece la posibilidad de crear un sitio web totalmente responsive mediante el uso de librerías CSS. En estas librerías, hay un gran número de elementos ya desarrollados y listos para ser utilizados como pueden ser botones, menús, cuadros e incluso un amplio listado de tipografías *(Antonio 2020)*. Este framework será utilizado en el desarrollo del Frontend de la herramienta propuesta; debido a sus características de excelencia para crear interfaces de usuarios limpias y totalmente adaptables a cualquier

[9] https://pypi.org/project/googletrans/

tipo de dispositivo y pantalla, independientemente de su tamaño. Además de que es muy sugerente.

El siguiente framework será utilizado en el desarrollo del motor web es decir el backend de la aplicación.

Django es un framework web de alto nivel, escrito en Python, que ayuda al desarrollo rápido y a un diseño limpio y pragmático. Resuelve una buena parte de los problemas del desarrollo web de tal manera que uno se pueda enfocar en su aplicación sin reinventar códigos reutilizables implementados por otros programadores. Una de las características de Django es que es gratis y de código abierto *(Luis 2020)*. Se definió la utilización de este debido a que hace uso del lenguaje Python, el cual es en el que la aplicación será implementada. Además, Django ayuda a los desarrolladores a evitar varios errores comunes de seguridad al proveer un framework que ha sido diseñado para hacer lo correcto para proteger el sitio web automáticamente. *(Saul 2019)*

1.7 Metodología de desarrollo de software

Las metodologías ágiles han ganado una amplia aceptación en el desarrollo de software, estas metodologías fueron creadas como una alternativa a las metodologías tradicionales. El enfoque ágil busca disminuir la carga impuesta por el enfoque tradicional en proyectos de pequeña y mediana escala. Este enfoque ágil a diferencia del tradicional es adaptativo para cada tipo de proceso y está orientado a las personas *(Fuentes 2022).* Dentro de las metodologías ágiles se encuentran AUP (Proceso Unificado Ágil), Scrum y XP (Programación Extrema).

Scrum: método de desarrollo ágil de software concebido por Jeff Sutherland y su equipo de desarrollo a principios de los 90. Los principios de Scrum son congruentes con el manifiesto ágil y se utilizan para guiar actividades de desarrollo dentro de un proceso de análisis que incorpora las siguientes actividades estructurales: requerimientos, análisis, diseño, evolución y entrega. Dentro de cada actividad estructural, las tareas del trabajo ocurren con un patrón del proceso llamado sprint. El trabajo realizado dentro de un sprint se adapta al problema en cuestión y se define y con frecuencia se modifica en tiempo real por parte del equipo de desarrollo. Scrum acentúa el uso de un conjunto de patrones de

ser eficaces para proyectos con plazos de entrega muy apretados, requerimientos cambiantes y negocios críticos. Cada uno de estos patrones de proceso define un grupo de acciones de desarrollo*. (Nader 2019) (Pressman 2020)*

AUP: el proceso unificado ágil (AUP) adopta una filosofía “en serie para lo grande” e “iterativa para lo pequeño” a fin de construir sistemas basados en computadora. Brinda un revestimiento en serie (por ejemplo, una secuencia lineal de actividades de ingeniería de software) que permite que el equipo visualice el flujo general del proceso de un proyecto de software. Sin embargo, dentro de cada actividad, el equipo repite con objeto de alcanzar la agilidad y entregar tan rápido como sea posible incrementos de software significativos a los usuarios finales. Cada iteración del AUP aborda las actividades modelado, implementación, pruebas, despliegue, configuración y administración del proyecto, administración del ambiente*.(Pressman 2020)*

XP: la programación extrema es una metodología de desarrollo de la ingeniería de software formulada por Kent Beck, autor del primer libro sobre la materia en el 1999. Es el más destacado de los procesos ágiles de desarrollo de software. La programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Se aplica de manera dinámica durante el ciclo de vida del software y es capaz de adaptarse a los cambios de requisitos. *(Hoda 2019)*Esta metodología contiene las siguientes fases:

- **Planificación:** se obtienen los requisitos con los cuales se elaboran las historias de usuarios, además en esta fase se estima el tiempo y esfuerzo que se empleará en cada historia de usuario.

  Las historias de usuario (HU) son una forma rápida de administrar los requisitos de los usuarios sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos*. (del Águila Cano, 2022)* Mientras que la estimación del esfuerzo es la capacidad que deberá tener el equipo del proyecto para desarrollar las HU. Se traduce en estimación de esfuerzo por HU. La metodología XP propone que un punto de estimación equivale idealmente a

una semana sin la influencia o retraso provocado por factores externos. *(Hoda 2019)*

- **Diseño:** conseguir diseños simples y sencillos para procurar hacerlo todo lo menos complicado posible para conseguir un diseño fácilmente entendible e implementable que a la larga costará menos tiempo y esfuerzo desarrollar. La complejidad innecesaria y el código extra debe ser eliminado inmediatamente. *"El diseño adecuado para el software es aquel que supera con éxito todas las pruebas, no tiene lógica duplicada, refleja claramente la intención de implementación de los programadores y tiene el menor número posible de clases y métodos". (Hoda 2019)* Dentro de los principales elementos del diseño de software se encuentra la arquitectura que no es más que la estructura del sistema, lo que comprende a los componentes del software, sus propiedades externas visibles y las relaciones entre ellos. *(Pressman 2020).*
- **Codificación**: el cliente es una parte más del equipo de desarrollo; su presencia es indispensable en las distintas fases de XP. A la hora de codificar una historia de usuario su presencia es aún más necesaria, además de que son los que crean las historias de usuario y negocian los tiempos en los que serán implementadas. La codificación debe hacerse ateniendo a estándares de codificación ya creados o definidos previamente. Programar bajo estándares mantiene el código consistente y facilita su comprensión y escalabilidad. *(Hoda 2019)*
- **Pruebas:** verificar el funcionamiento de la codificación implementada y validar las historias de usuario. La fase de prueba constituye uno de los pilares de la metodología XP *(Pressman 2020) (Hoda 2019)* ; al realizarse durante todo el ciclo de desarrollo del producto se garantiza que los errores sean detectados rápidamente y corregidos a tiempo, lo que permite el éxito del producto. Existen cuatro niveles de pruebas:
- **Pruebas de unidad:** se concentra en el esfuerzo de verificación de la unidad más pequeña del diseño, ya sea una clase o un método del software. *(Pressman 2020)*
- **Pruebas de integración:** es una actividad sistemática para elaborar la arquitectura de software mientras, al mismo tiempo, se aplican las pruebas para descubrir errores asociados con la interfaz. *(Pressman 2020)*

- **Pruebas de sistema:** el objetivo principal es ejercitar profundamente el sistema de cómputo. Aunque cada prueba tiene un propósito diferente, todas trabajan para verificar que se hayan integrado adecuadamente todos los elementos del sistema y que realizan las funciones adecuadas. *(Pressman 2020)*
- **Pruebas de aceptación:** son mayormente desarrolladas y ejecutadas por el cliente o un especialista de la aplicación. Se realiza con el objetivo de determinar cómo el sistema satisface sus criterios de aceptación, validando los requisitos que han sido levantados para el desarrollo, incluyendo la documentación y procesos de negocio. *(Pressman2020).*

  Luego de los elementos planteados sobre las metodologías se decidió emplear XP como metodología de desarrollo de software, debido a que esta se caracteriza por los siguientes valores la simplicidad, comunicación, retroalimentación y coraje. Además, la programación es organizada propensa a una menor taza de errores y una mayor satisfacción del programador y cliente.

Conclusiones parciales

- El estudio de los diversos criterios de calidad de requisitos de software abordados en varias literaturas permitió identificar que el criterio de no ambigüedad es de interés significativo para la ingeniería de requisitos, por tanto, es el criterio a tratar en esta investigación.
- La profundización del PLN permitió conocer sobre la relación existente entre técnicas, herramientas y recursos de PLN que permiten llevar a cabo tareas de PLN como la detección, la que se trata a lo largo de la presente investigación.
- El estudio de herramientas de PLN permitió conocer que la mayoría de las herramientas de PLN para la IR que se encuentran disponible son en lengua inglesa y están obsoletas.
- La profundización de los conjuntos de datos disponibles demostró que los componentes para llevar a cabo el PLN son los datos y algoritmos.

- El estudio de las técnicas de PLN permitió seleccionar la tokenización, la eliminación de ruidos y el etiquetado de parte del discurso como técnicas de PLN para el desarrollo de la propuesta solución.

Capítulo II. Análisis y diseño de la propuesta solución

Introducción

En el presente capítulo se describen los principales elementos que se tendrán en cuenta en el desarrollo de la solución propuesta. El uso de XP como metodología organizará el proceso de trabajo, en dependencia de las fases y artefactos que define XP, entre los que se encuentran las HU y estimación por esfuerzo. En este capítulo quedan definidos los patrones de diseño, la arquitectura, la base de datos y los algoritmos que permitirán a la herramienta detectar y analizar las ambigüedades.

2.1 Descripción general de SIDARES

La herramienta informática propuesta permite identificar la existencia de términos ambiguos en requisitos de software. Los requisitos analizados por esta, pueden ser redactados de forma manual o importados desde un documento con extensión (.xlsx,.csv). La herramienta una vez que obtiene y analiza los requisitos, es capaz de mostrar el requisito con la ambigüedad que detectó ya sea léxica o sintáctica, con el propósito de que la palabra o frase ambigua pueda ser corregida por el ingeniero de requisitos. A continuación, la Figura 5 muestra la interfaz de la herramienta propuesta.

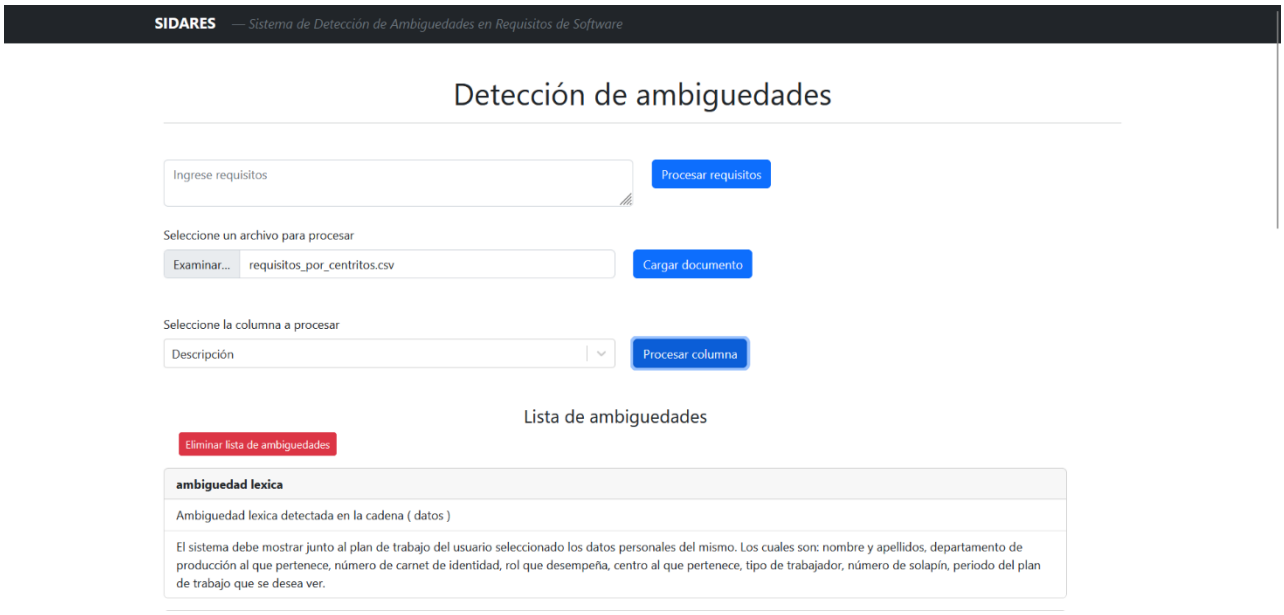


Figura 5: SIDARES (Sistema de Detección de Ambigüedades en Requisitos de Software)

La interfaz de la herramienta está compuesta por un conjunto de componentes visuales. En la parte superior izquierda se encuentra el nombre del sistema. SIDARES, cuenta con los botones “Procesar requisito” y “Procesar columna “, los cuales son los encargados de analizar el requisito y documento de requisitos para así detectar ambigüedades léxicas o sintácticas. El botón “Cargar documento” es el que permite importar el documento de requisito para ser posteriormente analizado. En la parte inferior del contenedor principal se encuentra una ventana que muestra los requisitos con las ambigüedades detectadas, además del botón “Eliminar lista de ambigüedades” para vaciar este campo si así lo desea el usuario.

2.2 Definición de un conjunto de datos para procesamiento de requisitos en lenguaje natural

Como parte de la propuesta solución se encuentra la elaboración de un conjunto de datos de requisitos de software que servirá como caso de estudio de la herramienta. Para la confección del conjunto de datos, se recopilaron los documentos que contenían los requisitos de software de los diferentes proyectos de cada centro de la UCI, siendo esta la primera etapa del algoritmo Kdd definido en el capítulo 1. Se realizaron dos etapas para el desarrollo del conjunto de datos.

La primera etapa del trabajo consistió en identificar los documentos de requisitos disponibles en la universidad. Se recopilaron de cada uno de los centros de desarrollo de software, los documentos que recogían requisitos funcionales y no funcionales, lo cual permitió recolectar un total de 14 documentos. Todos ellos estaban en un formato uniforme .xlsx y presentaban un contenido estructurado con una distribución de campos donde se destacaban: ‘tipo de requisito’, ‘centro ’y ‘descripción del requisito’.

En una segunda etapa fueron sometidos a un proceso ETL (Extracción-Transformación-Limpieza) fases o etapas de Kdd. En esta etapa se estructuraron uniformemente y eliminaron ruidos e impurezas que pudieran afectar el futuro análisis del contenido de estos. El resultado fue almacenado en un documento único de formato .csv el cual quedó conformado por un total de 19 357 requisitos entre los que se encuentran 19 139 requisitos funcionales y 216 no funcionales descritos en lenguaje natural. La Figura 6 muestra la estructura del conjunto de datos de requisitos.

868	CDAE	RF58_Exportar Anexo III Medidas Disciplinarias .	El sistema debe permitir exportar el Anexo III relacionado con las medidas disciplinarias aplicadas, en formato PDF.	Funcional
869	CDAE	RF59_Exportar Anexo IV Principales Deficiencias	El sistema debe permitir exportar el Anexo IV relacionado con las deficiencias detectadas en las acciones de control ejecutadas, en formato PDF.	Funcional
870	CDAE	RF63_Eliminar organismos de la plantilla	El sistema debe permitir eliminar los organismos a la plantilla de auditores.	Funcional
871	CDAE	RF65_Modificar completamiento de la plantilla	El sistema debe permitir modificar la información necesaria para actualizar el completamiento de la plantilla de auditores.	Funcional
872	CDAE	RF61_Modificar organismos de la plantilla	El sistema debe permitir modificar los organismos de la plantilla de auditores.	Funcional
873	CDAE	RF64_Añadir completamiento de la plantilla	El sistema debe permitir adicionar la información necesaria para realizar el completamiento de la plantilla de auditores.	Funcional
874	CDAE	RF62_Mostrar organismos de la plantilla	El sistema debe permitir mostrar los organismos de la plantilla de auditores, filtrando la información y exportándola a formato PDF o XLS.	Funcional
875	CDAE	RF57_Exportar Anexo II Estructura y Completam	El sistema debe permitir exportar el Anexo II relacionado con la estructura y completamiento de la plantilla, en formato PDF.	Funcional
876	CDAE	RF60_Adicionar organismos a la plantilla	El sistema debe permitir adicionar los organismos a la plantilla de auditores.	Funcional
877	CEDIN	Agregar Subcanal	El sistema debe permitir agregar un subcanal.	Funcional
878	CEDIN	Modificar tecla funcional	El sistema debe permitir modificar la tecla funcional seleccionada. (se repiten los campos de adicionar tecla funcional).	Funcional
879	CEDIN	Agregar grupo operacional de privilegio	El sistema debe permitir agregar grupos operacionales por privilegios.	Funcional
880	CEDIN	Importar grupo operacional de privilegio	El sistema deber permitir importar grupos operacionales por privilegio.	Funcional
881	CEDIN	Eliminar grupo operacional de privilegio	El sistema deber permitir eliminar grupo operacional de privilegio. El cual debe haber sido creado.	Funcional
882	CEDIN	Modificar datos del grupo operacional de privile	El sistema deber permitir modificar los datos del grupo operacional de privilegio.	Funcional
883	CEDIN	Exportar grupo operacional de privilegio	El sistema deber permitir exportar los grupos operacionales de privilegio.	Funcional
884	CEDIN	Crear nuevo proyecto	El sistema debe permitir crear un nuevo proyecto.	Funcional
885	CEDIN	Cargar proyecto desde el servidor	El sistema debe permitir cargar un proyecto desde el servidor, el cual debe estar creado previamente.	Funcional
886	CEDIN	Cargar proyecto local	El sistema debe permitir cargar un proyecto local, el cual debe estar creado previamente.	Funcional
887	CEDIN	Eliminar proyecto	El sistema debe permitir eliminar un proyecto local, el cual debe estar creado previamente.	Funcional
888	CEDIN	Importar proyecto	El sistema debe permitir que se importen proyectos.	Funcional
889	CEDIN	Modificar datos del proyecto	El sistema debe permitir modificar los datos de un proyecto.	Funcional
890	CEDIN	Exportar proyecto local	El sistema debe permitir exportar proyectos locales.	Funcional
891	CEDIN	Exportar proyecto a servidor	El sistema debe permitir exportar proyectos al servidor.	Funcional
892	CEDIN	Agregar nodo	El sistema debe permitir agregar nodos.	Funcional
893	CEDIN	Importar nodo	El sistema debe permitir importar nodos.	Funcional
894	CEDIN	Modificar propiedades del nodo	El sistema debe permitir modificar propiedades del nodo existente.	Funcional

Figura 6: Conjunto de datos de requisitos

El conjunto de datos de requisito está estructurado por cuatro columnas, que contienen la siguiente información: centro, nombre del requisito, descripción y tipo.

2.3 Proceso de desarrollo

En el capítulo 1 quedó seleccionado el uso de la metodología XP para guiar el proceso de desarrollo de software de la presente investigación. A continuación, se describe como fueron llevadas a cabo las fases de planificación y diseño. Las fases de prueba y codificación se desarrollarán en el próximo capítulo.

2.3.1 Planificación

En esta fase se describen los requisitos de la herramienta haciendo uso de las HU y se realiza una estimación del esfuerzo necesario en cada una de ellas. Los encargados de dar solución a la problemática se familiarizan con las tecnologías y herramientas que se utilizarán.

Requisitos de la solución

Los requisitos describen las características de una solución que cumpla con necesidades de las partes interesadas. En reiteradas ocasiones se dividen en requisitos funcionales y no funcionales o de calidad, especialmente cuando los requisitos describen una solución de desarrollo de software.

Para dar respuesta a la problemática se obtuvieron los siguientes **requisitos funcionales** (RF):

RF1. Importar documento: el sistema debe permitir importar documento de requisitos de software.

RF2. Mostrar datos del documento: el sistema debe mostrar los datos (título, extensión) del documento importado.

RF3. Redactar requisito: el sistema debe permitir redactar requisitos de software.

RF4. Mostrar tipo de ambigüedad: el sistema debe mostrar el tipo de ambigüedad que presente el requisito de software.

RF5. Detectar ambigüedad: el sistema debe permitir detectar ambigüedades (léxicas, sintácticas).

RF6. Eliminar ambigüedad: el sistema debe permitir eliminar la lista de ambigüedades detectadas en los requisitos.

RF7. Seleccionar columna: el sistema debe permitir seleccionar la columna del documento analizar.

Para la investigación se proponen los siguientes **requisitos de calidad** (RC):

Requisitos de apariencia o interfaz externa

RC 1: La herramienta debe contar con una interfaz sugerente que facilite la navegación por ella, es decir que cada opcionalidad le sugiera al usuario la funcionalidad que realiza.

RC 2: La letra debe ser visible y legible.

Requisitos de software

RC 1: Para el uso de la herramienta la PC debe contar con cualquier navegador web.

RC 2: Para el funcionamiento de la herramienta se debe disponer de PostgreSQL 9.2 o superior.

Requisitos de hardware

RC 1: Para el correcto funcionamiento de la herramienta la PC debe contar con 512 MB de memoria RAM.

RC 2: Para el correcto funcionamiento de la herramienta la PC debe contar con un procesador de 2.0 GHz o superior.

Requisito de seguridad

RC 1: Si ocurre alguna excepción la herramienta mostrará un mensaje de alerta.

Los requisitos expuestos en el presente epígrafe fueron analizados manualmente, a partir de los criterios de calidad que brida la ISO/IEC/IEEE 29148 y los criterios del lenguaje que propone (Ferrari et al. 2019). Este previo análisis aseguró la calidad de los requisitos y permitió definir requisitos que no poseen términos negativos, ambiguos y vagos.

Historias de Usuarios

Las HU conforman la parte central de muchas metodologías de desarrollo ágil entre ellas XP.

Para el desarrollo del presente trabajo, se identificaron un total de 7 HU, a continuación, se muestran algunas de ellas, el resto se encuentra en el Anexo 5 del documento:

Tabla 8: HU. Importar documento de requisitos

Historia de usuario	
Número: 1	Nombre de la Historia de Usuario: Importar documento
Modificación de la Historia de Usuario: ninguna	
Usuario: Samira Enríquez González	
Prioridad en negocio: alta	Riesgo en desarrollo: medio
Puntos estimados: 5 días	Iteración asignada: 1
Descripción: El usuario podrá importar un documento con requisitos de software.	
Observaciones: 1. Solo admite documentos con las extensiones (.csv, .xlsx), en caso de cargar un documento con otro formato el sistema debe mostrar una notificación (“formato incorrecto”) 2. En caso de no seleccionar un documento y presionar el botón “Cargar documento” el sistema debe mostrar una notificación (“seleccione un documento”)	

Tabla 9: HU. Detectar ambigüedades léxicas y sintáctica

Historia de usuario	
Número: 5	Nombre de la Historia de Usuario: Detectar ambigüedades léxicas y sintácticas
Modificación de la Historia de Usuario: ninguna	
Usuario: Jonathan Ramírez Reyes	
Prioridad en negocio: alta	Riesgo en desarrollo: alto
Puntos estimados: 10 días	Iteración asignada: 2
Descripción: El usuario podrá detectar las ambigüedades léxicas y sintácticas presentes en el documento o el requisito redactado, haciendo uso de las técnicas de PLN.	
Observaciones: Debe haber requisitos cargados o escritos para ser analizados.	

Estimación de esfuerzo por historias de usuario

Para el desarrollo de la herramienta propuesta se efectuó una estimación de esfuerzo haciendo uso de la técnica de punto de HU, obteniéndose los siguientes resultados.

Tabla 10: Estimación de esfuerzo por historias de usuario

Historias de usuario	**Puntos de estimación (días)**
Importar documento	5
Mostrar datos del documento importado	5
Detectar ambigüedades léxicas y sintácticas	10
Mostrar tipo de ambigüedad	3
Redactar requisito	3
Eliminar ambigüedades	3
Seleccionar columna analizar	3

La estimación de desarrollo de esta herramienta arrojó como resultado que la implementación debía durar un tiempo total de 32 días aproximadamente 8 semanas, donde 5 días equivale a una semana y 40 horas de trabajo.

Plan de iteraciones

El plan de iteraciones constituye varias iteraciones sobre la aplicación antes de su entrega. En este plan queda definida la HU que será implementada en cada iteración. A continuación, se describen cada una de las iteraciones propuestas, donde la duración total de iteraciones en días, se obtiene a partir del esfuerzo en días estimados para implementar cada HU.

- Iteración 1: en esta iteración se implementó las HU 1, 2, 3, 6 y 7 las cuales tienen relación a todo lo referido a los requisitos de software tanto redactados como del documento importado.
- Iteración 2: en esta iteración se implementó las HU 4 y 5 las cuales tienen en cuenta las funcionalidades de detectar, mostrar y señalar ambigüedades.

La tabla 11 muestra la duración de cada iteración, así como el orden en que serán implementadas las HU en cada una de ellas según la prioridad asignada.

Tabla 11:Plan de iteraciones

Iteraciones	Historias de usuario	Duración	Prioridad
1	Importar documento	5 días	Alta
	Mostrar datos del documento importado	5 días	Alta
	Redactar requisito	3 días	Alta
	Eliminar ambigüedades	3 días	Media
	Seleccionar columna analizar	3 días	Media
2	Detectar ambigüedades léxicas y sintácticas	10 días	Alta
	Mostrar tipo de ambigüedad	3 días	Alta

Plan de entregas

Luego de quedar establecido el plan de iteración, se procede a la elaboración del plan de entrega, con la intención de obtener una estimación realista del tiempo que tomaría la

implementación de las funcionalidades que describe cada HU. En la tabla 12 se muestra el plan de entrega.

Tabla 12:Plan de entrega

Producto	Iteración 1	Iteración 2
Herramienta para analizar ambigüedad en requisitos de software	versión 1.0 del producto	versión 1 del producto
Fecha de inicio	18/6/2022	30/6/2022
Fecha de entrega	29/6/2022	22/7/2022

2.3.2 Diseño

La metodología de desarrollo de software XP propone el diseño de la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto. Los principales elementos del diseño de software que propone XP como artefactos se presentan a continuación.

Arquitectura de Software

La arquitectura de software es una representación que permite analizar y evaluar la efectividad del diseño para cumplir con los requisitos establecidos, considerar alternativas arquitectónicas en una etapa temprana posibilita hacer cambios al diseño de forma fácil y sencilla, además reduce los riesgos asociados con la elaboración del software.

Para el desarrollo de la presente investigación se utilizó el patrón arquitectónico Modelo Vista Plantilla (MVT, por sus siglas en inglés) que ofrece Django. El empleo de este patrón se debe a que el framework utilizado para el desarrollo de Backend [10] es Django, específicamente el marco de trabajo de Django Rest Framework para la confección de la API REST, además que el lenguaje definido para la construcción de la herramienta es Python.

MVT es un patrón de diseño o arquitectura de diseño que sigue Django para desarrollar aplicaciones web. Es ligeramente diferente del patrón de diseño Modelo-Vista-

[10] Backend es el entorno donde se ejecutan uno o varios programas informáticos que almacenan, asegura ,procesan y analizan datos en forma continua para luego generar resultados(información), que es representada en forma visual por las Frontend developers (Fernando 2018)

Controlador, comúnmente conocido (*Antonio 2020) (Edgardo 2020)*. Este patrón se conforma por tres componentes el modelo, la vista y la plantilla. El modelo es el cual gestiona los datos y está representado por una base de datos. La vista es la que se encarga de recibir solicitudes y envía respuestas. Además, la vista interactúa con el modelo y una plantilla para completar una respuesta, donde la plantilla es básicamente la capa frontal y el componente HTML dinámico de la aplicación. En la Figura 7 se encuentran representados los componentes del patrón arquitectónico a utilizar.

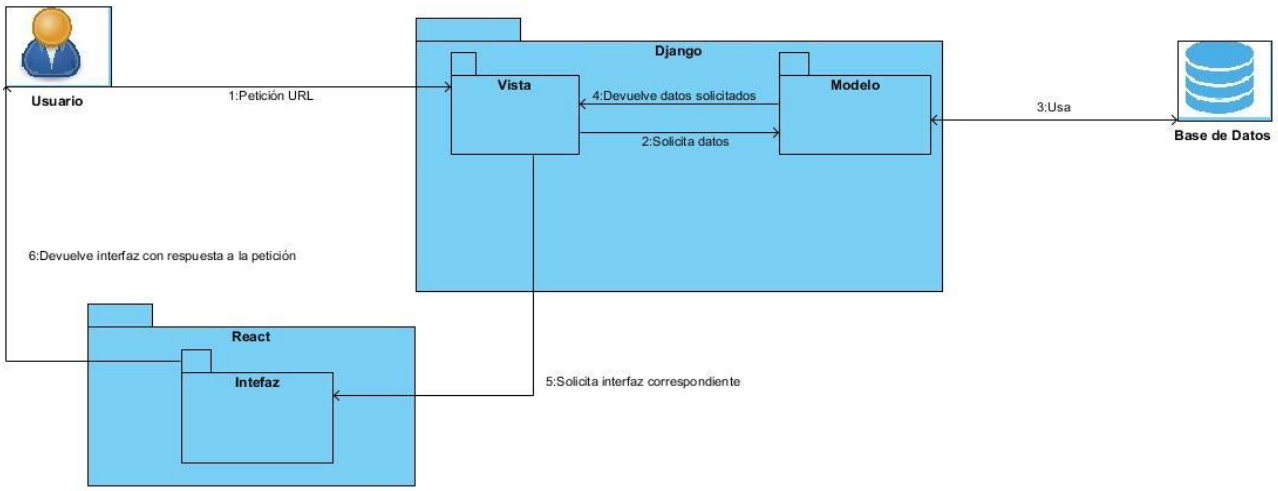


Figura 7: Arquitectura MVP. Elaboración propia

El paquete Modelo contiene la estructura de los datos como el tipo de ambigüedad y su descripción; a su vez utiliza la base de datos para almacenarlos. En el paquete Vista se encuentra las clases principales como AmbigüedadView y FicheroView. Para crear las interfaces con la que interactúa el usuario se utilizó React.

Patrones de diseño

Un patrón de diseño es una solución a un problema en un contexto particular. Además, facilitan la reutilización de diseños o arquitecturas de software que han tenido éxito y describen el esquema básico para estructurar subsistemas y componentes *(Macario 2018)*. Para el diseño de la solución se emplearon varios patrones de diseño los cuales se describe a continuación.

Patrones GRASP

Los patrones GRASP (General Responsibility Assignment Software Patterns) describen los principios fundamentales para asignar responsabilidades a los objetos *(Macario 2018)*. Dentro de los patrones GRASP a utilizar se encuentran:

Experto: este patrón ayuda a identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases. El patrón está presente en la clase AmbiguedadesView que contienen todos los métodos e información para tener poca dependencia del resto de las clases.

Creador: se encuentra en las clases que tienen la tarea de instanciar objetos de otras. El empleo de este patrón se evidencia en la clase AmbiguedadesView la cual se encarga de crear objetos y almacenarlo.

Controlador: este se encarga de coordinar el trabajo de otros. Este patrón se evidencia en la clase Main que brinda Django.

Alta cohesión: se utiliza para realizar un diseño que evite la sobrecarga de métodos. La clase tiene alta cohesión si todo lo que hace está bien delimitado dentro del mismo objeto. El patrón se evidencia en cada de una de las clases de la herramienta.

Bajo acoplamiento: es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que, en caso de producirse una modificación en alguna de ellas se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases este patrón se evidencia en cada de una de las clases de la herramienta.

Patrones GOF

Dentro de los patrones GOF (Gang of four) se encuentran tres grupos de patrones los de creación estructurales y de comportamiento. Se utilizará el siguiente patrón para el diseño de la solución.

Singleton: es un patrón de diseño de creación que permite asegurar que una clase tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia.*(Alexander 2021)*. Este patrón se evidencia en la clase Main de Django.

Descripción de las tarjetas CRC

Las tarjetas CRC (clase, responsabilidad y colaboración) son una metodología para el diseño de software orientado por objetos creada por Kent Beck y Ward Cunningham. Permite romper con el modo de procedimiento y de pensamiento para apreciar mejor la tecnología de objetos. También permiten que todo el equipo pueda contribuir al diseño del proyecto *(Alejandra 2022)* . A continuación, se describen algunas de las tarjetas, el resto se encuentra en el Anexo 6.

Tabla 13: Tarjeta CRC AmbiguedadesView

Tarjeta CRC	
Clase: AmbiguedadesView	
Responsabilidades	**Colaboración**
Detectar ambigüedades léxicas y sintácticas a partir de un texto recibido.	Detectar ambigüedades léxicas. Detectar ambigüedades sintácticas.

Tabla 14: Tarjeta CRC AmbiguedadesFicheroView

Tarjeta CRC	
Clase: AmbiguedadesFicheroView	
Responsabilidades	**Colaboración**
Detectar ambigüedades léxicas y sintácticas a partir de un documento cargado.	Detectar ambigüedades léxicas. Detectar ambigüedades sintácticas.

Estándares de codificación

Los estándares de código, son parte de las llamadas buenas prácticas. Estas son un conjunto no formal de reglas, que han ido surgiendo en las distintas comunidades de desarrolladores con el paso del tiempo y las cuales, bien aplicadas pueden incrementar notablemente, la calidad del código.*(Arthur 2020)*. El estándar de codificación para el desarrollo de la herramienta definido luego de un estudio realizado previamente es el **CamelCase**. Se decidió el uso de este estándar debido a que reduce el esfuerzo necesario para leer y entender el código fuete, además de que mejora la apariencia de este al no permitir abreviaturas.*(Keefe 2022)*

CamelCase es un estilo de escritura que se aplica a frases o palabras compuestas. El nombre se debe a que las mayúsculas a lo largo de una palabra se asemejan a las jorobas de un camello cuando la primera letra de cada una de las palabras es mayúscula.*(Keefe 2022).* Existen dos tipos de CamelCase el UpperCamelCase, cuando la primera letra de cada una de las palabras es mayúscula y el lowerCamelCase, igual que la anterior con la excepción de que la primera letra es minúscula. A continuación, algunas pautas utilizadas del estándar anteriormente mencionado.

- Los nombres de las clases serán en mayúscula, en caso de ser un nombre compuesto las siguientes palabras se escribirán de igual forma.
- Los nombres de los métodos serán con mayúscula, en caso de ser un nombre compuesto, la primera palabra será en minúscula y la siguiente en mayúscula.
- Los identificadores para las variables y los parámetros serán en minúsculas.
- Los nombres de variables o funciones deben ser lo suficientemente descriptivos no más de 15 caracteres.
- De haber comentarios serán en idioma español para una mejor comprensión.

2.4 Procesamiento del lenguaje natural

En este trabajo se aplicó una línea de procesamiento del lenguaje natural compuesta por tres técnicas de PLN: 1-) tokenización para dividir el texto en tokens, 2-) eliminación de ruido como, caracteres no deseados o espacios en blanco y números, 3-) clasificación para asignar una etiqueta por parte del discurso (POS), por ejemplo, sustantivo, verbo o pronombre, a cada token de cada frase. La Figura 8 muestra como ocurre el preprocesamiento haciendo uso de las técnicas mencionadas. Como entrada se tiene al documento de requisitos o al requisito escrito manualmente. Luego se pasa a la fase de procesamiento del contenido textual donde se aplican las técnicas de PLN que se muestran en la Figura 8 y definidas en el capítulo 1. Una vez terminado el proceso de PLN se determina que el requisito presenta ambigüedad.

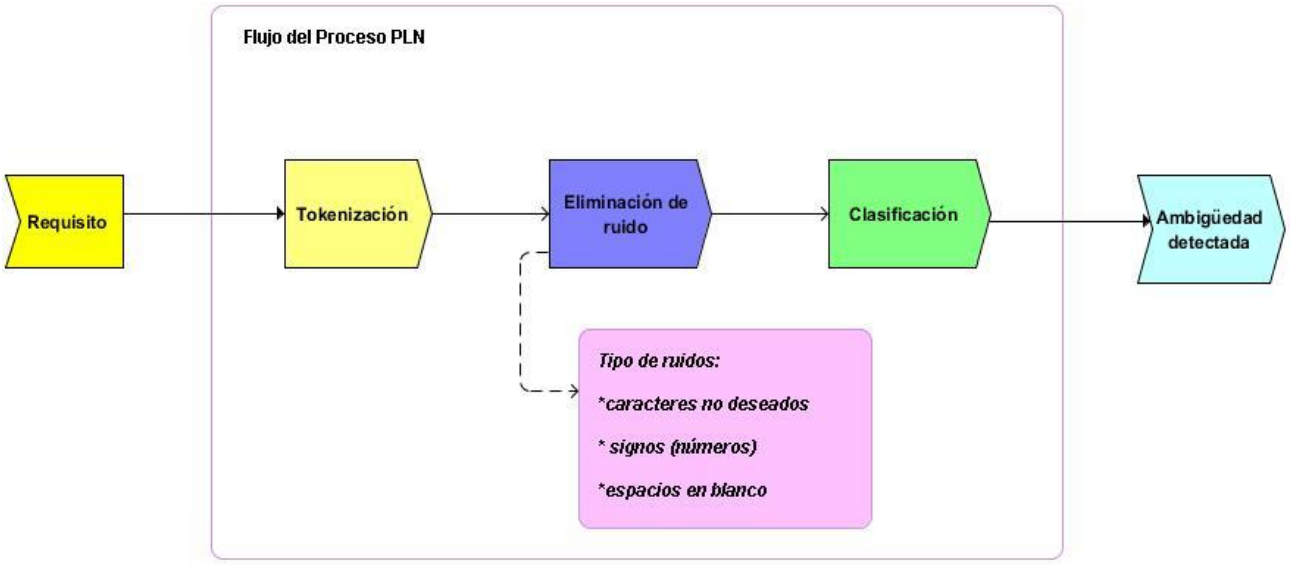


Figura 8: Flujo del proceso PLN

2.5 Algoritmos para la detección de ambigüedades

Para la detección de ambigüedad léxica y sintáctica en los requisitos se implementaron 2 algoritmos basados en reglas heurísticas y el modelo de calidad obtenido luego del estudio realizado en el capítulo 1. A continuación se presentan las reglas y algoritmos obtenidos.

Detección de ambigüedad léxica

Para comprobar la presencia de ambigüedad léxica en un requisito, se defino una regla. Sean A conjuntos de términos ambiguos porque tienen más de una interpretación, S un requisito, y sea T (S) cualquier secuencia de palabras del requisito. La regla es la siguiente:

R1: ∀a ∈ A, ∀t ∈ T (S), si t = a, marca t como ambigüedad léxica.

Si un requisito tiene asociado un término con más de un significado como el patrón anterior entonces el requisito está en presencia de ambigüedad léxica.

A continuación, el código para la detección de ambigüedad léxica.

1. def AddLexicaAmbiguity **(text)** → Entrada requisito redactado o documento de requisitos
2. lexicals = [ ]
3. for x in ListaPalabrasAmbiguas: → Recorrer lista de palabras ambiguas

4. if x in text:

Si la palabra(s) ambigua(s) pertenece a el requisito o documento de requisitos

5. lexical.append(“Ambiguedad lexica detectada en la cadena " + " " + "(" + x + ")")

6. return np.unique(lexicals)

Muestra si detecto o no ambigüedad

Detección de ambigüedad sintáctica

Para comprobar la presencia de ambigüedad sintáctica en un requisito, se definieron las siguientes reglas.

R1: (Token)* (CC) (Token.kind != "punct")* (CC) (Token)*

R2: (Token)* (CD) (Token.kind != "punct")* (CD) (Token)*

R3: (Token)* (P) (Token.kind != "punct")* (P) (Token)*

R4: (JJ) (NN | NNS) (y | o) (NN | NNS)

La regla 1 busca al menos dos apariciones de conjunciones copulativas (y, e, ni, que) en el requisito. Estas conjunciones no deben estar separadas por signos de puntuación (por ejemplo, comas, punto y coma, guiones, etc.). Las comas y otros tipos de puntuación pueden aclarar la estructura sintáctica. La regla 2 por su parte busca la presencia de al menos dos conjunciones disyuntivas (o, u, sea, bien), aunque el funcionamiento es similar a la regla 1. Si un requisito contiene al menos una preposición separable (a, con, de, en) entonces estamos en presencia de la regla 3. Si un adjetivo (JJ) precede a un par de sustantivos singulares (NN) o plurales (NNS), unidos por "y" o "o" estamos en presencia de la regla 4 una ambigüedad sintáctica coordinativa.

El código para detectar ambigüedad sintáctica se presenta a continuación.

1.def SyntacticAnalissys(**text**):

Entrada requisito redactado o documento de requisitos

2.sintactic = []

3. c = [' y ', ' e ' , ' ni ', ' que ']

Conjunciones y preposiciones separables

```
cp = [' o ', ' u ', ' sea ', ' bien ']
cps = [' a ', ' con ', ' de ', ' en ' ]
```

```
4.cont = 0
  cont2 = 0
  cont3 = 0
```

Contadores

```
5. palabras = word_tokenize(text)
  print(palabras)
```

Tokenizar las palabras

```
6. for sentence in blob.sentences:
        for j in sentence:
            if j == '.' or j == ',' or j == ';':
                cont = 0
```

Dividir los requisitos en oraciones

```
7.   for y in c:
            if y in sentence:
                cont = cont + 1
                if cont >= 2:
    sintactic.append(
```

Buscar en las oraciones si aparece más de una conjunción copulativa

'posible ambigüedad sintáctica: El requisito presenta más de una conjunción copulativa (y, e, ni, que) ')

```
8. for sentence in blob.sentences:
        for r in cp:
            if r in sentence:
                cont2 = cont2 + 1
                if cont2 >= 2:
                    sintactic.append(
                    'posible ambigüedad sintáctica: El requisito presenta más de una conjunción disyuntiva (o, u, sea, bien) ')
```

Buscar en las oraciones si aparece más de una conjunción disyuntiva

```
9.   for sentence in blob.sentences:
        for z in cps:
            if z in sentence:
```

Buscar en las oraciones si aparece más de una preposición separable

```
cont3 = cont3 + 1
if cont3 >= 2:
sintactic.append( 'posible ambigüedad sintáctica: El requisito presenta más de una preposición separable  ( a, con, de, en) ')
10. return sintactic
```

Muestra si detecto o no ambigüedad sintáctica

2.6 Descripción de la base de datos a utilizar

Los algoritmos que emplean las reglas para el análisis de las ambigüedades sintácticas y léxicas presentes en los requisitos hacen uso de una base de datos para almacenar datos del requisito de software o el documento de requisitos. La base de datos está compuesta por las tablas AMBIGUEDADES_ambiguedad y AMBIGUEDADES_file.

La tabla AMBIGUEDADES_ambiguedad contiene el requisito,el tipo y descripción de la ambigüedad mientras que la tabla AMBIGUEDADES_file contiene el documento de requisitos. Las tablas fueron introducidas por los autores del presente trabajo, con el propósito de hacer posible el funcionamiento de la herramienta propuesta. Una vez que el usuario decide procesar el requisito o el documento de requisito ambas tablas capturan la información correspondiente a ellas. En la siguiente Figura 9 se muestra el diagrama de la base de datos.

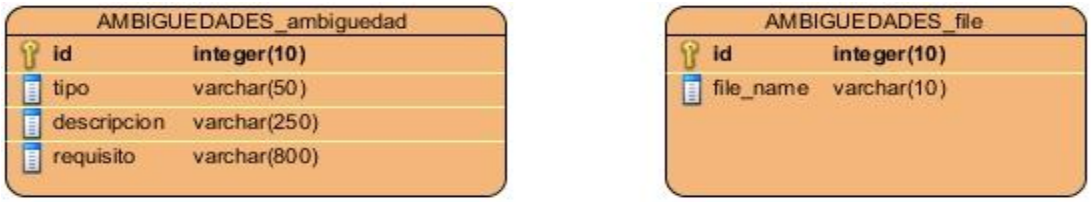


Figura 9: Base de datos

Conclusiones parciales

- El uso de la metodología XP como guía del proceso de desarrollo de la herramienta permitió realizar un trabajo organizado y estructurado, generando los artefactos correspondientes a las fases de planificación y diseño.

- La recolección y limpieza de los requisitos de software de los proyectos de la UCI permitió la confección de un conjunto de datos que posee 19 357 requisitos de software, que constituye una línea base para futuras investigaciones.
- El empleo del patrón arquitectónico modelo vista plantilla y patrones de diseño, garantizan obtener una solución de software con poca dependencia entre clases, flexible al mantenimiento, cambios y escalabilidad.
- El flujo del proceso de PLN compuesto por las técnicas de tokenización, eliminación de ruido y clasificación, además de los algoritmos que implementan cada una de las reglas heurísticas definidas basadas en el criterio de no ambigüedad, permiten detectar ambigüedades léxica y sintáctica en los requisitos de software.

Capítulo III. Implementación, prueba y validación de la herramienta

Introducción

En el presente capítulo se describe la fase codificación y prueba de la metodología de desarrollo empleada. En la cual una vez implementada la herramienta se le realizan las pruebas y chequeos antes de ser entregada al cliente. Además, se exponen los artefactos obtenidos en estas fases, tales como: las tareas de ingeniería y el acta de aceptación. En el capítulo también se presenta la validación de los resultados de la herramienta en relación al cumplimiento del objetivo general de la investigación.

3.1 Fase de codificación

En esta fase se realiza la implementación de las HU. Para un desarrollo adecuado y mayor organización a la hora de implementar las HU se elaboran las tareas de ingeniería. Estas tareas son descritas por los desarrolladores, se realizan en un lenguaje técnico y fácil de comprender para los que la crean.

A continuación, se muestra cada iteración donde fueron implementadas las HU con sus respectivas tareas de ingeniería.

3.1.1 Primera Iteración

En esta iteración se desarrollan las HU 1, 2, 3, 6 y 7 relacionadas con el documento de requisitos y la redacción de requisitos. Para ello se realizaron las tareas que a continuación se muestran y el resto se encuentran en el Anexo 7**:**

Tabla 15: TI. Diseño de importar documento

Tarea	
Número de tarea: 1	Número de HU: 1
Nombre de la tarea: diseño de importar documento	
Tipo de tarea: Diseño	Estimación: 2 días
Descripción: Se diseñó un input de tipo file con el nombre “examinar” que al presionarlo abre una ventana emergente del explorador de archivos que permite navegar por el dispositivo y seleccionar el archivo deseado.	

Tabla 16: TI. Importar documento

Tarea	
Número de tarea: 2	Número de HU: 1
Nombre de la tarea: importar documento	
Tipo de tarea: Desarrollo	Estimación: 3 días
Descripción: Se implementó el método cargarDocumento() que es el encargado de guardar el archivo seleccionado por el usuario para su posterior procesamiento.	

3.1.2 Segunda Iteración

En esta iteración se desarrollan las HU 4 y 5 relacionadas con detectar y mostrar ambigüedades léxicas y sintácticas. Para ello se realizaron las tareas que a continuación se muestran y el resto se encuentran en el Anexo 7:

Tabla 17: TI. Diseñar detectar ambigüedades léxica y sintáctica

Tarea	
Número de tarea: 4	Número de HU: 5
Nombre de la tarea: diseñar detectar ambigüedades léxica y sintáctica	
Tipo de tarea: Diseño	Estimación: 5 días
Descripción: Se diseñaron botones con los nombres “Procesar requisito” y “Procesar columna” que al presionarlo analiza la columna del documento de requisitos o el requisito redactado manualmente.	

Tabla 18: TI. Detectar ambigüedades léxicas

Tarea	
Número de tarea: 5	Número de HU: 5
Nombre de la tarea: detectar ambigüedades léxicas	
Tipo de tarea: Desarrollo	Estimación: 5 días
Descripción: Se implementaron los métodos analisisLexico() y analisisSintactico () que	

se encarga de encontrar las ambigüedades léxicas y sintácticas presentes en los requisitos de software que se procesen.

3.2 Validación de la herramienta

Para dar cumplimiento al objetivo de la presente investigación se realiza la evaluación de un conjunto de criterios de medida definidos por los autores del presente trabajo. Estos permiten validar la propuesta solución en función de comparar cómo a través de la solución obtenida se resuelve la problemática planteada.

Criterios de medida definidos:

- Exhaustividad: para informar la **cantidad** que los algoritmos son capaces de detectar ambigüedades.
- Precisión: para medir la **calidad** de los algoritmos de detección de ambigüedades.
- F1: se utiliza para combinar las medidas de precisión y exhaustividad en un sólo valor. Esto es práctico porque hace más fácil el poder comparar el rendimiento combinado de la precisión y la exhaustividad entre varias soluciones.

A continuación, en la tabla 19, se evalúan cada uno de los criterios de medida definidos. La evaluación se realiza con un conjunto de 100 requisitos pertenecientes al conjunto de datos de requisitos desarrollado por los autores de la investigación. De los 100 requisitos de muestra, se detectó que de 94 instancias clasificadas como que contienen ambigüedad, 54 fueron clasificadas mal en que no contenían ambigüedad y de 6 que si contenían ambigüedad 5 fueron clasificados de mal que no tenían ambigüedad. En el Anexo 8 se encuentra la explicación del uso de estas métricas de manera manual en los 100 requisitos de prueba.

Tabla 19: Evaluación de SIDARES

Criterios de Medida	SIDARES
Precisión	98%
Exhaustividad	91%
F1	94%

El 98% de precisión significa que el algoritmo se equivocará un 2% de las veces cuando detecte las ambigüedades. El 91% de exhaustividad informa del porcentaje de clases positivas que ha sido capaz de identificar correctamente (verdaderos positivos, verdaderos negativos), es decir la capacidad de detectar ambigüedades o no en los requisitos correctamente. Por último, la métrica F1 combina la precisión y la exhaustividad en una sola medida. Estos criterios de medida analizados en la tabla anterior, demuestran que la solución obtenida permite detectar ambigüedades léxicas y sintácticas presentes en los requisitos de software, cumpliendo así con el objetivo general de la presente investigación.

3.3 Fase de prueba

Las pruebas, es una etapa del proceso de desarrollo del producto en la cual se asegura la calidad y el buen funcionamiento del producto. La metodología XP propone la realización de las pruebas unitarias y de aceptación (Pressman 2020), descritas a continuación.

3.3.1Pruebas unitarias

Las pruebas unitarias son una de las piedras angulares de XP. Todos los módulos deben de pasar estas pruebas antes de ser liberados o publicados (Hoda 2019). Consisten en verificar el comportamiento de las unidades más pequeñas de la aplicación (clases, métodos o módulos). El propósito de estas pruebas es proporcionar una retroalimentación al desarrollador sobre el diseño y la implementación del software (Hoda 2019).

Con el método de caja blanca se verifica el código que permite que el producto funcione correctamente. Como parte de este método se decidió probar los métodos de la herramienta propuesta. Para realizar las pruebas se hizo uso del marco de Python debido a que ya viene integrado a este.

A continuación, se muestra las pruebas unitarias realizadas al caso de prueba “Detectar ambigüedades”.

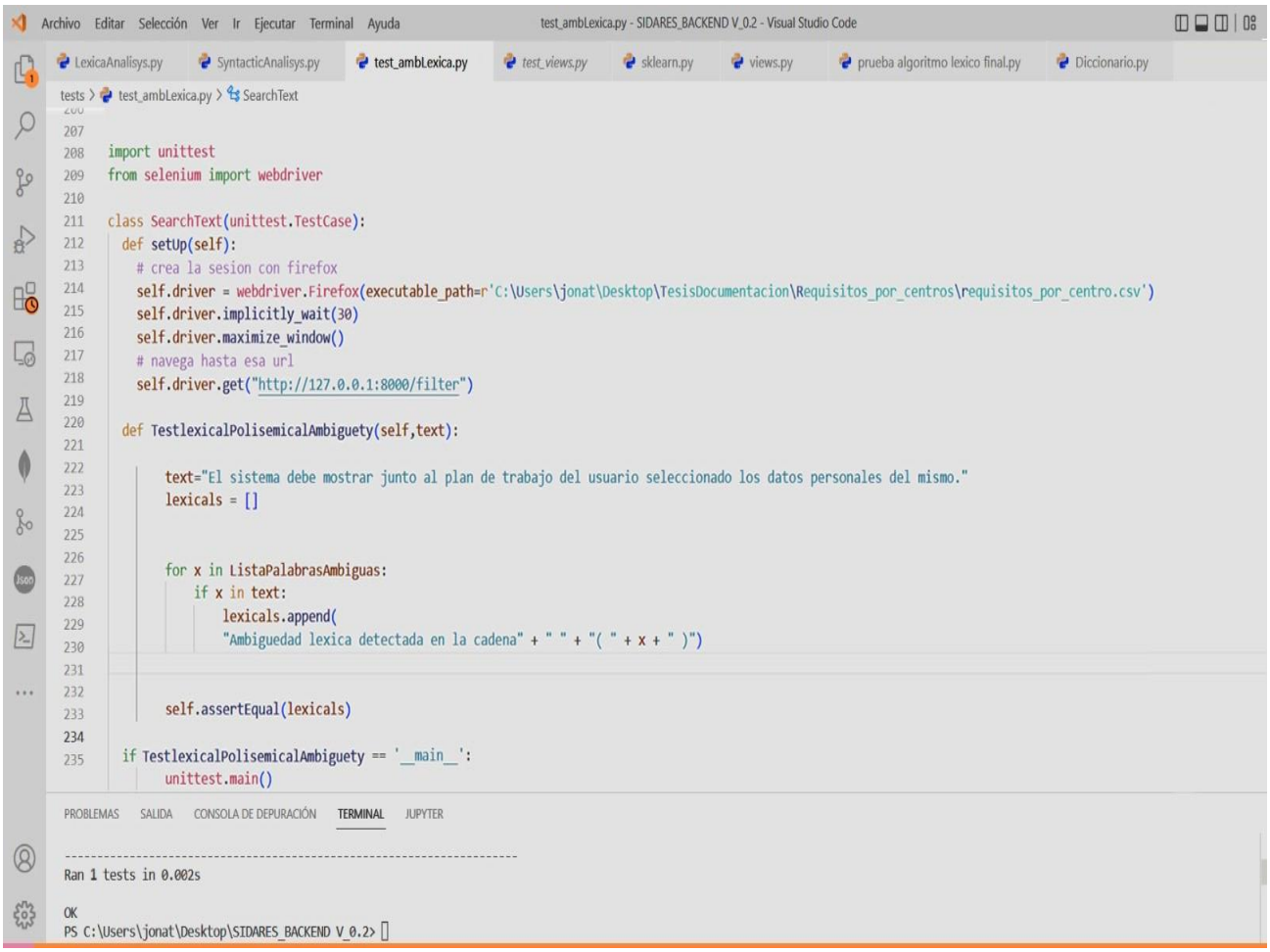


Figura 10: Detectar ambigüedad léxica

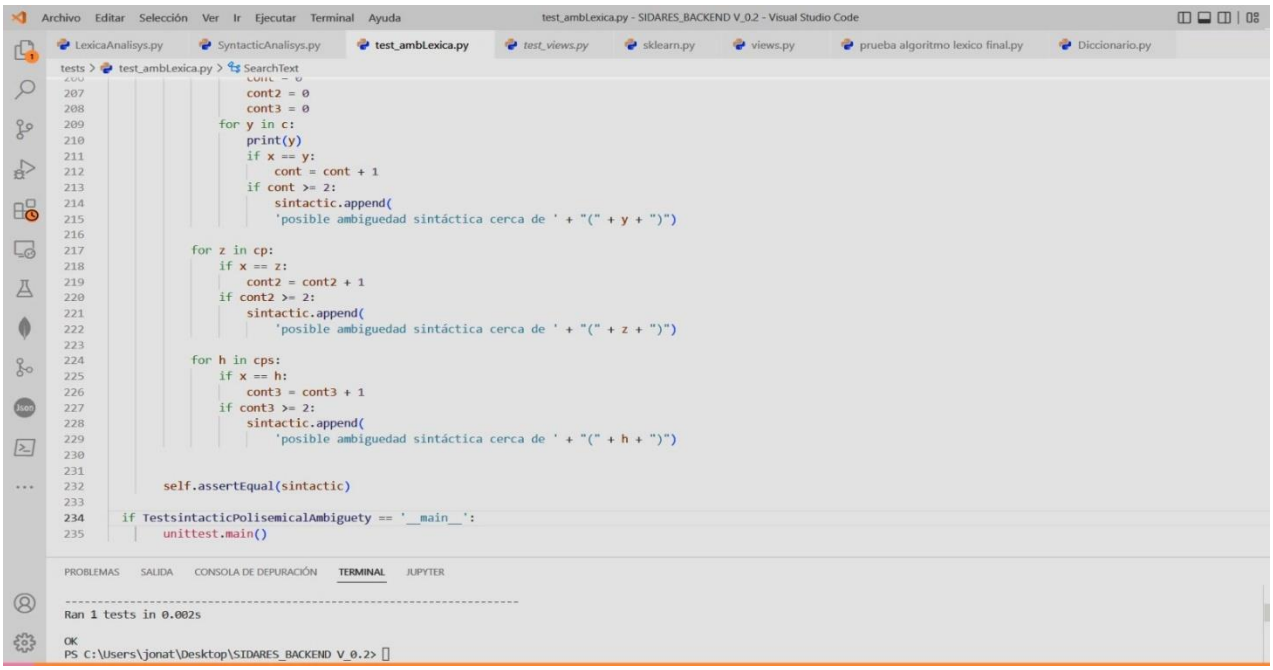


Figura 11: Detectar ambigüedad sintáctica

3.3.2 Pruebas de aceptación

Este tipo de prueba constituye el último nivel de prueba del producto, para crear un software confiable y de uso apropiado. (Pressman 2020). Para realizar estas pruebas se confeccionó un caso de prueba de aceptación para cada HU. La tabla 20 ilustra el caso de prueba de aceptación correspondiente a la HU detectar ambigüedades, el resto de estas pruebas se encuentran en el Anexo 9.

Tabla 20: PA. Detectar ambigüedades

Caso de Prueba de Aceptación
Código: P5HU Número de HU: 5
Nombre: Detectar ambigüedades léxica y sintáctica
Descripción: Se verifica el correcto funcionamiento del requisito funcional, siendo este capaz de detectar ambigüedades léxicas y sintácticas
Condiciones de ejecución: Se debe tener un documento cargado o los requisitos redactados en la aplicación. En caso de ser un documento, se debe haber seleccionado una columna previamente.
Entrada/Pasos de ejecución: 1. Presionar el botón PROCESAR COLUMNA O PROCESAR REQUISITO
Resultado esperado: La herramienta detecta ambigüedades léxicas y sintácticas
Evaluación de la prueba: Satisfactorio

Se realizó una entrevista a la (M.Sc. Aymara Marín Díaz), directora de calidad de la UCI, con el objetivo de ratificar la necesidad de que se desarrolle una herramienta para analizar la calidad de los requisitos enfocada en la detección de la ambigüedad. Además de que brinde su valoración a la herramienta desarrollada. Los resultados obtenidos fueron satisfactorios, avalados por la especialista entrevistada como un aporte importante para el desarrollo de la informática en Cuba. El Acta de Aceptación del Producto (Anexo 10) se generó al culminar las pruebas de aceptación con la cliente y la entrevista se encuentra en el Anexo 11.

3.3.3 Pruebas funcionales

Con el método de caja negra se examina si la herramienta funciona y cumple con su objetivo para satisfacer al cliente. Como parte de este método se realizó las pruebas funcionales por parte de la cliente para dar solución a las no conformidades encontradas en cada iteración. Partición de equivalencia fue la técnica empleada para realizar esta prueba con el objetivo de detectar errores en las funcionalidades y validar los requisitos de la propuesta solución. A continuación, la tabla 21 muestra un ejemplo esta prueba al de caso de prueba “Detectar ambigüedad” usando la técnica mencionada.

Tabla 21: Caso de prueba

Escenario	Descripción	Requisito	Respuesta del sistema	Resultado de la prueba
EC 1.1 detectar ambigüedad léxica o sintáctica correctamente	Detecta ambigüedad léxica o sintáctica en el requisito o documento de requisitos si se presiona el botón “PROCESAR	Válido	El sistema muestra una lista con el tipo de ambigüedad que presenta el requisito escrito manualmente o los	Satisfactoria
		Hay un requisito ambiguo /el documento de requisitos contiene requisitos ambiguos ejemplo “El sistema debe ser seguro y consistente y amigable”		

	REQUISITO" o "PROCESAR DOCUMENTO"		requisitos del documento de requisitos	
EC1.2 campos vacíos	Presiona el botón "PROCESAR REQUISITOS" estando el campo vacío	Inválido	El sistema muestra el mensaje "debe escribir un requisito"	Satisfactoria el sistema alerto al usuario del problema
EC 1.3 EC 1.3 Sin documento	Presiona el botón "PROCESAR DOCUMENTO" sin haber importado alguno	Inválido	El sistema muestra el mensaje "seleccione un documento"	Satisfactoria el sistema alerto al usuario del problema

Con el objetivo de comprobar que las funcionalidades de la herramienta se realizaron correctamente y responden a las necesidades del cliente, se probó la funcionalidad de esta en dos iteraciones. En la primera se detectaron un total de 7 No Conformidades (NC) (Anexo 12), predominando entre ellas de tipo funcionales como por ejemplo el mal funcionamiento del botón "Eliminar lista de ambigüedades"**.** Al finalizar la iteración todas estas NC quedaron resueltas.

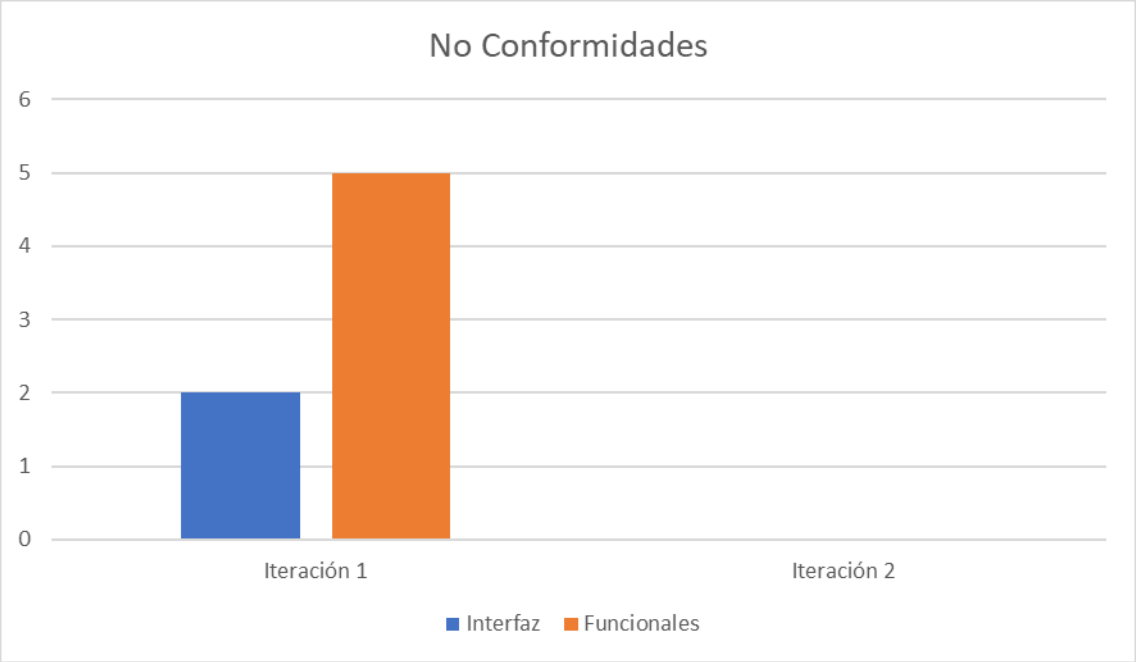


Figura 12: No Conformidades

En la segunda iteración los resultados fueron satisfactorios, obteniéndose cero NC). La Figura 10 muestra los resultados de estas pruebas, teniendo en cuenta las NC encontradas.

Conclusiones parciales

- En este capítulo fueron generadas las tareas de ingeniería, para una mejor organización a la hora de implementar las HU.
- La aplicación de pruebas como las unitarias y de aceptación a la solución propuesta, permitió la obtención de un software funcional que responde a las especificaciones del cliente, avalado en cada caso por el acta de aceptación.
- Los resultados obtenidos en la validación de la herramienta a partir de la definición de un conjunto de métricas, evidencia el cumplimiento del objetivo de la presente investigación.
- La realización de las pruebas unitarias y de aceptación permitió detectar varias no conformidades que fueron resueltas de forma satisfactoria,

logrando con esto que la herramienta se ajuste lo más posible a las expectativas del cliente.

Conclusiones generales

El desarrollo de la presente investigación permitió arribar a las siguientes conclusiones generales:

- La elaboración del diseño teórico evidenció la ausencia de una solución para la detección de ambigüedad de requisitos de software en español.
- La investigación acerca de las tendencias tecnológicas actuales y estrategias utilizadas para el análisis de la ambigüedad en los requisitos de software, mostró elementos a tener en cuenta en la solución, como fueron: las bibliotecas de Python, técnicas de PLN y reglas heurísticas.
- El estudio de los criterios de calidad en los requisitos de software y del idioma español, así como las soluciones existentes en la actualidad para la identificación de ambigüedades, constituyeron la base para el diseño de los algoritmos basados en reglas heurísticas que fundamentan el desarrollo de la herramienta propuesta.
- El empleo de patrones de diseño, el uso de una arquitectura modelo vista plantilla y la elaboración de algoritmos basados en reglas heurísticas definidas a partir de las características del lenguaje español, posibilitaron obtener una herramienta que detecta ambigüedades léxicas y sintácticas en requisitos en idioma español.
- El conjunto de datos obtenidos constituye una línea base para futuras investigaciones.
- Las pruebas realizadas permitieron verificar que la herramienta desarrollada cumple con los requisitos propuestos, validándose además la calidad de la solución propuesta.

Recomendaciones

Luego de concluir con el desarrollo de la herramienta propuesta se recomienda lo siguiente:

- Incorporar a la herramienta nuevos algoritmos que permitan detectar otros tipos de ambigüedades como la semántica o la pragmática.
- Incorporar a la herramienta un diccionario con mayor número de términos ambiguos para mejorar la eficiencia de los algoritmos de detección de ambigüedades.
- Extender el uso de la herramienta a otras entidades que desarrollan softwares para mejorar la calidad de sus requisitos.
- Optimizar los algoritmos de detección de ambigüedades para una mayor rapidez de respuesta de la aplicación.

Acrónimos

HU: Historia de usuario

IR: Ingeniería de requisitos

LN: Lenguaje natural

NC: No conformidad

PLN4IR: Procesamiento del lenguaje natural para la ingeniería de requisitos

PLN: Procesamiento del lenguaje natural

R: Reglas

RF: Requisito funcional

RNF: Requisitos no funcional

UCI: Universidad de las Ciencias Informáticas

UML: Lenguaje unificado de modelado (siglas en inglés)

Referencias bibliográficas

1. Alejandra J Serrano A. 2022. «Desarrollo de un sistema de Gestión y Control Administrativo para la Coordinación de Servicio Comunitario de la Universidad de Oriente- Núcleo Monagas».
2. Abualhaija, Sallam, Davide Fucci, y Fabiano Dalpiaz. 2020. «3rd Workshop on Natural Language Processing for Requirements Engineering (NLP4RE'20) », 3.
3. Alexander Shvets. 2021. «Sumérgete en los PATRONES DE DISEÑO».
4. Alexandra Carranza. 2022. «¿Que es Java? - Guía de Java empresarial para principiantes - AWS». 2022. https://aws.amazon.com/es/what-is/java/.
5. Ambriola Vincenzo, Gervasi Vincenzo.2018. Processing Natural Language Requirements.
6. Antonio Javier Gallego. 2020. «Bootstrap 4».
7. Arendse Brian,Lucassen Gram.2016.Toward tool Mashups:Comparing and Combinig NLPRE tools.
8. Arthur Hiken. 2020. «Una onza de prevención: seguridad y protección a través de estándares de codificación de software», 2020.
9. Aschauer, Bernd, Lars Baumann, Peter Hruschka, Kim Lauenroth, Markus Meuten, Sacha Reis, y Gareth Rogers. 2018. «IREB Certified Professional for Requirements Engineering-Advanced Level RE@Agile-Syllabus». http://www.ireb.org.
10. Ayala de la Vega, Joel. 2018. «Ingeniería en computación lenguaje de programación estructurada. 2018B».
11. Bahit, Eugenia. 2020. Python para principiantes.
12. Boduch, Adam, y Roy Derks. 2020. React and React Native: A Complete Hands-on Guide to Modern Web and Mobile Development with React.Js. Third edition. Birmingham, U.K.: Packt Publishing Ltd.
13. Bustos Ricardo Gacitúa .2018. Procesamiento de Lenguaje Natural en Ingeniería de Requisitos: Contribuciones Potenciales y Desafíos de Investigación.
14. Casamayor Agustin, Godoy Daniela, Campo Marcelo.2018. Functional grouping of natural language requirements for assistance in architectural software design. 78-86.VOL 30.DOI 10.1016/j.knosys.2011.12.009.

15. Cassanelli, Gerardo, Rodrigo directora, y Ana Haydee Di Iorio. 2019. «Proyecto final NLP aplicado a análisis de texto».
16. Cuevas Álvarez, Alberto. 2020. Comenzar a programar con python 3. Place of publication not identified: lulu com.
17. del Águila Cano, Isabel María. 2022. Fundamentos de ingeniería de los requisitos. Universidad de almería.
18. Dr. Henderi Dr. Untung Rahardja Efana Rahwanto, M.T.I. 2022. Uml powered design system using visual paradigm.
19. Durand, Sandra Wong. 2017. «discapacidad e integridad Manual Autoformativo Interactivo Análisis y requerimientos de software». http://www.continental.edu.pe/.
20. Duygu Altinok.2021. Mstering spacy an end to end practical guide to implementing NLP applications using the Python ecosystem.
21. Edgardo Stasi.2020. Programación en Python Proyectos prácticos.
22. Espinoza Mina, Marcos Antonio. 2018. «Weka, áreas de aplicación y sus algoritmos: una revisión sistemática de literatura». Revista Científica Ecociencia 5 (diciembre): 1-26. https://doi.org/10.21855/ecociencia.50.153.
23. Falessi Davide, Cantone Giovanni, Canfora Gerardo.2018. Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques. IEEE Transactions on Software Engineering. 18-44.VOL 39.DOI 10.1109/TSE.2011.122.
24. Fernando Luna,Claudio Peña Millahual ,Matias Iacono. 2018. «Programación web full stack desarrollo frontend y backend».
25. Ferrari, Alessio, Giorgio O. Spagnolo, Antonella Fiscella, y Guido Parente. 2019. «QuOD: An NLP Tool to Improve the Quality of Business Process Descriptions». En Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 11865 LNCS:267-81. Springer Verlag. https://doi.org/10.1007/978-3-030-30985-5_17.
26. Fuentes, Allan Pierra. 2022. «Adopción de una cultura Ágil en el desarrollo de proyectos de aplicaciones para Internet en Cuba», 2022. https://rcci.uci.cu/?journal=rcci&page=article&op=view&path%5B%5D=2545.

27. Glinz, Martin, Hans Van Loenhoud, Stefan Staal, y Stan Bühne. 2020. «Handbook for the CPRE Foundation Level according to the IREB Standard Education and Training for Certified Professional for Requirements Engineering (CPRE) Foundation Level Terms of Use».
28. Gnesi, Stefania, y Gianluca Trentanni. 2019. «QuARS a NLP Tool for Requirements Analysis». http://quars.isti.cnr.it/.
29. Guadalupe Sánchez Escribano Gonzalo, María. 2019. «definición de requisitos funcionales bajo especificación ieee para un sistema de ingeniería».
30. Guzmán-Luna, J.A, Gómez Arias, y & Vélez-Carvajal. 2018. «Un modelo de procesamiento de lenguaje natural para la detección de errores en requisitos de software».
31. Harrison, Matt, y Theodore Petrou. 2020. Pandas 1.x Cookbook: Practical Recipes for Scientific Computing, Time Series Analysis and Exploratory Data Analysis Using Python. Second edition. Birmingham: Packt Publishing.
32. Haverbeke, Marijn. 2018. Eloquent JavaScript : a modern introduction to programming.
33. Hayman Oo, Khin, Azlin Nordin, Amelia Ritahani Ismail, y Suriani Sulaiman. 2018. «An Analysis of Ambiguity Detection Techniques for Software Requirements Specification (SRS)». International Journal of Engineering & Technology. Vol. 7. www.sciencepubco.com/index.php/IJET.
34. Hoda, Rashina, ed. 2019. Agile Processes in Software Engineering and Extreme Programming – Workshops: XP 2019 Workshops, Montréal, QC, Canada, May 21–25, 2019, Proceedings. Vol. 364. Lecture Notes in Business Information Processing. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-30126-2.
35. Hung, Jason C., Neil Y. Yen, y Jia-Wei Chang, eds. 2022. Frontier Computing: Proceedings of FC 2021. Singapore: Springer.
36. Jane Radatz, Chairperson. 2018.«IEEE_Standard_Glossary_of_Software_Engineering_Terminology .pdf».
37. Jhon Hunter,Firing Eric.2018.Matplotlib development team.https://matplotlib.org

38. Jolly, Kevin. 2018. Machine Learning with Scikit-Learn Quick Start Guide: Classification, Regression, and Clustering Techniques in Python. Birmingham, UK: Packt Publishing.
39. Kasser, Joseph E, Xuan-Linh Tran, y Simon P Matisons. 2018. «Prototype Educational Tools for Systems and Software (PETS) Engineering». 14 th Annual AAEE Conference. Vol. 532.
40. Keefe Denning. 2022. «¿por Que Usar Camelcase?» 2022. https://mejorsoftware.info/app33/82199/por-que-usar-camelcase.
41. Kumar Gupta, Ashok, Aziz Deraman, y Shams Tabrez Siddiqui. 2019. «A Survey Of Software Requirements Specification Ambiguity» 14 (17). www.arpnjournals.com.
42. López Natalia.2019. La interpretación subjetiva de la ambigüedad léxica: una aplicación lexicográfica.
43. Luis Salcedo.2020. Python.https://pythondiario.com /
44. «Macario Polo Usaola-Patrones GRASP 1 Patrones GRASP». 2018.
45. Mckinney, Wes. 2018. «Python for Data Analysis».
46. Miguel Casado, Gregorio de, Jorge Júlvez Bueno, y Jorge Gracia del Rio. 2021. Introducción a la programación C++ para ingenieros. Zaragoza: Prensas de la Universidad de Zaragoza.
47. Miguel de Icaza.2022. Tensor Flow. https://www.TensorFlow.org
48. Marcos Marino.2022.PyTorch.
49. Morales-Ramirez, Itzel, Fitsum Meshesha Kifetew, y Anna Perini. 2019. «Speech-Acts Based Analysis for Requirements Discovery from Online Discussions». Information Systems 86 (diciembre): 94-112. https://doi.org/10.1016/j.is.2018.08.003.
50. Nader, Rad K. 2019. Los Fundamentos De Agile Scrum. Van Haren Publishing.
51. Naeem, Afrah, Zeeshan Aslam, y Ali Shah. 2019. «Analyzing Quality of Software Requirements; A Comparison Study on NLP Tools».
52. Oliphant, Travis E. 2018. Guide to NumPy. 2nd edition. Austin, Tex.: Continuum Press.
53. Ortuño Sánchez, Yamila, Ing Yordanis, García Leiva, Msc Yarina, Amoroso Fernández, Ing Reinier, y Silverio Figueroa. 2016. «Herramienta informática para la

identificación de la ambigüedad en textos de la legislación cubana Autor: Tutores: Cotutor: “Año 58 de la Revolución”».

54. Osama, Mohamed, Aya Zaki-Ismail, Mohamed Abdelrazek, John Grundy, y Amani Ibrahim. 2020. «Score-Based Automatic Detection and Resolution of Syntactic Ambiguity in Natural Language Requirements». En Proceedings - 2020 IEEE International Conference on Software Maintenance and Evolution, ICSME 2020, 651-61. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ICSME46990.2020.00067.
55. Osman, Mohd Hafeez, y Mohd Firdaus Zaharin. 2018. «Ambiguous software requirement specification detection: An automated approach». En Proceedings - International Conference on Software Engineering, 33-40. IEEE Computer Society. https://doi.org/10.1145/3195538.3195545.
56. «Overti». 2022.
57. Paul Jasmin Rani, Jason Bakthakumar, Praveen Kumaar.B, Praveen Kumaar.U, Santhosh Kumar.2018. Voice Controlled Home Automation System Using Natural Language Processing (Nlp) And Internet Of Things (Iot)
58. Peng Qi,* Timothy Dozat,* Yuhao Zhang,* Christopher D. Manning. 2018. «Universal Dependency Parsing from Scratch».
59. Pressman, Roger S. 2020. Ingeniería del software : un enfoque práctico. McGraw-Hill.
60. Ramírez, Denniye Hinestroza, y Juan Manuel Cárdenas. 2018. «El Machine Learning A Través De Los Tiempos, Y Los Aportes A La Humanidad», 17.
61. Reese Richard M, Ashish Singh Bhatia.2018. Natulal language processing with Java second edition.
62. Reyes Sandler, Joaquín Ramón. 2018. Programación en C++: aprende a programar en C++. 3a edición. Place of publication not identified: IT Campus Academy.
63. Ricardo Javier Celi-Parraga., 1,, Eleanor Alexandra Varela-Tapia., 2, , Iván Leonel Acosta_Guzmán., y 3 & Nestor Rafael Montaño-Pulzara. 2021. « Técnicas de procesamiento de lenguaje natural en la inteligencia artificial conversacional textual».
64. Robert Hundt.2022.Quantum Computing for Programmers.

65. Sabriye, Ali Olow Jim'Ale. 2018. «An approach for detecting syntax and syntactic ambiguity in software requirement specification». . . Vol., n.º 8: 11.
66. Samuel Burns.2019.Natural Lenguage Processing a quick introduction to NLP with python and NLTK.
67. Saul Garcia M. 2019. «Introducción a Django - Aprende sobre desarrollo web | MDN». 2019. https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Introduction.
68. Sergio Delgado Quintero. 2022. «aprendepython.pdf».
69. Sommerville, Ian, Víctor Campos Olguín, y Sergio Fuenlabrada Velázquez. 2011. Ingeniería de software. Pearson Educación de México.
70. Soren Lauesen. 2020. Requirements Engineering: Foundation For Software Quality: 18th International Working Conference.
71. Timarán Pereira, Silvio Ricardo, Isabel Hernández Arteaga, Segundo Javier Caicedo Zambrano, Arsenio Hidalgo Troya, y Juan Carlos Alvarado Pérez. 2018. Descubrimiento de patrones de desempeño académico con árboles de decisión en las competencias genéricas de la formación profesional. Universidad Cooperativa de Colombia. https://doi.org/10.16925/9789587600490.
72. Tjong Fatimah, Berry Daniel M.2018. LNCS 7830 - The Design of SREE — A Prototype Potential Ambiguity Finder for Requirements Specifications and Lessons Learned.
73. Vaz, Eduardo. 2018. «Delivering Better Projects on Time by Ensuring Requirements Quality Upfront». INCOSE International Symposium 28 (julio): 575-86. https://doi.org/10.1002/j.2334-5837.2018.00501.x.
74. Wesley Clark. 2020. Metodología ágil: Una guía para principiantes sobre el método y los principios ágiles.
75. Yadav Apurwa, Patel Aarshil, Shah Manan.2021. A comprehensive review on resolving ambiguities in natural language processing. 85-92.VOL 2. DOI 10.1016/j.aiopen.2021.05.001.
76. Yang, Xin-She, Simon Sherratt, Nilanjan Dey, y Amit Joshi, eds. 2022. Proceedings of Sixth International Congress on Information and Communication Technology: ICICT 2021, London. Volume 3. Singapore: Springer.

77. Zait Fatima, Zarour Nacereddine .2018. Addressing Lexical and Semantic Ambiguity in Natural Language Requirements.
78. Zhao, Liping, Waad Alhoshan, Alessio Ferrari, y Keletso J. Letsholo. 2022. «Classification of Natural Language Processing Techniques for Requirements Engineering», abril. http://arxiv.org/abs/2204.04282.
79. Zhao, Liping, Waad Alhoshan, Alessio Ferrari, Keletso J. Letsholo, Muideen A. Ajagbe, Erol Valeriu Chioasca, y Riza T. Batista-Navarro. 2021. «Natural Language Processing for Requirements Engineering». ACM Computing Surveys 54 (3). https://doi.org/10.1145/3444689.

Anexos

Anexo 1: Tipos de ambigüedad

Tabla 22: Tipos de ambigüedad

Tipo de Ambigüedad	Subtipo	Ejemplo
Ambigüedad léxica: término aislado que admite diferentes interpretaciones	**Ambigüedad de la Polisemia:** una palabra tiene diversos significados y no queda claro a cuál de todos se hace referencia.	"Me gusta esta chaqueta" puede referirse a una chaqueta individual o a todos los tipos de chaqueta igual a esa
	Ambigüedad Homónimos: términos que comparten una misma escritura o pronunciación, pero difieren de significado.	(por ejemplo, banco puede ser el asiento, o el Establecimiento comercial que almacena dinero y concede préstamos).
Ambigüedad Sintáctica: ocurre cuando una determinada secuencia de palabras se le puede asociar más de una estructura gramatical con significados distintos.	**Ambigüedad del Apego:** ocurre cuando una oración tiene más de dos frases preposicionales.	El niño vio a la niña con el telescopio. Aquí se relaciona el sintagma preposicional "con el telescopio" a dos variantes 1-el niño vio a la niña que lleva un telescopio. 2-el niño vio a la niña atreves del telescopio.
	Ambigüedad Analítica: se produce cuando la naturaleza de la palabra o frase está en duda, se puede sacar más de un análisis de la palabra o frase.	Cuando el papel del componentes dentro de una frase o la oración es ambiguo.
	Ambigüedad de la Coordinación: es cuando un modificador se puede adjuntar a una conjunción completa o a un componente de la conjunción.	Vi a Pedro y Pablo y María me vio a mí.
Ambigüedad Semántica: una sentencia o palabra tiene más de una interpretación dentro de su contexto.	**Ambigüedad Anafórica:** ocurre cuando el texto ofrece dos o más antecedentes candidatos potenciales.	El caballo subió corriendo la colina. Fue muy empinado. Pronto se cansó. "se cansó" hace referencia a el caballo de la primera oración.
	Ambigüedad del Ámbito de Aplicación: ocurre cuando dos cuantificadores o	Todas las luces tienen un interruptor, puede significar un interruptor para muchas

	expresiones similares pueden tener alcance uno sobre el otro de diferentes maneras en el significado de la oración.	luces, o un interruptor para una luz.
Ambigüedad Pragmática: cuando una oración tiene varios significados en el contexto en el que se expresa.	**Ambigüedad Referencial:** aparece cuando una palabra localiza su referencia a partir de más de un elemento precedente.	Los profesores proporcionarán información a los alumnos antes de que se vayan de vacaciones.

Anexo 2: Criterios de calidad

Tabla 23: Resumen de criterio de calidad

Criterio de Calidad	**Criterios del lenguaje de requisito**
ISO/IEC/IEEE29148:2018	Superlativos (como "mejor", "más")
	Lenguaje subjetivo (como "fácil de usar", “rentable")
	Pronombres vagos (como "eso", "esto", "aquello")
	Términos ambiguos como adverbios y adjetivos (como "casi siempre", "significativo", "mínimo") afirmaciones lógicas ambiguas (como "o", "y/o")
	Términos abiertos y no verificables (como "proporcionar apoyo", "pero no limitado a", "como mínimo")
	Frases comparativas (como "mejor que", "mayor calidad")
	Lagunas (como "si es posible", "según proceda", "según proceda")
	Términos que implican una totalidad (como "todo", "siempre", "nunca" y "cada")
	referencias incompletas (no especificar la referencia con su fecha y número de versión; no especificar sólo las partes aplicables de la referencia para restringir el trabajo de verificación)
(Gnesi y Trentanni, 2019)	Opcionalidad significa que el requisito contiene una parte opcional (es decir, una parte que puede o no puede considerarse).
	Subjetividad significa que el requisito expresa opiniones o sentimientos personales

	Vaguedad significa que el requisito contiene palabras que no tienen un significado cuantificable de forma única por ejemplo malo, claro, cerca, fácil, lejos, rápido, bueno son algunas de las palabras que revelan vaguedad.
	Debilidad significa que la frase contiene un verbo "débil". Un verbo que hace que la frase no sea imperativa se considera débil (es decir, puede, podría, puede).
	Implícito significa que el requisito no especifica el sujeto o el objeto mediante su nombre específico, sino que utiliza un pronombre u otra referencia indirecta.
	Multiplicidad es la aparición de términos que revelan multiplicidad como son y/o, así como la presencia de listas detalladas.
	Especificación insuficiente significa que el requisito contiene una palabra que identifica una clase de objetos sin un modificador que especifique una instancia de esta clase.
(Arendse y Lucassen, 2016)	El término y/o, imperativo, lista de sustantivo, frases múltiples facilitan la descripción de múltiples características.
	Pronombre del que no se sabe con certeza a qué sustantivo o frase nominal se refiere.
	Cuantificador cuyo alcance no es seguro.
	Uso de un número sin unidad.
	Barra oblicua cuyos sustantivos o frases sustantivas a ambos lados tienen un significado diferente.
	Uso de término cuya veracidad o exactitud no puede determinarse.
	Términos que las personas pueden interpretar de manera diferente.

	Términos que expresan una relación con otros sustantivos o frases sustantivas.
	Términos que expresan una restricción.
	Términos que significan la mayor forma de un adverbio o adjetivo.
	Términos generales que pueden causar ambigüedad.
	Términos que implican un final abierto de una frase
(Ferrari, et al. 2019)	Ambigüedad léxica se produce siempre que un término puede tener diferentes significados
	Ambigüedad sintáctica se manifiesta siempre que la frase puede tener más de una estructura gramatical, cada una con un significado diferente
	Longitud excesiva indica que una frase es demasiado larga
	Jerga jurídica es el uso de términos y construcciones que pertenecen al ámbito jurídico
	Jerga difícil cuantifica la cantidad de frases que utilizan términos (de una o varias palabras) que se consideran difíciles, bien porque son poco frecuentes, bien porque son expresiones demasiado complejas que pueden sustituirse por otras más sencillas
	Actor poco claro este indicador indica que el actor de una acción no está claro.
	Acrónimo poco claro Un acrónimo es una palabra formada por las letras iniciales o

	partes de otras palabras, generalmente utilizada para identificar organizaciones. Este indicador comprueba los acrónimos que nunca se expresan en su forma extendida

Anexo 3: Relación de tareas y técnicas de PLN

Tabla 24: Relación de tareas y técnicas PLN

Tarea	Técnica con que se relaciona	Explicación
Etiquetado de parte del Discurso	• Etiquetado de parte del discurso	Asociar palabras con etiquetas de parte del discurso (POS) para distinguir entre sustantivos, verbos, adjetivos, adverbios, etc.
Etiquetado semántico	• Extracción de términos Correspondencia de términos • Fragmentación • Extracción de conceptos Reconocimiento de entidades con nombre • Etiquetado semántico de roles • Etiquetado temporal	Extraer del texto fragmentos de información útiles (palabras, términos, relaciones, etc.).
Análisis sintáctico	• Análisis sintáctico de dependencias • Análisis sintáctico de circunscripciones • Gramática de enlaces	Construir una estructura sintáctica que represente la relación entre los componentes lógicos de un flujo de texto, como el árbol de análisis sintáctico o el gráfico de análisis de dependencias.
Análisis semántico	• Análisis semántico • Análisis de sentimiento • Anotación de textos • Anotación semántica • Modelización de temas • Resumen • Asignación de Dirichlet latente (LDA) • Indexación semántica latente (LSI) • Patrones semánticos • Gramática de casos • Marcos semánticos • Grafos de conocimiento reconocimiento de detalles	Identificar y etiquetar componentes y relaciones semánticamente relevantes en el texto.

	textuales • Detección de homónimos • Detección de sinónimos • Resolución de coreferencias • Resolución de anáforas	
Análisis de la frecuencia	• Bolsa de palabras • Frecuencia de palabras • Frecuencia de términos • Frecuencia inversa de documentos Análisis de colocalización • Matriz término documento • Recuento de caracteres • Concordancia	Analizar la frecuencia de aparición de elementos léxicos (por ejemplo, palabras y caracteres) y grupos de elementos (por ejemplo, frases y multipalabras) en un texto.
Análisis de similitudes	• Similitud de coseno • Afinidad léxica • Distancia de similitud • Similitud de documentos • Similitud léxica	Calcular los valores numéricos de la similitud entre los elementos del texto, como para identificar la relación semántica.
Análisis basado en reglas	• Expresión regular • patrones léxicos • Reglas de generación	Utilizar reglas o patrones para analizar la sintaxis o la semántica de un texto o transformar el texto.
Normalización de textos	• Desglose • Lematización • Eliminación de palabras vacías Eliminación de ruido • Eliminación de puntuación • Minúsculas • Separación de mayúsculas y minúsculas	Convierte las palabras en su forma original y elimina las palabras o caracteres innecesarios del texto
Segmentación de textos	• Tokenización • Segmentación de frases	Descomponer un texto en una secuencia de tokens individuales (es decir, palabras o frases).
Representación del texto	• N-gram • Word2Vec • Context2Vec • Doc2Vec • GloVe • FastText	Representar palabras, frases o documentos mediante vectores de números reales.

Anexo 4: Comparación de lenguajes

Tabla 25: Comparación de lenguajes

Característica	**Java**	**JavaScript**
Compilador	Necesario un kit de	No es necesario que sus

	desarrollo y un compilador	programas se compilen, sino que éstos se interpretan por parte del navegador.
Orientado a objetos	orientado a objetos	No requiere programar orientado a objetos, aunque sí lo permite
Propósito	Propósito general	Programas para que se ejecuten en páginas web

Anexo 5: Historias de usuario

Tabla 26: HU Mostrar datos del documento importado

Historia de usuario	
Número:2	Nombre de la Historia de Usuario: Mostrar datos del documento importado
Modificación de la Historia de Usuario: ninguna	
Usuario: Jonathan Ramírez Reyes	
Prioridad en negocio: alta	Riesgo en desarrollo: medio
Puntos estimados: 5 días	Iteración asignada: 1
Descripción: La herramienta debe permitir visualizar el título y extensión del documento importado.	
Observaciones: se muestran el título y extensión referente al documento importado	

Tabla 27: HU Mostrar tipo de ambigüedad.

Historia de usuario	
Número:4	**Nombre de la Historia de Usuario:** Mostrar tipo de ambigüedad

Modificación de la Historia de Usuario: **ninguna**	
Usuario: Samira Enríquez Gonzales	
Prioridad en negocio: **Alta**	**Riesgo en desarrollo:** medio
Puntos estimados: **3 días**	**Iteración asignada:** 2
Descripción: **la herramienta debe permitir visualizar las ambigüedades léxicas y sintácticas detectadas.**	
Observaciones: 1. **Se muestra las ambigüedades léxicas y sintácticas detectadas si existen ambigüedades léxicas y sintácticas en los requisitos redactados o del documento.** 2. **En caso de no existir ambigüedad léxica o sintáctica el sistema debe mostrar la notificación** (“no hay ambigüedad ”).	

Tabla 28: HU. Redactar requisito

Historia de usuario	
Número:3	Nombre de la Historia de Usuario: Redactar requisito
Modificación de la Historia de Usuario: ninguna	
Usuario: Jonathan Ramírez Reyes	
Prioridad en negocio: alta	Riesgo en desarrollo: medio
Puntos estimados: 3 días	Iteración asignada: 1
Descripción: La herramienta debe permitir redactar un requisito de software	

Observaciones: **1.** Se muestran el requisito redactado **2.** En caso de no haber un requisito redactado y presionar el botón (“Procesar requisito”) el sistema debe mostrar la notificación (“debe escribir un requisito”)

Tabla 29: HU: Eliminar ambigüedades

Historia de usuario	
Número:6	Nombre de la Historia de Usuario: Eliminar ambigüedades
Modificación de la Historia de Usuario: ninguna	
Usuario: Jonathan Ramírez Reyes	
Prioridad en negocio: media	Riesgo en desarrollo: medio
Puntos estimados: 3 días	Iteración asignada: 1
Descripción: La herramienta debe permitir eliminar la lista de ambigüedades detectada en los requisitos redactados y del documento	
Observaciones:	

Tabla 30: HU: Seleccionar columna analizar

Historia de usuario	
Número:7	Nombre de la Historia de Usuario: Seleccionar columna analizar
Modificación de la Historia de Usuario: ninguna	
Usuario: Jonathan Ramírez Reyes	
Prioridad en negocio: media	Riesgo en desarrollo: medio
Puntos estimados: 3 días	Iteración asignada: 1

Descripción: La herramienta debe permitir seleccionar la columna donde se encuentran los requisitos para analizarlos.
Observaciones:

Anexos 6: Tarjetas CRC

Tabla 31: TCR. Detectar ambigüedades léxicas

Tarjeta CRC	
Clase: Detectar ambigüedades léxicas.	
Responsabilidades	**Colaboración**
Detectar ambigüedades léxicas	word_tokenize PorterStemmer TextBlob

Tabla 32: TCR.Detectar ambigüedades sintácticas

Tarjeta CRC	
Clase: Detectar ambigüedades sintácticas.	
Responsabilidades	**Colaboración**
Detectar ambigüedades sintácticas	word_tokenize PorterStemmer TextBlob

Anexos 7: Tareas de ingeniería

Tabla 33: TI. Diseñar mostrar datos del documento

Tarea	
Número de tarea: 3	**Número de HU:** 2
Nombre de la tarea: **diseñar mostrar datos del documento**	
Tipo de tarea: Diseño	**Estimación:** 5 días
Descripción: **se diseñó un apartado al lado del botón de carga del documento que muestra el nombre del archivo cargado y la extensión del mismo.**	

Tabla 34: TI. Diseñar mostrar tipo de ambigüedades

Tarea	
Número de tarea: 6	**Número de HU:** 4
Nombre de la tarea: **diseñar mostrar tipo de ambigüedades**	
Tipo de tarea: Diseño	**Estimación:** 1 días
Descripción: **Se diseñó una fila en el resultado del análisis que muestra el tipo de ambigüedad que se detectó en el requisito analizado**	

Tabla 35: TI. Mostrar tipo de ambigüedades

Tarea	
Número de tarea: 7	**Número de HU:** 4
Nombre de la tarea: **mostrar tipo ambigüedades**	
Tipo de tarea: Desarrollo	**Estimación:** 2 días
Descripción: **Se implementó el método mostrarAmbiguedades() que genera una tarjeta que contiene el tipo de ambigüedad detectadas, su descripción correspondientes y el requisito donde se detectó dicha ambigüedad.**	

Tabla 36: TI. Eliminar ambigüedad

Tarea	
Número de tarea: 8	**Número de HU:** 6
Nombre de la tarea: **eliminar ambigüedad**	
Tipo de tarea: Desarrollo	**Estimación:** 2 días
Descripción: **se implementó el método eliminarAmbiguedades() que borra la base de datos y elimina las ambigüedades detectadas anteriormente**	

Tabla 37: TI. Diseño de eliminar ambigüedad

Tarea	
Número de tarea: 9	**Número de HU:** 6

Nombre de la tarea: **diseño de eliminar ambigüedad**	
Tipo de tarea: Diseño	**Estimación:** 1 días
Descripción: **se diseñó un botón con el nombre Eliminar lista de ambigüedades que al presionarlo elimina las ambigüedades que estén en la lista.**	

Tabla 38:TI.Redactar requisito

Tarea	
Número de tarea: 10	**Número de HU:** 3
Nombre de la tarea: **redactar requisito**	
Tipo de tarea: Desarrollo	**Estimación:** 2 días
Descripción: **se implementó un input de tipo textArea que permite redactar en el requisito de software para su posterior análisis**	

Tabla 39: TI.Diseño redactar requisito

Tarea	
Número de tarea: 11	**Número de HU:** 3
Nombre de la tarea: **diseño redactar requisito**	
Tipo de tarea: Diseño	**Estimación:** 1 días
Descripción: **se diseñó un área de texto que permite redactar los requisitos de software de manera manual para su posterior análisis**	

Tabla 40: TI. Seleccionar columna

Tarea	
Número de tarea: 12	**Número de HU:** 7
Nombre de la tarea: **seleccionar columna**	
Tipo de tarea: Desarrollo	**Estimación:** 2 días
Descripción: **se implementó un método selecColumna() que recibe el archivo de tipo csv o xlsx, y devuelve las columnas que tenga el mismo para su posterior selección.**	

Tabla 41: TI. Diseño seleccionar columna

Tarea	
Número de tarea: 13	**Número de HU:** 7
Nombre de la tarea: **diseño seleccionar columna**	
Tipo de tarea: Diseño	**Estimación:** 1 días
Descripción: **se diseñó un input de tipo select que al desplegarse muestra las columnas que contiene el ultimo archivo cargado por la aplicación.**	

Anexo 8: Validación

Tabla 42: Matriz de confusión

Respuesta	Precisión		Respuesta	Precisión	
	0	**1**		**0**	**1**
0	VN	VP	**0**	40	54
1	FN	FP	**1**	5	1

0: detecto correctamente que hay ambigüedad

1: detecto correctamente que no hay ambigüedad

VN: verdaderos negativos

VP: verdaderos positivos

FN: falsos negativos

FP: falsos positivos

Fórmulas de las métricas

$$\boldsymbol{Precisión} = \frac{VP}{VP + FP} = \frac{54}{54 + 1} = 0.98 = 98\%$$

$$\boldsymbol{Exhaustividad} = \frac{VP}{VP + FN} = \frac{54}{54 + 5} = 0.91 = 91\%$$

$$F1 = 2\,\frac{Precisión * Exhaustividad}{Precisión + Exhaustividad} = 2\frac{0.98 * 0.91}{0.98 + 0.91} = 0.94 = 94\%$$

Anexo 9: Pruebas de Aceptación

Tabla 43: PA. Importar documento

Caso de Prueba de Aceptación
Código: P1HU **Número de HU: 1**
Nombre: Importar documento
Descripción: Se verifica si el usuario puede importar un documento
Condiciones de ejecución: El usuario debe acceder al documento con el formato adecuado (.xlsx,.csv)
Entrada/Pasos de ejecución: 1. Presionar el botón **examinar**. 2. Seleccionar el archivo que se desea importar a la aplicación para ser analizado.
Resultado esperado: Se carga el documento en la aplicación.
Evaluación de la prueba: Satisfactorio.

Tabla 44: PA. Redactar requisito

Caso de Prueba de Aceptación
Código: P3HU **Número de HU: 3**
Nombre: redactar requisito
Descripción: Se verifica si el usuario puede redactar un requisito de software
Condiciones de ejecución: El usuario debe redactar un requisito en el cuadro de texto
Entrada/Pasos de ejecución: **1.** Escribir el requisito de software de forma manual en el campo correspondiente.
Resultado esperado: Se puede redactar un requisito
Evaluación de la prueba: Satisfactorio.

Tabla 45: PA: Eliminar lista de ambigüedades

Caso de Prueba de Aceptación
Código: P6HU **Número de HU: 6**
Nombre: Eliminar lista de ambigüedades
Descripción: Se verifica si el usuario puede eliminar la lista de ambigüedades detectadas en los requisitos
Condiciones de ejecución: debe existir una lista de ambigüedades detectada
Entrada/Pasos de ejecución: 1. El usuario presiona el botón **Eliminar lista de ambigüedades.**
Resultado esperado: Se elimina la lista de requisitos ambiguos
Evaluación de la prueba: Satisfactorio.

Tabla 46: PA. Mostrar datos del documento importado

Caso de Prueba de Aceptación
Código: P2HU **Número de HU: 2**
Nombre: Mostrar datos del documento importado
Descripción: Se verifica si la herramienta muestra el título y extensión del documento importado previamente.
Condiciones de ejecución: El usuario debe haber cargado el documento con el formato adecuado (.xlsx,.csv)
Entrada/Pasos de ejecución:
Resultado esperado: Se muestran los datos del documento.
Evaluación de la prueba: Satisfactorio

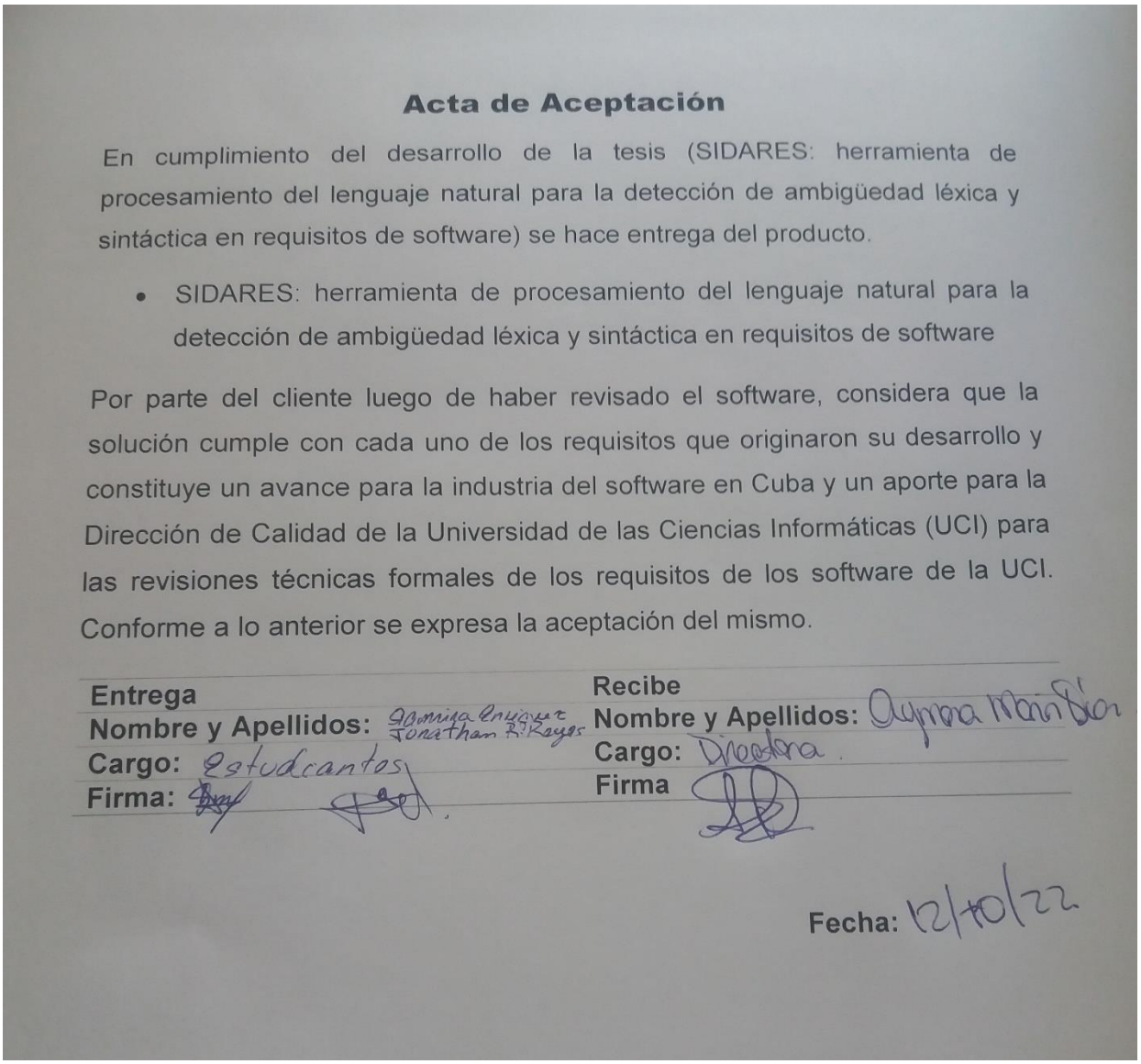
Tabla 47: PA. Mostrar ambigüedades

Caso de Prueba de Aceptación
Código: P4HU **Número de HU: 4**
Nombre: Mostrar ambigüedad léxica y sintáctica
Descripción: Se verifica si la herramienta muestra correctamente las ambigüedades léxicas y sintácticas
Condiciones de ejecución: Debe estar cargado un documento y haber procesado la columna correspondiente a la descripción de los requisitos o los requisitos redactados de forma manual. Se debe haber presionado anteriormente el botón PROCESAR COLUMNA o PROCESAR REQUISITO. El requisito a analizar debe representar al menos una ambigüedad léxica o sintáctica
Entrada/Pasos de ejecución:
Resultado esperado: Muestra las ambigüedades léxicas y sintácticas detectadas.
Evaluación de la prueba: Satisfactorio

Tabla 48: PA. Seleccionar columna

Caso de Prueba de Aceptación
Código: P7HU **Número de HU: 7**
Nombre: seleccionar columna analizar
Descripción: Se verifica si el usuario puede seleccionar correctamente la columna donde se encuentra los requisitos analizar
Condiciones de ejecución: Debe estar cargado el documento. El documento debe ser del formato correcto (csv o xlsx)
Entrada/Pasos de ejecución: 1. Expandir el input de tipo select que contiene las columnas del documento cargado. 2. Seleccionar la columna deseada.
Resultado esperado: selecciona la columna de requisitos
Evaluación de la prueba: Satisfactorio

Anexo 10: Acta de aceptación



Acta de Aceptación

En cumplimiento del desarrollo de la tesis (SIDARES: herramienta de procesamiento del lenguaje natural para la detección de ambigüedad léxica y sintáctica en requisitos de software) se hace entrega del producto.

- SIDARES: herramienta de procesamiento del lenguaje natural para la detección de ambigüedad léxica y sintáctica en requisitos de software

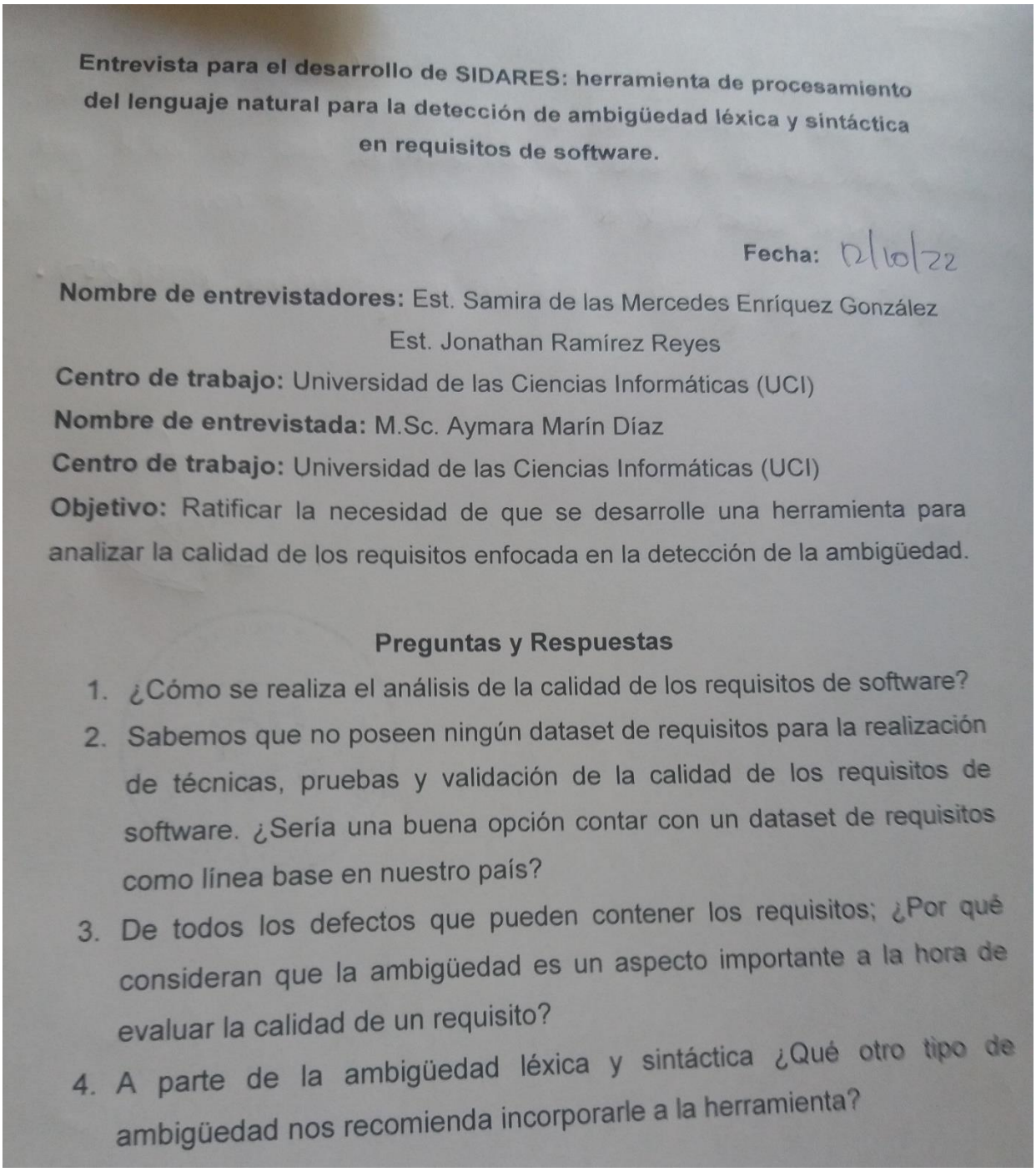
Por parte del cliente luego de haber revisado el software, considera que la solución cumple con cada uno de los requisitos que originaron su desarrollo y constituye un avance para la industria del software en Cuba y un aporte para la Dirección de Calidad de la Universidad de las Ciencias Informáticas (UCI) para las revisiones técnicas formales de los requisitos de los software de la UCI. Conforme a lo anterior se expresa la aceptación del mismo.

Entrega		Recibe	
Nombre y Apellidos:	Jonathan B. Reyes	Nombre y Apellidos:	Ayrna Martínez
Cargo:	Estudiantes	Cargo:	Directora
Firma:		Firma	

Fecha: 12/10/22

Figura 13: Acta de aceptación

Anexo 11: Entrevista



Entrevista para el desarrollo de SIDARES: herramienta de procesamiento del lenguaje natural para la detección de ambigüedad léxica y sintáctica en requisitos de software.

Fecha: 12/10/22

Nombre de entrevistadores: Est. Samira de las Mercedes Enríquez González
Est. Jonathan Ramírez Reyes

Centro de trabajo: Universidad de las Ciencias Informáticas (UCI)

Nombre de entrevistada: M.Sc. Aymara Marín Díaz

Centro de trabajo: Universidad de las Ciencias Informáticas (UCI)

Objetivo: Ratificar la necesidad de que se desarrolle una herramienta para analizar la calidad de los requisitos enfocada en la detección de la ambigüedad.

Preguntas y Respuestas

1. ¿Cómo se realiza el análisis de la calidad de los requisitos de software?
2. Sabemos que no poseen ningún dataset de requisitos para la realización de técnicas, pruebas y validación de la calidad de los requisitos de software. ¿Sería una buena opción contar con un dataset de requisitos como línea base en nuestro país?
3. De todos los defectos que pueden contener los requisitos; ¿Por qué consideran que la ambigüedad es un aspecto importante a la hora de evaluar la calidad de un requisito?
4. A parte de la ambigüedad léxica y sintáctica ¿Qué otro tipo de ambigüedad nos recomienda incorporarle a la herramienta?

Figura 14: Entrevista

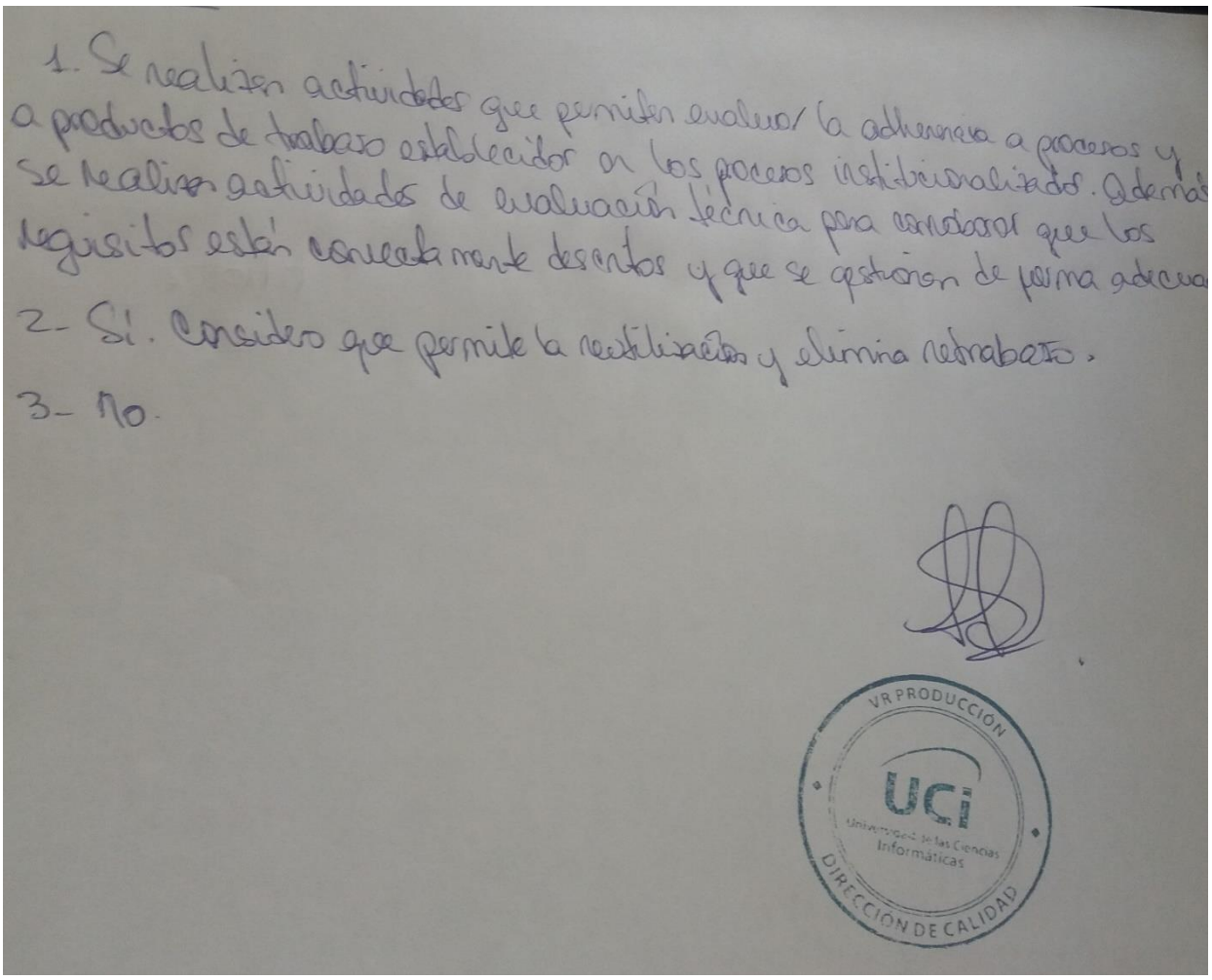


Figura 15: Respuesta de la entrevista

Anexo 12: No conformidades

Tabla 49: No conformidades

No	No Conformidades	Tipo	Estado
1	Redactar requisito: al analizar el requisito no mostraba la lista de ambigüedades actualizada	Interfaz	Resuelta
2	Cargar documento: si no hay un documento cargado y se presiona el botón Procesar columna no muestra una alerta de error	Funcional	Resuelta
3	Procesar requisito	Funcional	Resuelta
4	Procesar columna	Funcional	Resuelta
5	Eliminar lista de ambigüedades	Funcional	Resuelta
6	Ambigüedades sintácticas detectadas erróneamente	Funcional	Resuelta
7	No se muestra el requisito con la ambigüedad detectada	Interfaz	Resuelta