Universidad de las Ciencias Informáticas
Facultad 3

Herramienta informática para la identificación de la ambigüedad en textos de la legislación cubana

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

Autor:

Yamila Ortuño Sánchez

Tutores:

Ing. Yordanis García Leiva

MsC. Yarina Amoroso Fernández

Cotutor:

Ing. Reinier Silverio Figueroa

Junio, 2016

"Año 58 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaro que soy la única autora de este trabajo y autorizo a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los _____ días del mes de _____ del año _____.

_____

Yamila Ortuño Sánchez

_____ _____

MsC. Yarina Amoroso Fernández Ing. Yordanis García Leiva

_____

Ing. Reinier Silverio Figueroa

AGRADECIMIENTOS

A mi mamá por haberme dado siempre todo su amor y comprensión.

A mi papá por siempre haber confiado en mí, apoyarme en mis decisiones y quererme tanto.

A mis tutores por el apoyo brindado, especialmente a Yordanis por darme el honor de ser su tesista.

A mis otros tutores Mimi y Erlis por estar siempre pendientes de mí.

A mis hermanos Anier , Niuska y Reinaldo por siempre darme su amor.

A mis primas Maité y Yaisel y mis tías Magalis y Niurvis por preocuparse siempre por mí.

A Noelia, Daniel y Niurka por quererme como una más de la familia.

A mi familia de Artemisa por siempre preocuparse por mí y brindarme su apoyo.

A Mairelys por sus consejos que me ayudaron a llegar a este punto.

A las chicas veneno por todas esas peleas que hacen que cada día sea especial.

A todos los amigos que me han apoyado durante el transcurso de la universidad.

A todos los profesores que de una forma u otra han influido en mi formación

DEDICATORIA

Dedico esta tesis a mi mamá y mi hermano, espero que desde donde estén puedan verme y estén orgullosos de mí.

A Niuska, Anier, Nayelis y Maité que son la razón de mi existir.

A mi papá por siempre haber creído en mí.

A Erlis por siempre brindarme su apoyo en los momentos más difíciles y creer en mi cuando yo no lo hacía.

RESUMEN

La ambigüedad se presenta en un texto cuando en este existen términos que pueden admitir diferentes interpretaciones; causando dudas, incertidumbre y confusión en su comprensión. Los textos jurídicos no están exentos de estos inconvenientes del lenguaje. En el campo de la informática se han realizado algunas soluciones que permiten enfrentar estos problemas lingüísticos, sin embargo, la cifra de ambigüedades que detectan es baja. El presente trabajo se ha realizado con el propósito de desarrollar una herramienta informática que permita elevar la identificación de ambigüedades presente en textos de la legislación cubana, contribuyendo a la desambiguación de los mismos. La solución está basada en el uso de técnicas de la minería de textos y el procesamiento del lenguaje natural, utilizando un conjunto de reglas heurísticas definidas a partir de las características del idioma español; las cuales fueron implementadas a través del uso de tecnologías de código abierto, cumpliendo con las políticas de soberanía tecnológica establecidas en el país. La herramienta obtenida constituye un avance en el desarrollo de la Informática Jurídica en Cuba, al identificar un mayor número de ambigüedades en los textos de la legislación cubana en relación a otras soluciones similares ya existentes.

PALABRAS CLAVES: ambigüedad, herramienta informática, Legislación Cubana, reglas heurísticas, textos.

ABSTRACT

The ambiguity is presented in a text when it can have different interpretations; causing doubt, uncertainty and confusion in understanding. The legal texts are not exempt from these drawbacks of language. In the computer field there have been some solutions to these linguistic problems, however, the number of ambiguities that detect is low. This work was carried out with the purpose of develop a software tool that will raise the identification of ambiguities present in the texts of Cuban legislation, contributing to the disambiguation of them. The solution is based on the use of techniques of text mining and natural language processing, using a set of heuristic rules defined from the characteristics of the Spanish language; which they were implemented through the use of open source technologies, meeting the technological sovereignty policies established in the country. The tool obtained constitutes a great advance in the development of Legal Informatics in Cuba, by identifying a greater number of ambiguities in the texts of Cuban legislation in relation to other similar solutions already exist.

KEYWORDS: *ambiguity, computer tool, Cuban legislation, heuristic rules, texts.*

.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

En la vida cotidiana para cada una de las actividades que realiza el hombre, existen límites y obligaciones. Esto trae consigo no solo las responsabilidades familiares y laborales, sino también cumplir y aceptar como ciudadanos las leyes que se instauran en una nación. Cada país cuenta con regulaciones, recogidas en textos jurídicos, que establecen la organización, funcionamiento y estructura política de la nación, así como los deberes y derechos de los ciudadanos.

Los textos jurídicos tienen una naturaleza ambigua. Esta se identifica cuando a través del análisis de un contexto no es posible determinar el significado de una palabra o una oración en general. Esto puede traer como resultado la ocurrencia de incertidumbre, duda e interferencia en la comprensión. Por ello, este asunto es un tema de investigación que ha acompañado a las ciencias jurídicas en conjunto con las ciencias filológicas y la lingüística computacional, para llegar a contar con herramientas que, basadas en las reglas y modelos de redacción de documentos legales, ayuden a los operadores jurídicos a redactar y revisar los documentos que emiten.

En Cuba existen pocos avances en la obtención de soluciones informáticas que posibiliten identificar la ambigüedad presente en textos de la legislación cubana, los cuales se caracterizan por la existencia de fenómenos lingüísticos que dan lugar a incertidumbre en la compresión del contenido; es decir, no se tiene una visión clara del significado que puede tener el contexto analizado. Además, provoca inconsistencia, pues no existe una compresión exacta de lo que realmente se está leyendo, a menos que sea un especialista en el tema, por tanto, el resultado del análisis de los textos es bajo, provocando confusiones en su interpretación.

En el Centro de Gobierno Electrónico (CEGEL), adjunto a la Facultad 3 de la Universidad de las Ciencias Informáticas (UCI), se desarrolló una herramienta informática que permite identificar la ambigüedad presente en textos de la legislación cubana. Esta solución constituye un aporte al desarrollo de la informática jurídica en Cuba, sin embargo, las características que se listan a continuación pasaron a ser limitantes de la herramienta:

- Las reglas que nutren el funcionamiento de la misma se encuentran implícitas en el código, imposibilitando la modificación o adición de otras, afectando así la escalabilidad del software.

- No se contempla el análisis de ambigüedades provocadas por conjunciones subordinantes causales.
- Importa un reducido número de formatos de textos.
- Presenta una baja cantidad de términos jurídicos en el diccionario de palabras, diseñado para la identificación de ambigüedades léxicas y semánticas.

Las limitaciones antes señaladas afectan la necesidad del cliente de aumentar la capacidad de identificación de ambigüedades en textos de la legislación cubana.

La situación problemática antes descrita ha generado el siguiente **problema de investigación:**

¿Cómo elevar la capacidad de identificación de ambigüedades presente en textos de la legislación cubana, contribuyendo a la desambiguación de los mismos?

Tomando en cuenta el problema antes propuesto se define como **objeto de estudio**: el proceso de desambiguación de textos.

Para ello se identifica como **campo de acción**: el proceso de desambiguación de los textos de la legislación cubana.

Determinándose como **objetivo general:** desarrollar una herramienta informática que permita elevar la capacidad de identificación de ambigüedades presentes en textos de la legislación cubana, contribuyendo a la desambiguación de los mismos.

Se desglosan del objetivo general los siguientes **objetivos específicos:**

- Elaborar el marco teórico referencial de la investigación relacionado con la ambigüedad presentes en los textos, a partir del análisis de las principales investigaciones desarrolladas sobre el tema.
- Implementar una herramienta informática que permita elevar la capacidad de identificación de ambigüedades en textos de la legislación cubana utilizando tecnologías de código abierto.
- Validar la herramienta informática propuesta aplicando diferentes métodos y métricas de validación de software.

Para dar cumplimiento a los objetivos propuestos se definen las siguientes **tareas de la investigación:**

- Desarrollo del marco teórico referencial de la investigación en función de los términos relacionados con la ambigüedad.
- Caracterización de cada una de las herramientas a utilizar en la investigación.
- Identificación y validación de los requisitos funcionales y no funcionales a tener en cuenta en el desarrollo de la solución.
- Definición de la arquitectura a utilizar en el desarrollo del trabajo.
- Definición de las reglas para la identificación de ambigüedad en textos legales.
- Implementación de cada uno de los componentes a partir de las reglas definidas para la identificación de la ambigüedad en textos legales.
- Realización de pruebas de validación a la herramienta informática obtenida, aplicando métodos y métricas.
- Realización del informe final de la investigación.

Determinándose como **idea a defender:**

El desarrollo de una herramienta informática permitirá elevar la capacidad de identificación de ambigüedades presentes en los textos de la legislación cubana, contribuyendo a mejorar la comprensión de los mismos.

Para lograr la realización de dichas tareas se emplearon los siguientes métodos de investigación:

Métodos teóricos:

Histórico-lógico se empleó para profundizar en las tendencias y tratamientos históricos de las ambigüedades en los textos legales y determinar su influencia en el problema de investigación actual.

Analítico – sintético se empleó para el procesamiento de la información y arribar a conclusiones prácticas y teóricas de la investigación. A partir del análisis de los referentes

teóricos y la bibliografía en cuestión, se realizó una síntesis de los elementos significativos de la investigación.

Inductivo-deductivo se empleó en el desarrollo de la investigación, mediante la inducción se realizó un razonamiento que partió de lo particular a lo general, obteniendo como resultado final las semejanzas existentes en cada una de las bibliografías estudiadas. Mediante la deducción con los conocimientos generales adquiridos durante la investigación, se infirieron otros conocimientos como las conclusiones parciales y generales de la investigación.

Modelación permitió representar los modelos y diagramas de cada una de las fases de desarrollo.

Métodos empíricos:

Observación se utilizó para realizar una evaluación de la situación problemática en cuestión.

Estructura de la investigación

El contenido de este trabajo consta de tres capítulos, definidos de la siguiente forma:

Capítulo 1: Fundamentación Teórica.

Se presentan un conjunto de conceptos relacionados con la problemática antes descrita. Además, se realiza un estudio del estado del arte de las herramientas informáticas para la identificación de la ambigüedad en textos legales, así como las técnicas existentes en la actualidad. Se describen la metodología, las herramientas y el lenguaje de programación utilizado en el desarrollo de la investigación.

Capítulo 2: Análisis, diseño e implementación de la herramienta informática.

Se describen las principales características de la herramienta propuesta, así como el diseño de la misma, guiando todo el proceso por las fases definidas por la metodología XP (Extreme Programming o Programación Extrema). Se definen los componentes que permiten detectar las ambigüedades y el diseño de la base de datos empleada. Se analizan los elementos necesarios para garantizar el éxito en el proceso de desarrollo, tales como:

las historias de usuario, arquitectura, patrones de diseño y tarjetas CRC (Clase, Responsabilidad y Colaborador).

Capítulo 3: Pruebas de la herramienta informática.

Se da continuidad a las fases de la metodología XP, con la realización de pruebas unitarias y pruebas de aceptación con el cliente a la herramienta propuesta, describiéndose los principales artefactos generados, tales como las actas de liberación interna de productos de software y de aceptación del cliente. Además, se describen los resultados de la aplicación de las métricas para validar el diseño y las técnicas para la validación de los requisitos. 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 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En este capítulo se realiza la fundamentación teórica del trabajo donde se describen conceptos asociados al dominio del problema. Se analizan las metodologías del proceso de desarrollo de software, las herramientas de ingeniería del software asistida por computadora (CASE), los lenguajes de programación, los entornos de desarrollo integrado, los sistemas gestores de bases de datos, con el propósito de definir las tecnologías que se emplean en el desarrollo de la herramienta. Además, se presenta un estudio de las tendencias actuales y las técnicas existentes para identificar la ambigüedad en textos.

1.2 Marco teórico de la investigación

Para comenzar el desarrollo de la investigación se considera necesario conocer el significado del término ambigüedad. A continuación, se presentan algunas definiciones dadas por autores que sirvieron de referencia para el análisis de este tema.

Ambigüedad: puede presentarse cuando es posible admitir diferentes interpretaciones a partir de la representación de una oración; también, se presenta cuando existe confusión al tener diversas estructuras asociadas a la misma oración *(Zapata, Palomino et al. 2007)*.

Ambigüedad: en el proceso lingüístico se presenta cuando pueden admitirse distintas interpretaciones a partir de la representación o cuando existe confusión al tener diversas estructuras y no tener los elementos necesarios para eliminar las incorrectas *(Haro and Gelbukh 2007).*

Ambigüedad: 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 *(Ramos 2009).*

Son variadas las definiciones del término ambigüedad presentes en la literatura, todas varían en dependencia del contexto en el cual se manifieste este problema lingüístico. En relación con la problemática que originó el problema de investigación de este trabajo, se decidió tener en cuenta a lo largo del mismo la definición dada por la autora Sulema Torres Ramos en su tesis de doctorado en Ciencias de la Computación.

Existen diferentes tipos de ambigüedades, centrándose esta investigación en el estudio de las ambigüedades léxicas, semánticas y sintácticas, las cuales se describen a continuación:

Ambigüedad léxica: se presenta cuando una palabra además de tener diferentes significados, éstos pueden desempeñar diferentes categorías sintácticas en la oración *(Suárez Cueto 2004).* Ejemplo: La palabra "corte", se puede referir a la forma del verbo cortar o al sustantivo corte de justicia.

Ambigüedad sintáctica o ambigüedad estructural: aparece cuando debido a la forma en que se asocian los distintos constituyentes de una oración puede ser interpretada de formas distintas. Siendo en ocasiones casi imposible de solucionar (*Ramos 2009).*

Las ambigüedades sintácticas son originadas en las conjunciones o preposiciones, y pueden ser de tipo coordinativa, preposicional o subordinante. La ambigüedad sintáctica coordinativa se puede presentar cuando una oración contiene más de una palabra de tipo conjunción; esta puede ser de tipo copulativa, disyuntiva o mixta. La ambigüedad sintáctica preposicional se puede presentar cuando una oración contiene una palabra de tipo preposición. La ambigüedad sintáctica subordinante se puede presentar cuando una oración contiene una conjunción subordinante causal (porque, ya que, puesto que, como) *(Zapata, Palomino et al. 2007).*

El siguiente ejemplo muestra como la conjunción ***porque***, genera más de un significado en una oración: "El tribunal de justicia no aprobó la ley porque carecía de elementos". La oración puede ser interpretada de dos formas distintas, resumidas en la siguiente interrogante: ¿Quién carecía de elementos, el tribunal o la ley?

Ambigüedad semántica: ambigüedad debido a las palabras polisémicas. En este caso, una misma palabra puede tener distintos significados dependiendo del uso que se le esté dando en cada momento (*Ramos 2009).* Ejemplo, en la oración: "Martha fue multada cerca del banco", se puede entender dos cosas diferentes:

- fue multada cerca del banco financiero
- fue multada próxima a un banco donde sentarse.

A continuación, se describen otras de las definiciones básicas que se consideran de necesario conocimiento para la comprensión del dominio de la presente investigación.

Lingüística: es una disciplina que se encarga del estudio científico y profundo de la lenguas naturales y todo lo relacionado con ellas, entiéndase por ello: idioma, léxico, forma de hablar, pronunciación, ubicación de las lenguas en un mapa étnico – cultural y la determinación y búsqueda de lenguas perdidas, entre otros aspectos que se enfocan en el habla del ser humano. La lingüística propone y recrea leyes y normas para el habla a fin de concentrar el uso de la lengua en algo correcto, estudia su funcionamiento general y cómo se comporta en el medio ambiente (*Orozco 2014)*.

Lingüística computacional o Procesamiento del Lenguaje Natural (PLN): disciplina que combina la lingüística y la informática con el fin de modelar el lenguaje humano desde un punto de vista computacional (*Ramos 2009).*

Texto jurídico: traslación de una lengua a otra de los textos que se utilizan en las relaciones entre el poder público y el ciudadano (por ejemplo: denuncias, querellas, exhortos, citaciones, leyes) y también, naturalmente, de los textos empleados para regular las relaciones entre particulares con transcendencia jurídica (que dan lugar a contratos, testamentos o poderes) (*Gutiérrez Arcones 2015)*.

Polisemia: capacidad que tiene una palabra para expresar más de un significado *(Ramos 2009*).

Desambiguación de sentido de palabras (WSD): es el problema de seleccionar un sentido de un conjunto de posibilidades predefinidas para una palabra dada en un texto o discurso *(Ramos 2012)*.

1.3 Estado del arte de la investigación

En la actualidad existen herramientas que permiten detectar ambigüedades, las cuales se relacionan a continuación:

3LB-SAT (3LB-Herramienta de Anotación Semántica): es una herramienta para el etiquetado semántico de corpus[1] multilingüe. Sus principales características son que está orientado a la palabra (o token), que permite introducir el corpus en diferentes formatos y

[1] Corpus: conjunto de datos, textos u otros materiales sobre determinada materia que pueden servir de base para una investigación o trabajo.

que usa EuroWordnet[2] para consultar el sentido de las palabras en cuatro lenguas (español, catalán, euskara e inglés). Las palabras monosémicas[3] se anotan automáticamente y se muestran todas las apariciones de un lema en el texto, siendo posible asociar más de un synsets[4] con una aparición de un lema (*Bisbal, Molina et al. 2003).*

Natural Language Toolkit (NLTK): es un paquete de herramientas y recursos libres para un espectro amplio de tareas dentro de PLN. NLTK está destinado a apoyar la investigación y la enseñanza en PLN o áreas relacionadas, que incluyen la lingüística empírica, las ciencias cognitivas, la inteligencia artificial, la recuperación de información y el aprendizaje de la máquina. En la actualidad NLTK incluye utilidades para más de 10 idiomas entre la que se encuentran el español, catalán, portugués, y euskera. El paquete integra diferentes tipos de corpus: texto simple, texto etiquetado con su categoría, etiquetado con sintaxis superficial, etiquetado con sintaxis profunda e incluso simples listas de palabras o léxicos. Cada corpus ofrece una serie de métodos para leer sus datos palabra por palabra, oración por oración, párrafo por párrafo o por unidades de etiquetado *(Manterola, Díaz de Ilarraza et al. 2010*).

MPRO: la tarea fundamental del programa MPRO consiste en realizar un análisis morfológico y sintáctico automático de textos españoles técnicos y generales. Consiste en una serie de subprogramas, diccionarios, léxicos y gramáticas que interactúan entre sí. Estos componentes pueden ser adaptados a las necesidades del usuario y al tipo de textos con los que se trabaja, sean estos textos especializados o generales. *(Haller, Donoso et al. 2002*).

Herramienta informática para la evaluación de la ambigüedad en textos legales: basada en la definición de un conjunto de reglas gramaticales, implementadas a partir del uso de técnicas del procesamiento del lenguaje natural y la minería de textos. Esta permite identificar las ambigüedades presentes en documentos de la legislación cubana, brindar una clasificación de las mismas (léxica, sintáctica, semántica) y evaluar cuantitativa y cualitativamente el grado de ambigüedad presente en los textos analizados, teniendo en

[2] EuroWordnet: diccionario electrónico semántico que tiene como fin la construcción de una base de datos léxico-semántica para las lenguas castellano, holandés, italiano e inglés.

[3] Monosémicas: son las palabras que cuentan con un significado único.

[4] Synsets: conjunto de sinónimos que se consideran representativos de un concepto lexicalizado.

cuenta la cantidad de oraciones que contienen los mismos *(Carrazana Galán and Marimón Baullosa 2015).*

Las herramientas 3LB-SAT, NLTK y MPRO poseen funcionalidades para detectar los sentidos de las palabras, hacer análisis sintáctico y semántico, pero no son capaces de representar las ambigüedades en todos los sentidos. 3LB-SAT solamente registra las apariciones de un lema, mostrando los conjuntos de sinónimos que aparecen; la herramienta NLTK realiza el análisis sintáctico, mostrando una representación de la construcción de árboles sintácticos y MPRO utiliza subgramáticas en secuencia para analizar una frase u oración.

Teniendo en cuenta las características fundamentales de las herramientas 3LB-SAT y NLTK, en CEGEL se desarrolló una herramienta informática para la evaluación de la ambigüedad en textos legales, la misma detecta y clasifica las ambigüedades, pero actualmente no satisface las necesidades del cliente. Por ello se decidió realizar una herramienta que satisfaga las nuevas necesidades del cliente, teniendo en cuenta las principales características de la solución ya existente.

1.4 Técnicas para la identificación de la ambigüedad.

Existen técnicas que permiten la evaluación de la ambigüedad tanto desde el PLN como de la minería de textos. Desde el PLN existen varios métodos de WSD, los que actualmente se dividen en dos categorías principales: métodos basados en conocimiento y métodos basados en corpus *(Tello Leal 2009).*

Métodos basados en conocimiento: estos métodos utilizan un conocimiento lingüístico previamente adquirido. La idea básica consiste en utilizar recursos externos para desambiguar las palabras, tales como diccionarios, tesauros (vocabularios controlados que representan las relaciones semánticas con otras palabras y sus significados), textos sin ningún tipo de etiquetado e incluso recursos de la Web (*Aguirre et al., 2000*).

Métodos basados en corpus: estos métodos se basan en el uso de técnicas estadísticas y de aprendizaje automático para inducir modelos del lenguaje a partir de grandes conjuntos de ejemplos textuales *(Pedersen, 2001).* Los métodos basados en corpus pueden subdividirse en: métodos supervisados (corpus etiquetado) y métodos no supervisados (corpus no etiquetado). Los métodos supervisados reducen la desambiguación de sentidos

de palabras a un problema de clasificación, donde a una palabra dada se le asigna el sentido más apropiado de acuerdo a un conjunto de posibilidades, basadas en el contexto en el que ocurre. Por otra parte, los métodos no supervisados utilizan recursos como EuroWordnet para poder asignar un sentido a cada palabra que aparece en los textos no marcados, consisten básicamente en elegir de un diccionario las palabras relacionadas con la palabra a desambiguar *(Ramos 2012).*

En algunas bibliografías define como tercera categoría a los métodos híbridos, los cuales se basan en la combinación de las dos antes analizadas.

La minería de textos adopta una serie de técnicas procedentes de la recuperación de información y de la lingüística computacional. Estas técnicas incluyen *(Brun and Senso 2004)*:

- **Pre-procesamiento de los documentos:** consiste en extraer las palabras utilizadas en un documento, o segmentar el texto en distintas formas gráficas. Incluye la eliminación de los signos de puntuación y palabras vacías[5], así como la extracción de las palabras.

- **Categorización automática:** se utiliza para clasificar los documentos en categorías preestablecidas. Existen dos tipos de categorización: etiqueta simple y etiqueta múltiple. En el primero se asigna a cada documento una única categoría. En el segundo, un mismo documento puede asignarse a más de una categoría.

- **Identificación de nombres propios:** la extracción de nombres propios relativos a personas, organizaciones, eventos, funciones, así como cantidades monetarias y fechas, es una de las principales funciones que debe satisfacer la minería textual. También debe permitir identificar las relaciones que existen entre estos nombres propios y constatar así hechos descritos en los documentos.

1.5 Metodologías de desarrollo de software

Desarrollar un buen software depende de un sinnúmero de actividades y etapas, donde el impacto de elegir la mejor metodología para un equipo, en un determinado proyecto es

[5] Palabras vacías: nombre que reciben las palabras sin significado, como los artículos, las preposiciones, pronombres, conjunciones entre otras.

trascendental para el éxito del producto. El papel preponderante de las metodologías es sin duda esencial en un proyecto, esta debe adecuarse a las características del equipo, guiar y organizar las actividades que conlleven al cumplimiento de las metas trazadas en el grupo (*Navarro Cadavid, Fernández Martínez et al. 2013)*. Una metodología es una colección de métodos de solución de problemas organizados bajo una filosofía común y gobernados por un conjunto de principios *(Santibáñez 2015).*

Las metodologías se clasifican en dos grupos, las metodologías tradicionales o pesadas y las metodologías ágiles o ligeras. En el proceso de desarrollo de proyectos grandes y que generan mucha documentación se recomienda emplear las metodologías tradicionales, mientras que para proyectos pequeños se recomiendan las metodologías ligeras.

Las metodologías tradicionales centran su atención en llevar una documentación exhaustiva de todo el proceso de desarrollo del proyecto y en cumplir con la planificación definida en la fase inicial. Otra de las características importantes dentro de este enfoque son los altos costos al implementar un cambio y al no ofrecer una buena solución para proyectos donde el entorno es volátil (*Navarro Cadavid, Fernández Martínez et al. 2013).*

Las metodologías ágiles son usadas en proyectos cuyo objetivo fundamental es tener el software funcionando lo antes posible, ya que así el cliente tendrá las primeras versiones, donde podrá comprobar y aportar su idea de negocio. Con este objetivo, se trabaja sobre las versiones previas, siendo el desarrollo de la siguiente interación una mejora de la anterior. Las metodologías ágiles proponen una forma de trabajo flexible cuya planificación se actualiza continuamente. Esto contrasta con las metodologías tradicionales, las cuales siguen una planificación precisa desde el principio (*Morales García 2015).*

Debido a que se requiere reducir drásticamente el tiempo de desarrollo de la herramienta propuesta, que el equipo de trabajo es pequeño y existe un constante intercambio con el cliente, siendo este parte del equipo, se decide utilizar una metodología ágil. Dentro de las metodologías ágiles se destacan: AUP (Proceso Unificado Ágil), Scrum, XP (Programación Extrema) y SXP (SCRUM-XP).

AUP: se enfoca especialmente en la gestión de riesgos. Propone que aquellos elementos con alto riesgo obtengan prioridad en el proceso de desarrollo y sean abordados en etapas tempranas del mismo. Para ello, se crean y mantienen listas, identificando los riesgos desde

etapas iniciales del proyecto. Especialmente relevante en este sentido es el desarrollo de prototipos ejecutables durante la base de elaboración del producto, donde se demuestre la validez de la arquitectura para los requisitos clave del producto y que determinan los riesgos técnicos (*Valderrama Guayan and Benites Barrientos 2014)*.

SCRUM: se centra en la gestión de proyectos y es óptima en aquellas situaciones en las que resulta complicado planificar el futuro con mecanismos de control de procesos empíricos. La retroalimentación entre iteraciones constituye el elemento más potente de la metodología. El software es desarrollado por equipos que se auto organizan en iterativos e incrementales ciclos de corta duración (no más de 30 días) denominados *Sprints*, comenzando cada uno de ellos con una planificación y finalizando con una revisión retrospectiva. Como puede observarse, Scrum resulta válido en los entornos que trabajan con requisitos cambiantes, y necesitan rapidez de respuesta y flexibilidad *(Gallo, Mansilla et al. 2010*).

XP: según Pressman, XP es el enfoque más utilizado del desarrollo de software ágil. XP pone el énfasis en la colaboración estrecha pero informal (verbal) entre los clientes y los desarrolladores, en el establecimiento de metáforas[6] para comunicar conceptos importantes, en la retroalimentación continua y en evitar la documentación voluminosa como medio de comunicación. Para alcanzar la simplicidad, XP restringe a los desarrolladores para que diseñen sólo para las necesidades inmediatas, en lugar de considerar las del futuro. El objetivo es crear un diseño sencillo que se implemente con facilidad en forma de código. La retroalimentación se obtiene de tres fuentes: el software implementado, el cliente y otros miembros del equipo de software *(Pressman 2010*).

SXP: es un híbrido cubano de las metodologías ágiles XP y Scrum, que ofrece una estrategia tecnológica, a partir de la introducción de procedimientos ágiles que permitan actualizar los procesos de software para el mejoramiento de la actividad productiva fomentando el desarrollo de la creatividad, aumentando el nivel de preocupación y responsabilidad de los miembros del equipo, ayudando al líder del proyecto a tener un mejor control del mismo. Consiste en una programación rápida o extrema, cuya particularidad es

[6]metáfora: En el contexto de XP, es "una historia que cada quien —clientes, programadores y gerentes— narra, acerca de cómo funciona el sistema"

tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar el éxito del proyecto *(Peñalver, Meneses et al. 2010).*

Debido a que se está en presencia de un proyecto pequeño, de corta duración, con un reducido equipo de desarrollo, donde el cliente forma parte del mismo, permitiendo la comunicación y retroalimentación frecuente entre ambas partes; se define la metodología XP para guiar el proceso de desarrollo de la presente investigación.

1.5.1 Descripción de la metodología XP

El ciclo de vida de XP consiste básicamente de seis fases: Exploración, Planificación, Iteraciones por entregas, Producción, Mantenimiento y Muerte. A continuación se describe en que consiste cada fase *(Calabria and Píriz 2003).*

Exploración: en esta fase los clientes realizan las historias de usuario (HU) que desean que estén para la primera entrega. Cada HU describe una de las funcionalidades que el programa tendrá. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, la tecnología y las prácticas a ser utilizadas durante el proyecto. La duración de esta fase puede extenderse desde unas pocas semanas a varios meses dependiendo de la adaptación del equipo de desarrollo. En esta fase también se describen los requisitos funcionales y no funcionales de la aplicación.

Planificación: el objetivo de esta fase es fijar la prioridad de cada una de las HU y se establece cual va a ser el contenido de la primera entrega. Los programadores estiman cuanto esfuerzo requiere cada HU y se establece el cronograma. El artefacto generado durante esta fase es la Estimación de esfuerzo por HU.

Iteraciones por entrega: esta fase incluye varias iteraciones del sistema antes de ser entregado. El calendario es dividido en un número de iteraciones de tal manera que cada iteración tome de una a cuatro semanas de implementación. En la primera iteración se crea un sistema que abarca los aspectos más importantes de la arquitectura global. El cliente decide que HU van a ser implementadas para cada iteración. Además, se realizan las pruebas funcionales, ejecutadas por el cliente, al final de cada iteración. Al final de la última iteración el sistema está listo para ser puesto en producción. Los artefactos generados en la misma son: el Plan de duración por iteración, las Tarjetas CRC (Clase, Responsabilidad y Colaborador) y las Tareas de ingeniería.

Producción: esta fase requiere realizar muchos más chequeos y pruebas al sistema antes de ser entregado al cliente. En esta fase aparecen nuevos cambios y se tiene que decidir si serán incorporados o no en dicha entrega. Durante esta fase suele suceder que las iteraciones se aceleren de tres a una semana. Las ideas pospuestas y las sugerencias son documentadas para luego ser implementadas más adelante, por ejemplo, en la fase de mantenimiento. Los artefactos generados durante la fase son: el Acta de liberación del producto y el Acta de aceptación.

Mantenimiento: el sistema debe mantenerse en funcionamiento al mismo tiempo que desarrolle nuevas iteraciones. Esta fase puede requerir de nuevo personal o cambios en la estructura del equipo.

Muerte del proyecto: hay dos buenas razones por la cual el sistema entre en esta fase. La primera puede ser debido a que el cliente esté muy satisfecho con el sistema y no tenga ninguna otra funcionalidad que agregar en el futuro. La otra razón suele ser que el sistema no termina de ser liberado y el cliente necesita más funcionalidades y es imposible costearlas.

Para el desarrollo de la presente investigación se tuvieron en cuenta todas las fases de la metodología seleccionada, pero solo se aplicaron las 4 primeras, ya que el alcance de la presente investigación es hasta el desarrollo una herramienta informática que permita la identificación de ambigüedades presente en textos de la legislación cubana y las 2 últimas fases se aplican una vez puesta en funcionamiento la solución.

1.6 Herramientas de Ingeniería del Software Asistida por Computadora (CASE)

Las herramientas CASE ayudan a los gestores y practicantes de la ingeniería del software en todas las actividades asociadas a los procesos de software. Automatizan las actividades de gestión de proyectos, los productos de los trabajos elaborados a través del proceso, y ayudan a los ingenieros en el trabajo de análisis, diseño y codificación, permitiendo así obtener resultados adicionales y personalizados que no serán fáciles ni prácticos de producir sin el soporte de herramientas *(Pressman 2002).*

Existen diversas herramientas CASE que soportan el lenguaje de modelado UML (Lenguaje Unificado de Modelado), entre estas se destacan Rational Rose y Visual Paradigm for UML.

Ambas herramientas son muy potentes, y permiten la generación de diversos diagramas como los de clases, de objetos, de casos de uso del negocio y de paquetes, además de generar código a partir de los mismos. Por ser multiplataforma, fácil de usar, proporcionar facilidad de trabajo con modelos UML y guardar todo el modelo en un solo fichero, se selecciona Visual Paradigm para UML en su versión 8.0 para la modelación del proceso de desarrollo de la herramienta propuesta. Además se tuvo en cuenta que la UCI posee una licencia para su uso.

1.7 Lenguajes de programación

Un lenguaje de programación es un conjunto de reglas semánticas, así como sintácticas que los programadores usan para la codificación de instrucciones de un programa o algoritmo de programación. Es un modo práctico para que los seres humanos puedan dar instrucciones a un equipo *(Kanagusico Hernández 2010).*

Son diversos los lenguajes de programación existentes, debido a las características de la herramienta que se desarrollará se analizaron solo los que se emplean para realizar aplicaciones de escritorio. Según el ranking de los lenguajes de programación más populares de 2015 de la IEEE se destacan como más usados Java, C, C++, Python y C# *(Diakopoulos and Cass 2015).*

Luego de analizar los lenguajes de programación más usados y teniendo en cuenta que existe una solución implementada en Java, que sirve de referencia al autor de esta investigación, se selecciona este lenguaje para desarrollar la herramienta propuesta.

Java es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por *Sun Microsystems*. La compañía Sun describe el lenguaje Java con las siguientes características: “simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico”, las misma se describen a continuación *(Castañeda Sanabria 2012)* :

- **Simple:** en Java hay un número reducido de formas claras para abordar una tarea dada. Ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de estas. Hereda la sintaxis de C/C++ y muchas de las características orientadas a objetos de C++ *(Castañeda Sanabria 2012) .*

- **Orientado a objetos:** se realiza una aproximación limpia, útil y pragmática a los objetos. El modelo de objeto de Java es simple y fácil de ampliar *(Castañeda Sanabria 2012*).
- **Distribuido:** posee una extensa capacidad de interconexión TCP/IP permitiendo a los programadores acceder a la información a través de la red *(Castañeda Sanabria 2012*).
- **Robusto:** es fuertemente tipado, por lo que permite comprobar el código en tiempo de compilación y .de ejecución. La liberación de memoria se realiza de forma automática, ya que se proporciona un recolector de basura automático para los objetos que no se utilizan. También proporciona una gestión de excepciones orientada a objetos *(Castañeda Sanabria 2012).*
- **Arquitectura neutral:** Java se compila a un código de bytes de alto nivel independiente de la máquina. Este código está diseñado para ejecutarse en cualquier máquina con un sistema *runtime* (intérprete) el cual si es dependiente de esta (*Castañeda Sanabria 2012)*.
- **Seguro:** Java proporciona seguridad a través de varias características de su entorno en tiempo de ejecución *(Castañeda Sanabria 2012)* :
    - Un verificador de *bytecodes*
    - La disposición de memoria en tiempo de ejecución
    - Restricciones de acceso a los archivos.

  Aunque el compilador solo genere código correcto el intérprete lo vuelve a verificar para asegurarse que el código no ha sido cambiado (intencionadamente o no) entre el momento de la compilación y la ejecución. Además, el intérprete Java determina la disposición de la memoria para las clases. Se puede considerar a Java uno de las aplicaciones más seguras para cualquier sistema (*Castañeda Sanabria 2012)* .
- **Portable:** implementa estándares de portabilidad, los enteros son siempre enteros, el sistema de interfaces de usuario lo constituye un sistema abstracto de ventanas, por lo que es independiente de la arquitectura en la que se implemente (UNIX, PC, Mac) (*Castañeda Sanabria 2012)*.
- **Interpretado:** los programas de Java son interpretados. En lugar de ser compilado en ejecutables nativos, el código de Java es traducido en códigos de *bytes* no asociados a una plataforma, por lo que no son dependientes a ningún entorno de ejecución o sistema operativo *(Castañeda Sanabria 2012)*.

- **Multihilo:** se puede escribir programas que hagan varias cosas a la vez y de una forma fácil y robusta *(Castañeda Sanabria 2012)*.
- **Dinámico:** Java no intenta enlazar todos los módulos que componen una aplicación hasta el tiempo de ejecución. Esto permite enlazar dinámicamente el código de una forma segura y conveniente (*Castañeda Sanabria 2012).*

El estudio de las características del lenguaje de programación Java permitió profundizar en las facilidades que brinda el mismo, con el propósito de ser empleadas en el desarrollo del presente trabajo.

1.8 Entorno de Desarrollo Integrado

En el proceso de desarrollo emplear un IDE resulta ventajoso ya que este brinda muchas facilidades y beneficios, entre los cuales está conocer los ficheros en los que existe algún error de sintaxis.

Un Entorno de Desarrollo Integrado, traducido del inglés *Integrated Development Environment* (IDE) es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien, poder utilizarse para varios. Provee de un marco de trabajo amigable para la mayoría de los lenguajes de programación (*Sanchez 2014).*

Existen diversos IDE entre los que sobresalen Netbeans y Eclipse. Se selecciona para el desarrollo de la solución informática la herramienta Netbeans en su versión 8.0, teniendo en cuenta que es libre, se integra perfectamente con Java que es el lenguaje de programación definido para el desarrollo de la solución propuesta, el equipo posee dominio de esta herramienta, además de un conjunto de elementos que se describen a continuación. Netbeans es un entorno de desarrollo integrado, modular y de base estándar. Consiste en un IDE de código abierto y una plataforma de aplicación, las cuales pueden ser usadas como una estructura de soporte general para compilar cualquier tipo de solución. Presenta una interfaz amigable e intuitiva y tiene todas las herramientas para crear soluciones profesionales ya sean de escritorio, empresariales, web, móviles y aplicaciones SOA (Arquitectura Orientada a Servicios), no solo en Java sino también en C/C++ y Ruby.

NetBeans ya viene con plugins[7] y módulos integrados, evitando tener que configurar el ambiente, brindando todo el entorno listo para trabajar *(Sanchez 2014).*

1.8.1 API Visual

Una API (siglas de *Application Programming Interface*) es un conjunto de reglas (código) y especificaciones que las aplicaciones pueden seguir para comunicarse entre ellas, sirviendo de interfaz entre programas diferentes de la misma manera en que la interfaz de usuario facilita la interacción humano software *(Merino, 2014).*Para el desarrollo de la presente investigación se tuvieron en cuenta Swing y JavaFx.

Swing es una biblioteca de interfaces gráficas de usuario para Java que viene incluida en el entorno de desarrollo. Extiende de una biblioteca más antigua llamada AWT (*Abstract Window Toolkit*, en español Kit de Herramientas de Ventana Abstracta). Está compuesto por un amplio conjunto de componentes de interfaces de usuario que funcionen en el mayor número posible de plataformas (*Montenegro 2012*).

JavaFX es una herramienta que permite a los desarrolladores crear aplicaciones llamadas *Rich Internet Applications* o Aplicaciones de Internet Enriquecidas (RIA). JavaFX está diseñada para proveer una plataforma ligera y acelerada de gráficos para aplicaciones de negocio empresariales, al mismo tiempo que permite a los desarrolladores crear sus aplicaciones completamente en el lenguaje de programación Java. Las aplicaciones JavaFX pueden ser ejecutadas en una amplia variedad de dispositivos (*Guerrero 2015).*

La autora de la presente investigación luego de realizar el estudio de las API visuales decidió emplear JavaFx en vez del tradicional Swing, debido a algunas ventajas que presenta, entre las que se encuentran: brinda mayores facilidades gráficas, trabajan con colecciones altamente sincronizadas, potente, optimizado, interactividad, permite crear animaciones y programación compatible con AJAX[8], Flash de Adobe y Silverlight de Microsoft, multiplataforma, utiliza el mismo lenguaje para la web, para el escritorio y para la telefonía móvil.

[7] plugins: es una aplicación informática que añade funcionalidades específicas a un programa principal.
[8] ajax: es un conjunto de métodos y técnicas que permiten intercambiar datos con un servidor y actualizar partes de páginas web sin necesidad de recargar la página completamente.

1.9 Sistema Gestor de Bases de Datos

Un sistema gestor de base de datos se define como el conjunto de programas que administran y gestionan la información contenida en una base de datos. Permite a los usuarios procesar, describir, administrar y recuperar los datos almacenados en una base de datos. También ayuda al mantenimiento de la integridad de los datos y controla la seguridad y privacidad (*Vallejos 2009).* Existen variados sistemas gestores de bases de datos entre ellos se encuentran MySQL, PostgreSQL, Apache Derby y Oracle.

Luego de analizados cada uno de los gestores de bases de datos antes mencionados y teniendo en cuenta que la solución se desarrolló en el lenguaje Java y se manejó una base de datos pequeña y estática, se decide utilizar Apache Derby por ser un gestor nativo de Java, garantiza un alto nivel de seguridad al estar la base de datos embebida. A continuación, se exponen algunos elementos del gestor seleccionado.

Apache Derby, sub-proyecto Apache DB, es una fuente de base de datos relacional abierto implementado en Java y está disponible bajo la licencia Apache, versión 2.0. Algunas de las ventajas clave incluyen (*Derby 2016)*:

- Tiene alrededor de 2,6 megabytes para el motor de base y el controlador JDBC[9] incorporado (*Derby 2016)*.
- Se basa en los estándares de Java, JDBC y SQL *(Derby 2016)*.
- Proporciona un controlador JDBC incorporado que permite incorporar Derby en cualquier solución basada en Java (*Derby 2016)*.
- Es compatible con el modo cliente / servidor más familiarizado con el conductor derby de red de cliente JDBC y *Derby Network Server (Derby 2016)* .
- Es fácil de instalar, implementar y utilizar (*Derby 2016).*

1.10 Mapeo Objeto-Relacional

En el desarrollo de una aplicación suelen estar involucrados dos elementos diferentes, por una parte el código que mueve la aplicación y por otra los datos que se manejan. Con el

9 JDBC (Java Database Connectivity): API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java.

tiempo el acceso a los datos desde los programas se ha vuelto una tarea en ocasiones, complicada. Los sistemas de Mapeo Objeto-Relacional (por sus siglas en ingles ORM) ayudan a combatir esta complicación.

ORM: es una técnica de programación para convertir datos entre el lenguaje de programación orientado a objetos utilizado y el sistema de base de datos relacional utilizado en el desarrollo de una aplicación (*Carrero 2010).*

Se selecciona para el mapeo de la base de datos de la herramienta propuesta, *EclipseLink* ya que es un ORM de código abierto y el marco de persistencia de objetos está apoyado en un entorno de Java. Soporta varias fuentes y formatos de datos incluyendo las bases de datos relacionales, no relacionales, los servicios XML (Lenguaje de Marcas Extensible), JSON[10] y se puede utilizar en cualquier plataforma Java.

1.11 Patrones de diseño

Los patrones de diseño son descripciones de la comunicación de objetos y clases que pueden personalizarse para resolver un problema de diseño en general en un contexto particular *(Gamma , Helm et al. 1994).*

Los patrones de diseño son soluciones para problemas típicos y recurrentes que pueden aparecer durante el desarrollo de una aplicación, estos ayudan a estandarizar el código y a hacer que el diseño sea más comprensible para otros programadores. Se pueden agrupar en dos grandes grupos; los GRASP (*General Responsibility Assignment Software Patterns*, en inglés), que son patrones generales de software para asignación de responsabilidades y los GOF (*Gang of Four, en inglés*), encargados de la inicialización, agrupación y comunicación de los objetos. En el capítulo 2, se muestra una explicación de los patrones de diseño que fueron empleados en la implementación de la herramienta propuesta.

1.12 Pruebas

El desarrollo de los sistemas de software implica una serie de actividades de producción en las que las posibilidades que el fallo humano aparezca son enormes. Los errores pueden empezar a darse desde el primer momento del proceso, en el que los objetivos pueden estar especificados de forma errónea, debido a la imposibilidad del hombre de trabajar y

[10] JSON (JavaScript Object Notation): formato de texto ligero para el intercambio de datos.

comunicarse de forma perfecta. El desarrollo de software debe ir acompañado de una actividad que garantice la calidad, es por esto que las pruebas de software constituyen un elemento crítico para la garantía de la calidad del software. Estas pruebas se ejecutan dirigidas a componentes o al sistema en su totalidad, con el objetivo de medir el grado en el que este cumple con los requerimientos pactados. Existen cuatro niveles de pruebas:

- **Pruebas de unidad:** se concentra en el esfuerzo de verificación de la unidad más pequeña del diseño, ya sea un componente o un módulo del software *(Pressman 2003*).

- **Pruebas de integración:** es una técnica sistemática para confeccionar la arquitectura del software mientras, al mismo tiempo, se aplican las pruebas para descubrir errores asociados con la interfaz. El objetivo es tomar componentes a los que se les aplicó una prueba de unidad y construir una estructura de programa que determine el diseño *(Pressman 2003).*

- **Pruebas de sistema:** abarca una serie de pruebas diferentes cuyo propósito 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 2003).*

- **Pruebas de aceptación:** una vez culminado el proceso de pruebas por parte del equipo de desarrollo, es indispensable, que el cliente verifique que el producto ha sido desarrollado con las normas y criterios establecidos, y cumple con todos los requisitos especificados por el cliente (*Zapata 2013).*

1.13 Conclusiones parciales

- El análisis de las categorías sintácticas del idioma español, las técnicas del procesamiento del lenguaje natural y la minería de texto, constituyen la base teórica para el desarrollo de la solución propuesta.

- El estudio de las soluciones existentes para la identificación y evaluación de la ambigüedad en los textos, permitió identificar las buenas prácticas que caracterizan las mismas y aplicarlas en el desarrollo de la solución propuesta.

- El estudio de las metodologías y tecnologías utilizadas en el desarrollo de software, permitió seleccionar las más adecuadas a las características de la herramienta propuesta.

CAPÍTULO 2: ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DE LA HERRAMIENTA INFORMÁTICA

2.1 Introducción

En el presente capítulo se describen los principales elementos tenidos en cuenta en el desarrollo de la solución propuesta. Con el uso de la metodología XP se organizó el trabajo en función de las fases que define y se hace referencia también a los artefactos correspondiente a cada una de ellas, entre los que destacan Historias de Usuario (HU), Estimación por esfuerzo y Plan de duración de las iteraciones. Además, en el capítulo se definen los patrones de diseño, la arquitectura, la base de datos y las reglas que permitirán a la herramienta detectar y clasificar las ambigüedades.

2.2 Descripción del proceso de desarrollo

Teniendo en cuenta que en el capítulo 1 se define la metodología XP para guiar el proceso de desarrollo de la presente investigación. A continuación, se describe como fueron aplicadas las fases de exploración, planificación e iteraciones por entrega y los artefactos generados en cada una de ellas. Las fases de producción, mantenimiento y muerte del proyecto no son descritas en este capítulo. La fase producción será descrita en el siguiente capítulo. Las dos últimas fases no serán descritas en la presente investigación debido a que estas se ejecutarán cuando ya la herramienta esté en uso por parte del cliente.

2.2.1 Fase de Exploración

En esta fase se describen los requisitos de la herramienta a través de las HU. También el equipo de desarrollo se familiariza con las tecnologías y herramientas a utilizar en el desarrollo del proyecto.

2.2.1.1 Requisitos de la herramienta

Los requisitos cumplen un papel primordial en el proceso de producción de software, su principal tarea consiste en la generación de especificaciones correctas que describan con claridad el comportamiento del sistema para así minimizar los problemas relacionados al desarrollo de sistemas. Estos se enfocan en la definición de lo que se desea producir y tienen dos clasificaciones: requisitos funcionales y no funcionales. Los requisitos funcionales indican lo que debe hacer la solución propuesta, es decir, lo que el sistema

hace para el usuario. Los requisitos no funcionales, por su parte, se refieren a propiedades que debe cumplir el software, tales como capacidad de almacenamiento, fiabilidad o mantenibilidad. A continuación, se listan los requisitos de la herramienta propuesta.

Requisitos funcionales

RF1: Importar documento.
RF2: Mostrar datos del documento importado.
RF3: Detectar ambigüedad sintáctica.
RF4: Detectar ambigüedad léxica.
RF5: Detectar ambigüedad semántica.
RF6: Mostrar cantidad de ambigüedades léxicas detectadas.
RF7: Mostrar cantidad de ambigüedades sintácticas detectadas.
RF8: Mostrar cantidad de ambigüedades semánticas detectadas.

Requisitos no funcionales

- **Requisitos de usabilidad**

RNF 1: La herramienta siempre tendrá visible la opción de Ayuda, lo que posibilitará un mejor aprovechamiento de sus funcionalidades

RNF 2: El usuario no tendrá que dar más de tres clics para acceder a las funcionalidades de la herramienta.

RNF 3: La herramienta debe proporcionar mensajes que sean informativos y orientados al usuario final.

RNF 4: La herramienta debe contar con una interfaz amigable y con botones que tengan nombres sugerentes para que usuarios inexpertos puedan interactuar fácilmente con el software.

RNF 5: Emplear el mismo formato de letra en toda la herramienta.

- **Requisito de rendimiento**

RNF 6: La herramienta no debe tardar más de seis segundos en mostrar los resultados de una petición hecha por el usuario.

- **Requisito de software**

RNF 7: Se debe tener instalada en la computadora la Máquina Virtual de Java en su versión 8 o superior y un lector de formato de documento portátil (PDF) para el funcionamiento de la herramienta.

RNF 8: La herramienta puede ejecutarse en los sistemas operativos Windows, Linux y OS X.

- **Requisitos de hardware**

RNF 9: Para ejecutar la herramienta la computadora debe contar con las siguientes características:

- 512 MB de memoria RAM.
- Procesador a 3.2 Ghz, equivalente o superior.
- Capacidad de 1 GB de disco duro.

2.2.1.2 Historias de usuario

La HU es la técnica utilizada en XP para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer. El tratamiento de las HU es muy dinámico y flexible, en cualquier momento pueden romperse, reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada HU es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas *(Letelier and Penadés 2006)*. A continuación, se describen las dos de HU de prioridad alta en el negocio.

Tabla 1. HU “Importar documento”.

Historia de Usuario	
Número: 1	**Nombre de la Historia de Usuario:** Importar documento
Modificación a la Historia de Usuario: Ninguna	

Usuario: Yamila Ortuño Sánchez	**Iteración asignada:** 1
Prioridad en negocio: Alta	**Puntos estimados:** 1 semana
Riesgo en desarrollo: Bajo	**Puntos reales:** 1 semana
Programador responsable: Yamila Ortuño Sánchez	
Descripción: **Importar documento:** permite importar el texto de un documento.	
Observaciones: **Importar documento:** admite documentos en las extensiones (*.doc, *.docx, *.odt, pdf).	

Tabla 2. HU "Detectar ambigüedades".

Historia de Usuario	
Número: 3	**Nombre de la Historia de Usuario:** Detectar ambigüedades
Modificación a la Historia de Usuario: Ninguna	
Usuario: Yamila Ortuño Sánchez	**Iteración asignada:** 1
Prioridad en negocio: Alta	**Puntos estimados:** 3 semanas
Riesgo en desarrollo: Bajo	**Puntos reales:** 3 semanas
Programador responsable: Yamila Ortuño Sánchez	
Descripción:	

Detectar ambigüedades: permite detectar las ambigüedades presentes en un texto ya sea importado o redactado en la herramienta, haciendo uso de las técnicas: diccionario perteneciente al PLN y pre procesamiento de los documentos correspondiente a la minería de texto.
Observaciones: **Detectar ambigüedades:** Anteriormente ya el usuario debe haber redactado un texto en la herramienta o haberlo importado desde un documento.

2.2.2 Fase de Planificación

En esta fase el cliente establece la prioridad de cada historia de usuario y los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Esta fase dura unos pocos días *(Letelier and Penadés 2006).*

2.2.2.1 Estimación de esfuerzo por historias de usuario

La estimación de esfuerzo asociado a la implementación de las historias de usuario la establecen los programadores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Si la estimación supera las 3 semanas, la HU deberá ser dividida hasta que pueda ser desarrollada en un tiempo factible. En caso de que el esfuerzo sea menor que una semana, la HU será combinada por otra *(Letelier and Penadés 2006).* En la siguiente tabla se muestra la estimación de esfuerzos por HU para el desarrollo de la herramienta propuesta:

Tabla 3. "Estimación de esfuerzo por HU".

Historias de usuario	Puntos de estimación
Importar documento	1
Mostrar datos del documento	1
Detectar ambigüedades	3
Mostrar cantidad de ambigüedades	1

2.2.3 Fase de Iteraciones por entrega

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Al final de la última iteración el sistema estará listo para entrar en producción *(Letelier and Penadés 2006).* A continuación, se describen cada una de las iteraciones propuestas.

Iteración #1: en esta iteración se implementaron las HU 1 y 2, las cuales se refieren a importar el texto de un documento y mostrar los datos del documento.

Iteración #2: en esta iteración se implementaron las HU 3 y 4, las cuales se refieren a los requisitos funcionales detectar ambigüedad sintáctica, detectar ambigüedad léxica, detectar ambigüedad semántica, mostrar cantidad de ambigüedades léxicas detectadas, mostrar cantidad de ambigüedades sintácticas detectadas y mostrar cantidad de ambigüedades semánticas detectadas.

2.2.3.1 Plan de duración de las iteraciones

Después de tener la estimación de esfuerzo por HU, es necesario crear el plan de duración para cada una de las iteraciones definidas, que tiene como objetivo mostrar la duración y el orden en que serán implementadas las HU dentro de cada iteración. Para la implementación de la herramienta propuesta se necesita un total de 6 semanas, dividas en 2 iteraciones, como se muestra en la tabla 4.

Tabla 4. Plan de duración de las iteraciones.

Iteraciones	Historias de usuario	Duración
1	Importar documento	2 semanas
	Mostrar datos del documento	
2	Detectar ambigüedades	4 semanas
	Mostrar cantidad de ambigüedades	

2.2.3.2 Tarjetas CRC (Clase, Responsabilidad y Colaboración)

Las tarjetas CRC son una técnica para el diseño de software orientado a objetos creada por Kent Beck y Ward Cunningham. Se utilizan, individualmente, para representar objetos. La clase del objeto puede ser escrita en la parte superior de la tarjeta, las responsabilidades se colocan en la parte izquierda y las clases que colaboran son listadas a la derecha de cada responsabilidad (*Zambrano 2014).* A continuación, se detalla la tarjeta CRC de la clase principal de la herramienta llamada *GestorAnalisis*.

CRC Tarjeta	
Clase: GestorAnalisis	
Descripción: Esta clase crea una instancia de los analizadores, se encarga de gestionarlos y le brinda la información a la clase PrincipalController	
Responsabilidades:	
Nombre	Colaborador
GestorAnalisis	accesoDiccionario LinkedList
getAccesoDiccionario	accesoDiccionario
detectarAmbiguedad	accesoDiccionario LinkedList

Figura 1. Tarjeta CRC “Analizador de ambigüedades” (Fuente: elaboración propia).

2.2.3.3 Tareas de ingeniería

Las tareas de ingeniería le permiten a los desarrolladores obtener un nivel de detalle más avanzado por las HU. A continuación se describe la tarea de ingeniería 3, que tiene como objetivo implementar la HU Detectar ambigüedades.

Tabla 5. Tarea de ingeniería 3.

	Tarea de ingeniería

Número de tarea: 3	**Historia de Usuario (No.3):** Detectar ambigüedades
Nombre de tarea: Implementar HU_ Detectar ambigüedades	
Tipo de tarea: Desarrollo	**Puntos estimados: 3** semanas
Fecha de inicio: 4-04-2016	**Fecha de fin:** 24-04-2016
Programador responsable: Yamila Ortuño Sánchez	
Descripción: este requisito debe permitir detectar las ambigüedades presentes en los textos redactados o importados en la herramienta.	

2.3 Diseño de la herramienta

La metodología de desarrollo de software XP propone que se realicen diseños simples y sencillos, que permita un fácil entendimiento a los desarrolladores, permitiendo reducir el tiempo al realizar las tareas de implementación.

2.3.1 Arquitectura

La arquitectura de software representa la estructura o estructuras del sistema que consiste en componentes de software, las propiedades externas visibles de esos componentes y las relaciones entre ellos *(Ruiz 2011)*.Para el desarrollo de la presente investigación se utilizó el patrón arquitectónico programación por capas o también conocida como n-capas. Esta arquitectura permite separar la lógica de negocios de la lógica de diseño, su principal ventaja es que se puede llevar a cabo varios niveles y en caso de haber algún cambio, solo se realiza en el nivel requerido sin necesidad de revisar en todo el código de la herramienta propuesta.

Las capas creadas son: Presentación, Negocio, Datos y Útiles. La capa presentación es la que interactúa con el usuario, le comunica la información, captura los datos y le muestra los resultados del análisis. Esta capa se comunica únicamente con la de negocio. La capa negocio es la encargada de recibir la información de la capa presentación, realizar el procesamiento de los datos y enviar una respuesta a la misma, además contiene las funcionalidades necesarias para el proceso de evaluación de las ambigüedades. Esta capa se comunica también con la capa de datos para acceder a los datos necesarios para

detectar las ambigüedades. La capa datos es la encargada de acceder a los datos que solicita la capa de negocio para detectar las ambigüedades que se encuentran en la base de datos. La capa útiles contiene las funcionalidades comunes para todos los paquetes. Seguidamente se muestra la estructura por paquetes de la solución propuesta siguiendo la arquitectura empleada.

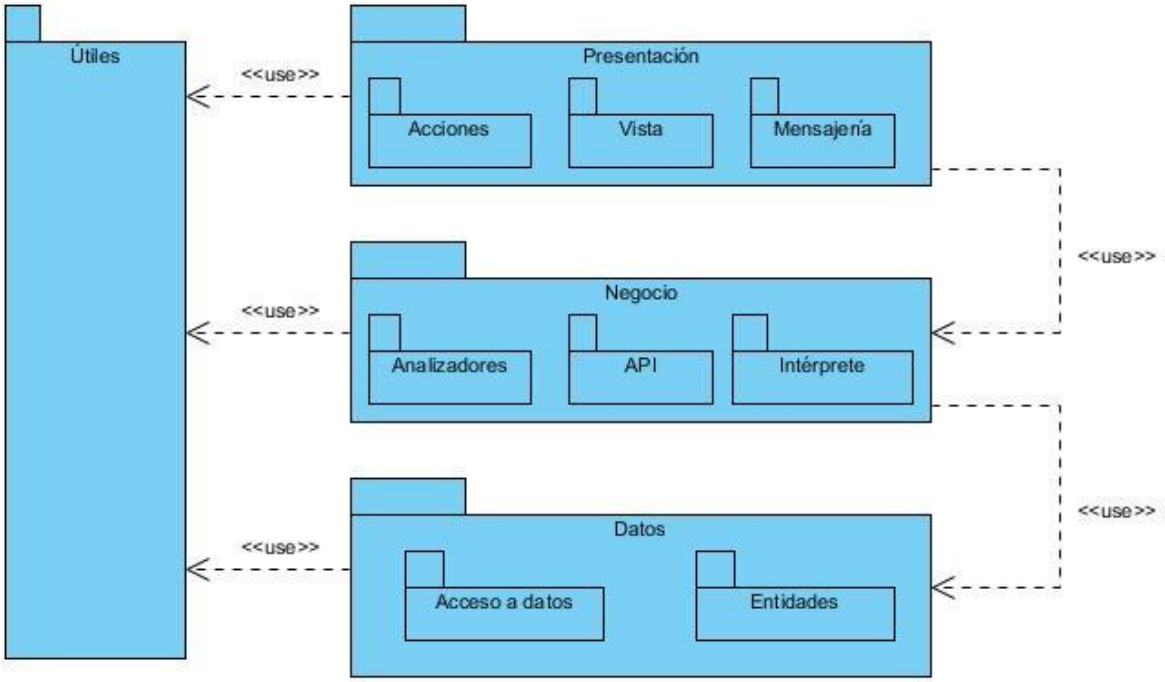


Figura 2. Diagrama de paquetes de la herramienta (Fuente: elaboración propia).

El paquete de *Presentación* está compuesto por los paquetes *Acciones, Vista y Mensajería*. El paquete *Vista* contiene la interfaz que le permitirá al usuario interactuar con la herramienta. El paquete *Acciones* contiene la clase que controla la interfaz de usuario. El paquete *Mensajería* contiene la clase necesaria para el tratamiento de mensajes de la herramienta. El paquete de Negocio está compuesto por los paquetes *Analizadores, API e Intérprete*. El paquete *Analizadores* contiene tres clases, cada una detecta un tipo de ambigüedad y la clase *GestorAnalisis* es la encargada de realizar llamadas a estas clases para detectar las ambigüedades. El paquete *API* contiene la interfaz de la que heredan los analizadores. El paquete *Intérprete* se encarga de realizar el procesamiento del texto para evaluar la ambigüedad. El paquete de *Datos* está compuesto por los paquetes *AccesoDatos y Entidades. El paquete Acceso a Datos* contiene las clases necesarias para el acceso a la base de datos y de realizar las consultas en la misma. El paquete *Entidades* contiene las

clases encargadas de mapear la base de datos. Paralelo a estos paquetes se encuentra el paquete *Útiles* que contiene las funcionalidades comunes para todos los paquetes, tales como constantes predefinidas, las clases contenedoras de datos y la clase *AnalizadorWord* que es la encargada de extraer el contenido de un documento en formato .*odt.

2.3.2 Patrones de diseño

Para diseñar la herramienta se emplearon un conjunto de patrones de diseño, los cuales son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software. A continuación, se describen los patrones empleados:

Patrones de diseño GRASP

- **Experto:** se pone en práctica con el uso de clases que poseen responsabilidades específicas a cumplir, de acuerdo con la información que manejan. El uso de este patrón se evidencia en la clase *AccesoDiccionario*.

- **Creador:** se refleja en las clases que tienen la responsabilidad de instanciar objetos de otras clases. El uso de este patrón se evidencia en la clase *GestorAnalisis* y *PrincipalController*.

- **Controlador:** es el experto coordinando el trabajo de otros expertos. Este patrón se evidencia en la clase *GestorAnalisis* que la encargada de buscar los analizadores y del acceso a la base de datos para obtener finalmente los resultados.

- **Alta cohesión:** se aplica para realizar un diseño que evite contener clases con un alto grado de abstracción, que asuman responsabilidades que podían haber delegado a otros objetos o que tengan responsabilidades complejas. El patrón se evidencia en cada de una de las clases de la herramienta, de tal forma que se elimina la sobrecarga de responsabilidades.

- **Bajo acoplamiento:** el acoplamiento mide la fuerza con que una clase está conectada a otra, de esta forma una clase con bajo acoplamiento debe tener un número mínimo de dependencia con otras clases. Este patrón se tuvo presente debido a la importancia que se le atribuye a realizar un diseño de clases independientes que puedan soportar los cambios de una manera fácil y que a su

vez permitan la reutilización. El patrón se evidencia en cada una de las clases diseñadas para la herramienta.

Patrones de diseño GOF

- **Instancia única (Singleton):** presenta un mecanismo para limitar el número de instancias de una clase. Su uso se aprecia en las clases *GestorAnalisis,* proporcionando una instancia única a la clase PrincipalController de la capa de Presentación, existiendo así, un solo objeto de las clases encargadas de la gestión del negocio.

2.3.3 Estándares de codificación

El estándar de codificación empleado en el desarrollo de la herramienta fue definido por el equipo de desarrollo. A continuación, se muestran algunas pautas de dicho estándar.

- Todas las nomenclaturas a utilizar se definirán en idioma español.
- Los nombres de los paquetes y las clases serán con mayúscula, en caso de ser un nombre compuesto las siguientes palabras se escribirán de igual forma.
- Los nombres de los métodos serán con minúscula, en caso de ser un nombre compuesto las siguientes palabras se escribirán con mayúscula.
- Los identificadores para las variables y los parámetros serán con letras en minúsculas y en caso de ser un nombre compuesto las siguientes palabras se escribirán con mayúscula.
- Las clases gestoras de negocio comienza con el prefijo Gestor y luego el nombre de la clase (GestorAnalisis.java).
- Las clases de acceso a datos comienzan con el prefijo Acceso y luego el nombre de la clase (AccesoDiccionario.java).
- Los nombres de variables o funciones deben ser lo suficientemente descriptivos, sin exceder de 30 caracteres.
- Todas las funciones deben tener comentarios explicando que realiza cada una de ellas.

2.3.4 Diagrama de clases

Un diagrama es un tipo de estructura estática que describe como está compuesto un sistema mostrando las clases del mismo, sus atributos, operaciones y las relaciones entre los objetos. En la figura 3 se muestra el diagrama de clases del diseño definido en la presente investigación, acotado solo a sus clases y relaciones. En el anexo 1 se observa el diagrama con sus clases, relaciones y sus atributos.

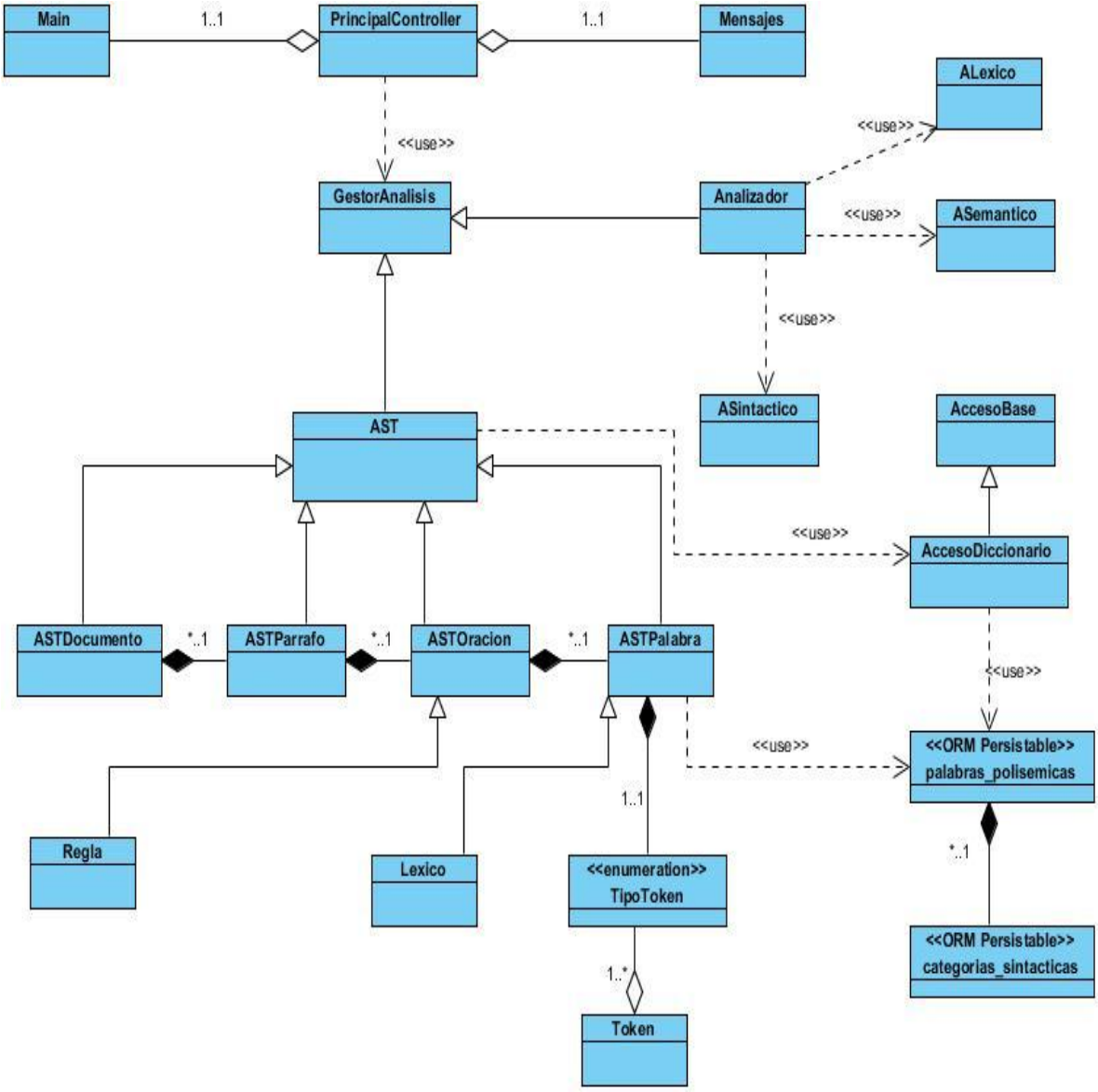


Figura 3. Diagrama de clases del diseño acotado a sus clases y relaciones (Fuente: elaboración propia).

El diagrama de clases muestra las relaciones entre las clases de la herramienta propuesta. La clase *GestorAnalisis* es la encargada del negocio de la aplicación, manejando el flujo de información para detectar y evaluar las ambigüedades. La clase *PrincipalController* es la encargada de controlar la vista y mostrar la información al usuario. Las clases *palabras_polisemicas* y *categorias_sintacticas* son las encargadas de mapear la base de datos; mientras que las clases *AccesoBase* y *AccesoDiccionario* son las encargadas de realizar las peticiones a la base de datos.

2.4 Componentes para la identificación de ambigüedades

Para la identificación de las ambigüedades en un texto se implementaron tres componentes, cada uno está basado en reglas heurísticas definidas a partir del estudio realizado en el capítulo 1 sobre las ambigüedades y sus tipos.

2.4.1 Analizador sintáctico

El analizador sintáctico se encarga de detectar las ambigüedades sintácticas presentes en un texto. A continuación, se detallan las reglas definidas para el análisis de este tipo de ambigüedad:

- **Regla 1**: si una oración contiene más de una conjunción y dichas conjunciones pertenecen al grupo de conjunciones coordinantes copulativas (y, e, ni, que), entonces la frase presenta *ambigüedad coordinativa copulativa*.

- **Regla 2:** si una oración contiene más de una conjunción y dichas conjunciones pertenecen al grupo de conjunciones coordinantes disyuntivas (o, u, sea, bien), entonces la frase presenta *ambigüedad coordinativa disyuntiva*.

- **Regla 3:** si una oración contiene al menos una conjunción coordinante disyuntiva y al menos una conjunción coordinante copulativa, entonces la frase presenta *ambigüedad coordinativa mixta*.

- **Regla 4:** si una oración contiene al menos una preposición separable (a, con, de, en) que sea sintácticamente ambigua, entonces la frase presenta *ambigüedad preposicional*.

- **Regla 5:** si una oración contiene una conjunción subordinante causal (porque, ya que, puesto que, como), entonces la oración presenta ambigüedad subordinante.

2.4.2 Analizador léxico

El analizador léxico se encarga de detectar las ambigüedades léxicas presentes en un texto. A continuación, se detalla la regla definida para el análisis de este tipo de ambigüedad:

- **Regla:** si una oración contiene una palabra con más de un significado, que indique diferentes categorías sintácticas (sustantivo, verbo, adjetivo), entonces la oración presenta ambigüedad léxica categorial.

2.4.3 Analizador semántico

El analizador semántico se encarga de detectar las ambigüedades semánticas presentes en un texto. A continuación, se detalla la regla definida para el análisis de este tipo de ambigüedad:

- **Regla:** si una oración contiene una palabra con más de un significado, en función del contexto, entonces la oración presenta ambigüedad léxica semántica.

2.5 Descripción de la base de datos

Las reglas que emplean los componentes semántico y léxico para el análisis de las ambigüedades presentes en los textos hacen uso de una base de datos compuesta por palabras polisémicas y sus categorías sintácticas, que responde dentro del PLN a los métodos basados en conocimiento que emplean diccionarios.

La base de datos está compuesta por las tablas *palabras_polisemicas*, *categorias_sintacticas* y *polisemicas_categorias*. La tabla *palabras_polisemicas* contiene las palabras polisémicas. La tabla *categorias_sintacticas* contiene las categorías sintácticas de las palabras. La tabla *polisemicas_categorias* se obtiene a partir de la relación de las dos tablas anteriores. La tabla *palabras_polisemicas* contiene un total de 170 palabras polisémicas relacionadas con el ámbito jurídico, las cuales fueron introducidas por la autora del presente trabajo, con el propósito de hacer posible el funcionamiento de la herramienta

propuesta. En la siguiente figura se muestra el diagrama físico de la base de datos.

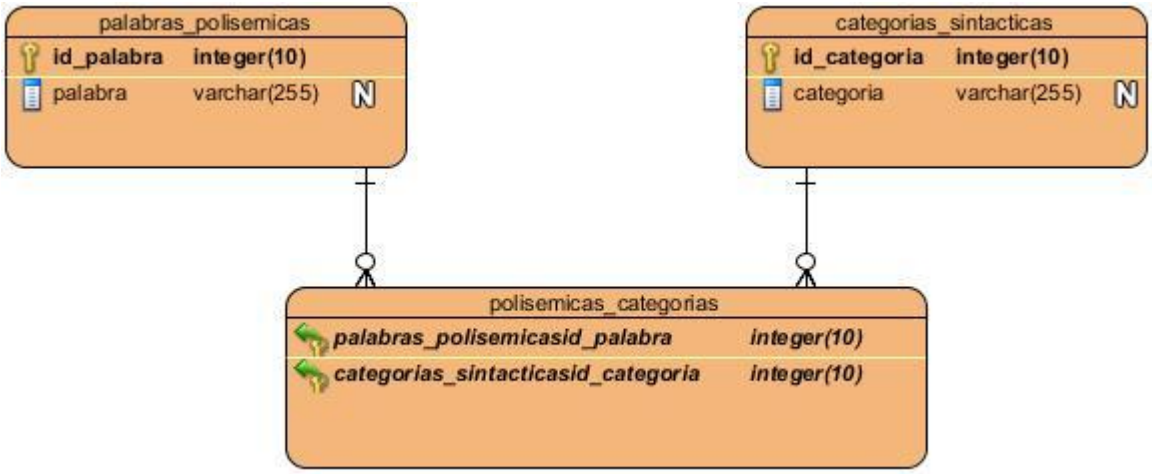


Figura 4. Diagrama físico de la base de datos de la herramienta (Fuente: elaboración propia).

2.6 Descripción del sistema

La herramienta informática propuesta permite identificar y clasificar las ambigüedades existentes en los textos de la legislación cubana. La misma brinda la opción de redactar los textos o importarlos desde un documento con las extensiones (*.doc, *.docx, *.odt y *pdf). Una vez importado el documento, muestra los datos del mismo (título, autor, comentarios, cantidad de caracteres, cantidad de palabras, cantidad de páginas) apoyándose de la minería de textos con el empleo de la técnica pre-procesamiento de los documentos. El usuario tiene la opción de observar la cantidad de ambigüedades total o por tipo.

2.7 Conclusiones parciales

- El empleo de la metodología XP en función de describir el proceso de desarrollo de la herramienta permitió realizar un trabajo organizado y estructurado, generando los artefactos de cada una de sus fases, en correspondencia con el cronograma de desarrollo propuesto.
- El uso de una arquitectura n-capas y el empleo de patrones de diseño, garantizan obtener una solución de software con poca dependencia entre clases, flexible al mantenimiento y la aceptación de cambios.

- La estructuración de la herramienta en componentes que implementan cada una de las reglas heurísticas definidas, constituye la base para obtener una solución capaz de elevar la identificación de ambigüedades en textos de la legislación cubana.

CAPÍTULO 3: PRUEBAS DE LA HERRAMIENTA INFORMÁTICA

3.1 Introducción

En el presente capítulo se describe la fase producción de la metodología de desarrollo empleada, en la cual se le realizaron chequeos y pruebas a la herramienta desarrollada antes de ser entregada al cliente. Además, se exponen los artefactos obtenidos en esta fase, tales como: el acta de liberación y el acta de aceptación. En el capítulo también se realiza una validación de los resultados de la herramienta en relación al cumplimiento del objetivo general de la investigación.

3.2 Técnica de validación de los requisitos

Con el objetivo de demostrar que los requerimientos previamente definidos cumplen con las expectativas del cliente, se aplicaron las técnicas de validación de requisitos:

- **Revisiones formales de los requisitos:** se realizaron revisiones formales a cada uno de los requisitos por parte del cliente y del equipo de desarrollo, obteniéndose un total de 3 no conformidades de tipo técnica, de redacción y formato, las que fueron corregidas en tiempo, generándose un Acta de Aceptación por parte del cliente de este encuentro (ver Anexo 2).

- **Construcción de prototipos:** se confeccionaron prototipos no funcionales, dando la posibilidad al cliente de poder comprobar de forma visual cómo quedaría la herramienta, obteniéndose finalmente un prototipo que satisface al cliente.

3.3 Validación del diseño

Para comprobar la calidad del diseño se emplearon las métricas de diseño relaciones entre clases (RC) y tamaño operacional de clase (TOC).

3.3.1 Relaciones entre clases (RC)

La métrica RC está dada por el número de relaciones de uso de una clase con otra. El primer paso es evaluar un conjunto de atributos de calidad entre los que se encuentran el acoplamiento, complejidad de mantenimiento y reutilización de cada clase *(Pressman 2002*).

A continuación, se explican los pasos que se llevaron a cabo para aplicar la métrica:

1. Determinar la cantidad de relaciones de uso (CRU) que poseen las clases a medir.

2. Calcular el promedio de las CRU.

3. Teniendo en cuenta los valores antes obtenidos se determina la incidencia de los atributos de calidad en cada una de las clases, según los criterios expuestos en la tabla 6.

Tabla 6. Rango de valores para medir la afectación de los atributos de calidad (RC).

Atributos de calidad	Clasificación	Criterio
Acoplamiento	Ninguna	CRU = 0
	Baja	CRU = 1
	Media	CRU = 2
	Alta	CRU > 2
Complejidad de mantenimiento	Baja	CRU <= Promedio
	Media	Promedio < CRU < = 2* promedio
	Alta	CRU > 2* promedio
Reutilización	Baja	CRU > 2* promedio
	Media	Promedio < CRU < = 2* promedio
	Alta	CRU <= Promedio
Cantidad de pruebas	Baja	CRU <= Promedio
	Media	Promedio < CRU < = 2* promedio
	Alta	CRU > 2* promedio

En la siguiente figura se muestra el resultado de la aplicación de la métrica RC

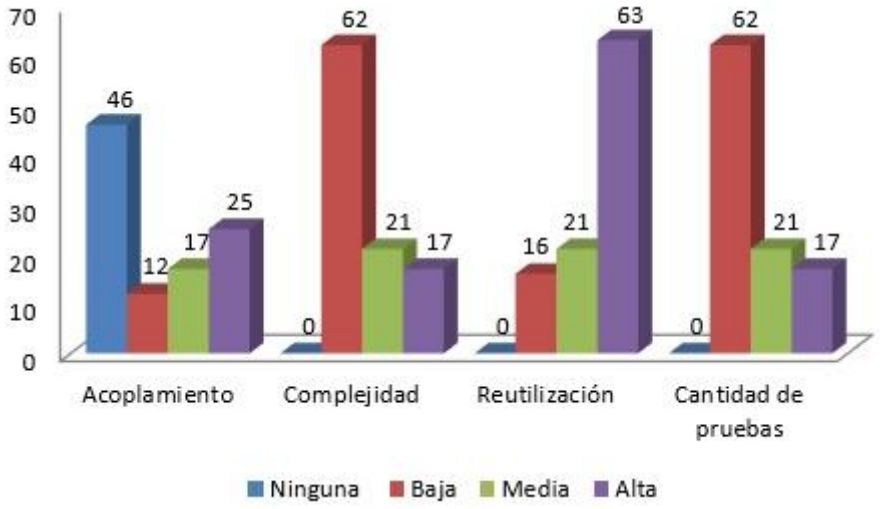


Figura 5. Representación en (%) de los resultados de la aplicación de la métrica RC (Fuente: elaboración propia).

Acoplamiento: según los resultados que se muestran, el 46% de las clases no posee relaciones de uso por lo que la mayoría de las clases no tienen valores de acoplamiento, validando una realización correcta del diseño.

Complejidad de mantenimiento: según los resultados que se muestran en la figura anterior, el 62% de las clases se comportan de forma satisfactoria pues, son de fácil soporte.

Reutilización: según los resultados que se mostraron en la figura, el 63% de las clases tiene un alto grado de reutilización.

Cantidad de pruebas: luego de aplicar la métrica se obtuvo que el 62% de las clases poseen un bajo grado de esfuerzo a la hora de realizar cambios, rectificaciones y pruebas de software.

Según lo analizado anteriormente, los valores de RC se comportan de forma satisfactoria. Estos resultados expresan que las clases del diseño de la herramienta presentan bajo acoplamiento, la complejidad de mantenimiento y la cantidad de pruebas son bajas y en consecuencia el grado de reutilización es alto.

3.3.2 Tamaño Operacional de clases (TOC)

La métrica TOC fue aplicada a cada una de las clases del diseño con el objetivo de medir la calidad de las mismas con respecto a su grado de responsabilidad, complejidad de implementación y reutilización *(Pressman 2002*).

A continuación se explican los pasos que se llevaron a cabo para aplicar la métrica:

1. Cálculo del umbral. El umbral se toma del tamaño general de una clase que se determina sumando todas las operaciones que posee.

2. Calcular el promedio de los umbrales.

3. Teniendo en cuenta los valores antes obtenidos se determina la incidencia de los atributos de calidad en cada una de las clases, según los criterios expuestos en la tabla 7.

Tabla 7. Rango de valores para medir la afectación de los atributos de calidad (TOC).

Atributos de calidad	Clasificación	Criterio
Responsabilidad	Baja	Umbral<= Promedio
	Media	Promedio < Umbral < = 2* promedio
	Alta	Umbral > 2* promedio
Complejidad de implementación	Baja	Umbral <= Promedio
	Media	Promedio < Umbral < = 2* promedio
	Alta	Umbral > 2* promedio
Reutilización	Baja	Umbral > 2* promedio
	Media	Promedio < Umbral < = 2* promedio
	Alta	Umbral <= Promedio

.

En la siguiente figura se muestra el resultado de la aplicación de la métrica TOC.

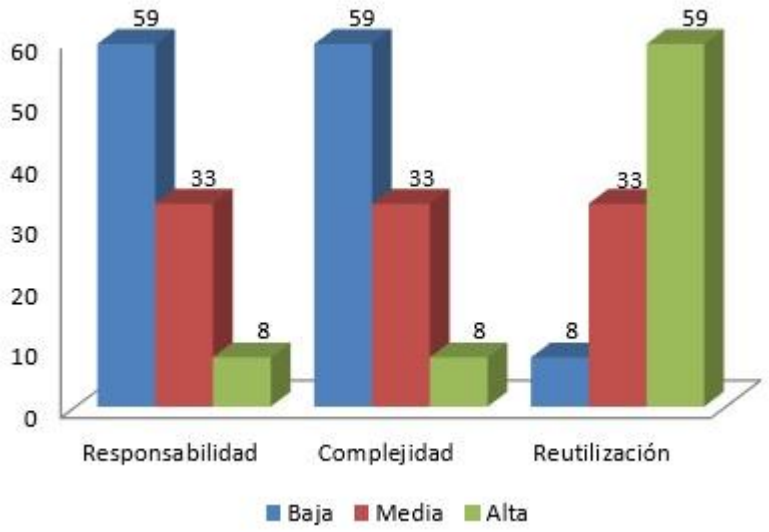


Figura 6. Representación en (%) de los resultados de la aplicación de la métrica TOC (Fuente: elaboración propia).

Responsabilidad: luego de aplicar la métrica se obtuvieron resultados satisfactorios que reflejan una responsabilidad baja con un valor del 59%.

Complejidad de Implementación: después de haberse realizado la medición de la métrica, arrojó también resultados positivos ya que la complejidad de las clases es baja en un 59 %.

Reutilización: se obtuvieron valores que según muestra la gráfica de la figura anterior se comporta en un nivel alto con un 59%

Los resultados arrojados por la métrica TOC expresan que las clases del diseño de la herramienta presentan una elevada reutilización y baja complejidad y responsabilidad. Por lo que se concluye que los resultados obtenidos en esta métrica son positivos.

3.4 Pruebas

En todo proceso de desarrollo de software es indispensable la realización de pruebas para garantizar el buen funcionamiento y la calidad del producto final. Una de las principales fortalezas de la metodología de desarrollo XP es el proceso de pruebas, al realizarse de

manera continua, garantizan que los errores sean detectados en un corto plazo de tiempo y se corrijan de manera más sencilla, lo cual permite asegurar el éxito del producto.

En la validación de la herramienta se utilizaron los niveles de pruebas de unidad y aceptación, definidos en el Capítulo 1. Para realizar las pruebas de unidad se aplicaron los métodos de caja negra y caja blanca. Las pruebas de aceptación fueron realizadas con el cliente.

3.4.1 Pruebas de unidad

Al desarrollar un nuevo software la primera etapa a considerar es la de las pruebas unitarias, también llamadas pruebas modulares ya que permiten determinar si un módulo del programa está listo y correctamente terminado, las mismas no se deben confundir con las pruebas informales que realiza el programador mientras está desarrollando *(Oré 2009*). Como parte de las pruebas unitarias se aplicaron a la herramienta los métodos de caja blanca y caja negra.

El método de caja blanca garantiza que se ejerciten por lo menos una vez todos los caminos independientes del código, así como la ejecución de todos los bucles en sus límites operacionales y todas las decisiones lógicas en las vertientes verdaderas y falsas (*Pressman 2003).* Para aplicar las mismas se empleó la técnica del camino básico. Se tomó como ejemplo el método *detectarAmbiguedad,* perteneciente a la clase *GestorAnalisis* como base para realizar el procedimiento anteriormente descrito. La selección del método se realizó teniendo en cuenta la complejidad del mismo, el cual da lugar a una de las principales funcionalidades que responden al objetivo general de la investigación. Esta permitió obtener una medida de la complejidad lógica para el diseño de los casos de prueba (CP) y usar dicha medida como guía para la definición de un conjunto básico de caminos de ejecución, en la Figura 7 se muestra este método.

```
public LinkedList<Ambiguedad> detectarAmbiguedad(String texto, int[] cantidad_oraciones) throws IOException {

    LinkedList<Ambiguedad> ambiguedades = new LinkedList<Ambiguedad>();
    Sintactico sintactico = new Sintactico(texto, ambiguedades);
    ASTDocumento documento = sintactico.documento();
    List<ASTParrafo> parrafos = documento.getParrafos();
    for (ASTParrafo parrafo : parrafos) {
        List<ASTOracion> oraciones = parrafo.getOraciones();
        for (ASTOracion oracion : oraciones) {
            for (int i = 0; i < listaAnalizadores.size(); i++) {
           listaAnalizadores.get(i).detectarAmbiguedad(oracion, ambiguedades, acceso_diccionario);


            }

        }
    }
    cantidad_oraciones[0] = documento.cantidadOraciones();
    return ambiguedades;
}
```

Figura 7. Método *detectarAmbiguedad* de la clase *GestorAnalisis* (Fuente: elaboración propia).

A continuación, se describen los pasos que se realizaron para desarrollar la técnica del camino básico.

1. **Confeccionar el grafo de flujo:** usando el código de la figura 7 se realizó la representación del grafo de flujo, el cual está compuesto por los siguientes elementos:
   - Nodos: son círculos que representan una o más sentencias procedimentales.
   - Aristas: son flechas que representan el flujo de control y son análogas a las flechas del diagrama de flujo.
   - Regiones: son las áreas delimitadas por aristas y nodos

En la figura 8 se muestra el grafo de flujo obtenido.

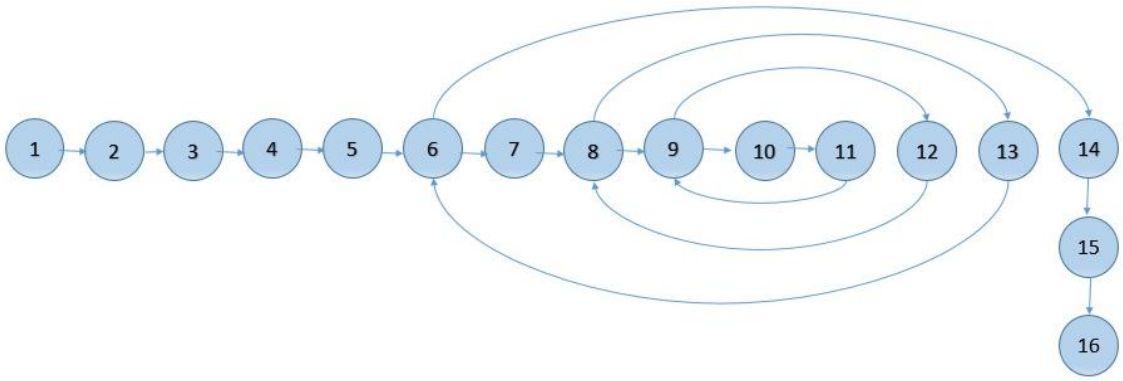


Figura 8. Grafo del camino básico del método *detectarAmbiguedad* (Fuente: elaboración propia).

2. **Calcular la complejidad ciclomática:** proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado define el número de caminos independientes del conjunto básico de un programa, la complejidad ciclomática se calculó con la siguiente fórmula:
    - V (G)= A-N+2, donde A es el número de aristas del grafo de flujo y N es el número de nodos del mismo.

    V (G) = A-N+2 = 18-16+2 = 4

3. **Determinar un conjunto básico de caminos linealmente independientes:** el valor de V (G) da el número de caminos linealmente independientes de la estructura de control del programa, por lo que se definen los siguientes 4 caminos:

    Camino básico #1: 1-2-3-4-5-6-14-15-16
    Camino básico #2: 1-2-3-4-5-6-7-8-13-6-14-15-16
    Camino básico #3: 1-2-3-4-5-6-7-8-9-12-8-13-6-14-15-16
    Camino básico #4: 1-2-3-4-5-6-7-8-9-10-11-9-12-8-13-6-14-15-16

4. **Obtención de casos de pruebas (CP):** cada camino independiente es un caso de prueba a realizar, de forma que los datos introducidos provoquen que se visiten las sentencias vinculadas a cada nodo del camino. En este caso se obtuvieron 4 caminos básicos, por tanto, se hace necesario la confección de igual número de CP, para aplicar las pruebas a este método. A continuación, se muestra uno de ellos:

Tabla 8. Caso de prueba de caja blanca para el camino básico #4.

Caso de prueba: Camino básico #4	
Entrada	El texto que va a ser analizado.
Resultados esperados	Una lista con las ambigüedades presentes en el texto.
Condiciones	Debe haber un párrafo con al menos una oración y debe haber algún analizador implementado en la lista de analizadores.

Una vez ejecutados todos los casos de pruebas obtenidos a través de la aplicación de la técnica camino básico, se concluye que los mismos fueron probados satisfactoriamente demostrando que el código generado no presenta ciclos infinitos y no existe código innecesario en el sistema desarrollado.

Los métodos de caja negra son pruebas funcionales que se realizan al software. En la validación de la solución propuesta se realizaron pruebas funcionales a la herramienta por parte de los especialistas del Grupo de Calidad del Centro de Gobierno Electrónico, quienes utilizaron la técnica de particiones equivalentes y valores límites con el empleo de 2 CP elaborados por el equipo de desarrollo. El objetivo de aplicar este método es: encontrar errores de funciones incorrectas o ausentes, de interfaz o en accesos a la base de datos. A continuación, en la tabla 9 se muestra el CP definido para la HU Importar documento.

Tabla 9. Caso de prueba de caja negra de la HU “Importar documento”

Escenario	**Descripción**	**Documento**	**Respuesta del sistema**
EC 1.1 Importación correcta.	Importar texto de un documento con una extensión correcta.	V Formato de documento *.doc, *.docx, *.odt y *.pdf.	Muestra el texto.
EC 1.2: Importación incorrecta.	Importar texto de un documento con una extensión incorrecta.	I Solo admite el formato de documento *.doc, *.docx, *.odt y *.pdf.	El archivo seleccionado no tiene una extensión válida.

Con el objetivo de comprobar que las funcionalidades de la herramienta se realizaron correctamente y responden a las necesidades del cliente, el método se aplicó en dos iteraciones. En la primera se detectaron un total de 8 No Conformidades (NC), predominando entre ellas las de tipo interfaz, al finalizar la iteración todas estas NC quedaron resueltas. En la segunda iteración los resultados fueron satisfactorios, obteniéndose cero NC, generándose así el Acta de Liberación Interna de Productos de Software (ver Anexo 3), artefacto que avala la calidad del software liberado. La Figura 9 ilustra los resultados de aplicar el método de caja negra, teniendo en cuenta los tipos de NC identificados (interfaz, funcionalidad, redacción y recomendación).

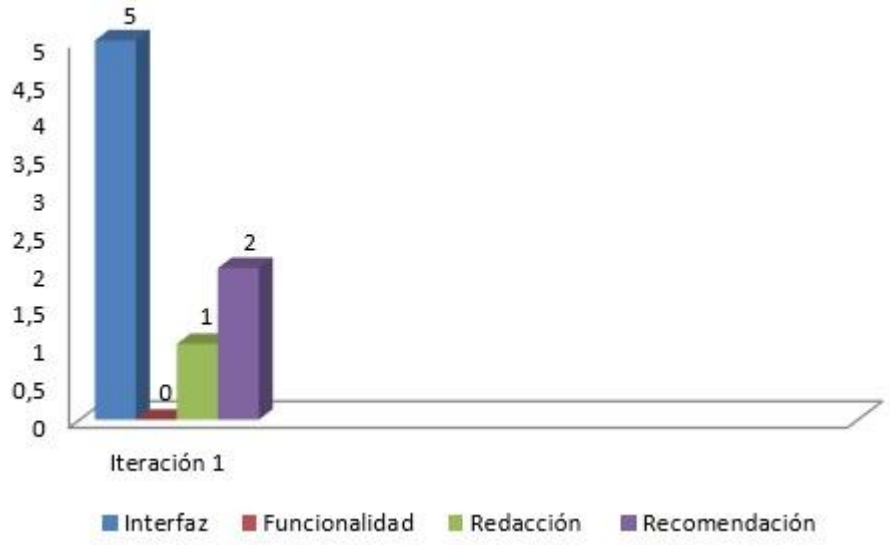


Figura 9. No conformidades detectadas al aplicar la técnica de caja negra (Fuente: elaboración propia).

3.4.2 Pruebas de aceptación

La prueba de aceptación es generalmente desarrollada y ejecutada por el cliente o un especialista de la aplicación y es conducida a 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. Está considerada como la fase final del proceso, para crear un producto confiable y apropiado para su uso *(Pressman 2010).*

Para la aplicación de estas pruebas se confeccionó un caso de prueba de aceptación por cada HU. Seguidamente se muestra el caso de prueba de aceptación de la HU “Detectar ambigüedades”.

Tabla 10. Caso de prueba de Aceptación de la HU "Detectar ambigüedades".

<table>
<tr><td colspan="3">Caso de prueba de aceptación</td></tr>
<tr><td colspan="2">Código de caso de prueba: 03</td><td>Nombre historia de usuario: Detectar ambigüedades.</td></tr>
<tr><td colspan="3">Nombre de la persona que realiza el caso de prueba: Yamila Ortuño Sánchez</td></tr>
<tr><td colspan="3">Descripción de la prueba: revisar a través de la herramienta el correcto funcionamiento de los RF detectar ambigüedad sintáctica, detectar ambigüedad léxica y detectar ambigüedad semántica.</td></tr>
<tr><td colspan="3">Condiciones de ejecución: se debe tener un texto previamente cargado o redactado en la herramienta.</td></tr>
<tr><td colspan="2">Entrada/Pasos de ejecución</td><td rowspan="2">Resultados esperados:</td></tr>
<tr><td>Acción:</td><td>Entrada:</td></tr>
<tr><td>Se selecciona una de las siguientes opciones: "Detectar ambigüedad"

"Detectar "</td><td>Texto.</td><td>El sistema debe mostrar un listado de ambigüedades detectadas.</td></tr>
<tr><td colspan="3">Evaluación de prueba: Satisfactoria</td></tr>
</table>

Se realizó un encuentro con la MSc. Yarina Amoroso Fernández, presidenta de la Sociedad Cubana de Derecho e Informática, con el objetivo de revisar la herramienta, teniendo en cuenta los CP definidos. Los resultados obtenidos fueron satisfactorios, avalados por la especialista entrevistada como un aporte importante para el desarrollo de la informática jurídica en Cuba. Al finalizar el encuentro se generó el Acta de Aceptación del Producto (ver Anexo 4).

3.5 Validación del resultado de la herramienta

La investigación realizada plantea como idea a defender que: “El desarrollo de una herramienta informática permitirá elevar la capacidad de identificación de ambigüedades presentes en los textos de la legislación cubana, contribuyendo a mejorar la comprensión de los mismos.”

El análisis del cumplimiento de la variable independiente *“elevar la capacidad de identificación de ambigüedades presentes en los textos de la legislación cubana”,* se realiza a partir de la evaluación de un conjunto de criterios de medida definidos por la autora del presente trabajo. Estos permiten validar el cumplimiento del objetivo general de la investigación, en función de comparar cómo a través de la solución obtenida se resuelven las limitantes de la herramienta anterior.

Criterios de medida definidos:

- Tipo de arquitectura
- Facilidad de mantenimiento y escalabilidad
- Cantidad de ambigüedades analizadas
- Tipo de formato de texto que importa
- Estructura del diccionario de palabras
- Cantidad de ambigüedades identificadas

A continuación, en la tabla 11, se evalúan cada uno de los criterios de medida definidos. La evaluación se realiza entre el antes y después del desarrollo de la herramienta informática propuesta, con el propósito de verificar cómo la herramienta desarrollada eleva la capacidad de identificación de ambigüedades presentes en los textos de la legislación cubana.

Tabla 10. Evaluación de los criterios de medida definidos para comprobar el cumplimiento de la variable independiente.

Criterio de medida	Antes	Después
Tipo de arquitectura	La herramienta anterior presenta una arquitectura no modular, en la cual las reglas se encontraban implícitas en el código.	La herramienta actual presenta una arquitectura modular basada en componentes, donde cada uno de estos se encarga de analizar un tipo

		de ambigüedad en específico. Lográndose así un desacoplamiento de las clases.
Facilidad de mantenimiento y escalabilidad	El diseño no modular de esta herramienta hace compleja la mantenibilidad y escalabilidad de la misma.	La estructura modular de esta herramienta permite realizar cambios sin afectar las funcionalidades ya existentes. Haciéndose fácil la mantenibilidad y escalabilidad de la misma.
Cantidad de ambigüedades analizadas	La herramienta anterior identifica 6 tipos de ambigüedades.	La herramienta desarrolla identifica 7 tipos de ambigüedades, con la inclusión del análisis de la subordinante causal.
Tipo de formato de texto que importa	La herramienta anterior importa 3 tipos de formatos de textos, sin incluir el formato PDF.	La herramienta actual importa 4 tipos de formatos de textos, incluyendo el formato PDF. Esto permite poder analizar a través de la misma una mayor gama de documentos jurídicos, teniendo en cuenta que la mayoría de estos se encuentran en formato PDF.
Estructura del diccionario de palabras	El diccionario de palabras que utiliza la herramienta anterior, se encuentra limitado de términos y su estructura no tiene una total correspondencia con las categorías sintácticas que identifican las ambigüedades léxicas y semánticas.	El diccionario de palabras que compone la nueva herramienta está basado en las categorías sintácticas que caracterizan las ambigüedades léxicas y semánticas y el número de términos que lo componen es alto.
Cantidad de ambigüedades identificadas	La herramienta anterior identifica un número menor de ambigüedades en relación a la herramienta actual.	El número de ambigüedades identificados por la herramienta actual es mayor que las identificadas por la herramienta anterior, lo cual quedó

		demostrado al analizar el texto del DECRETO-LEY No. 281 "DEL SISTEMA DE INFORMACION DEL GOBIERNO". Análisis que al ser realizado por ambas herramienta, demostró que la herramienta actual obtiene un 16 % más de ambigüedades que la anterior.

Los criterios de medida analizados en la tabla anterior, demuestran que la solución obtenida permite elevar la capacidad de identificación de ambigüedades presentes en los textos de la legislación cubana, teniendo en cuenta que los criterios evaluados son superiores en la nueva herramienta.

3.6 Conclusiones parciales

- La aplicación de técnicas de validación de requisitos, métricas de validación del diseño, pruebas unitarias y de aceptación a la solución propuesta, certifican la obtención de un software funcional que responde a los requerimientos del cliente, avalado en cada caso por las actas de liberación y aceptación emitidas.
- La validación del resultado de la herramienta a partir de la definición de un conjunto de indicadores certifica el cumplimiento del objetivo general de la investigación.

CONCLUSIONES GENERALES

- El estudio de las categorías sintácticas que originan ambigüedad en los textos, las características del lenguaje español y las soluciones existentes en la actualidad para la identificación de ambigüedades, constituyen la base para la definición de las reglas heurísticas que fundamentan el desarrollo de la herramienta propuesta.

- El empleo de patrones de diseño, el uso de una arquitectura n-capas y la estructuración de la solución en componentes que implementan un conjunto de reglas heurísticas definidas a partir de las características del lenguaje español, posibilitaron obtener una herramienta informática que aumenta la identificación de ambigüedades en textos de la legislación cubana.

- La comparación de ambas herramientas a partir de un conjunto de indicadores previamente definidos, demostró que la solución obtenida permite elevar la capacidad de identificación de ambigüedades presentes en los textos de la legislación cubana.

- El resultado del trabajo contribuye al fortalecimiento del desarrollo de la informática jurídica en Cuba, constituyendo un resultado dentro del Grupo de Investigación de Informática Jurídica del Centro de Gobierno Electrónico.

RECOMENDACIONES

- Extender la herramienta informática con nuevas reglas heurísticas que permitan identificar otros tipos de ambigüedades.
- Incorporar a la herramienta nuevas funcionalidades que brinden al usuario una propuesta de desambiguación a partir de las ambigüedades identificadas.

BIBLIOGRAFÍA REFERENCIADA

Bisbal, E., A. Molina, L. Moreno, F. Pla, M. Saiz-Noeda and E. Sanchis (2003). 3LB-SAT: Una herramienta de anotación semántica.

Brun, R. E. and J. A. Senso. (2004). "Minería Textual." from http://eprints.rclis.org/11491/1/Artmineriapdf.pdf.

Calabria, L. and P. Píriz (2003). Metodología XP.

Carrazana Galán, D. and D. Marimón Baullosa (2015). Herramienta informática para la evaluación de la ambigüedad en textos legales.

Carrero, A. (2010). Conceptos básicos de ORM (Object Relational Mapping)

Castañeda Sanabria, E. (2012). Java Basico.

Derby, A. (2016, Marzo 4). "Apache Derby." from https://db.apache.org/derby/.

Diakopoulos, N. and S. Cass. (2015). "Interactive: The Top Programming Languages 2015." from http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2015.

Gallo, E., M. Mansilla, A. Yague, J. Garbajosa and X. Lacurrea (2010). Informe valorativo acerca del uso de metodologías ágiles en comunidades de desarrollo de sotware libre.

Gamma , E., R. Helm, R. Johnson and J. Vlissides (1994). Design Patterns.Elements of Reusable Object-Oriented Software.

Guerrero, N. (2015). "¿Qué es JavaFx ?", 2016.

Gutiérrez Arcones, D. (2015). Estudio sobre el texto jurídico y su traducción: características de la traducción jurídica, jurada y judicial.

Haller, J., A. Donoso and Y. Ramírez (2002). MPRO- Un programa para el análisis morfológico y sintáctico de textos en español.

Haro, S. N. G. and A. Gelbukh (2007). INVESTIGACIONES EN ANÁLISIS SINTÁCTICO PARA EL ESPAÑOL. México.

Kanagusico Hernández, D. E. (2010). Introducción a la programación.

Letelier, P. and M. C. Penadés. (2006). "Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP)." from http://www.cyta.com.ar/ta0502/v5n2a1.htm.

Manterola, I., A. Díaz de Ilarraza, K. Gojenola and K. Sarasola (2010). Recursos en euskera para la herramienta NLTK para enseñanza de procesamiento del lenguaje natural.

Montenegro, M. (2012). Interfaces gráficas con swing.

Morales García, I. (2015). Metodologías de desarrollo software.¿ Tradicional o Ágil?

Navarro Cadavid, A., J. D. Fernández Martínez and J. Morales Vélez (2013). Revisión de metodologías ágiles para el desarrollo de software.

Oré, A. (2009). UNIT TESTING - PRUEBAS UNITARIAS - CAP 1.

Orozco, D. (2014, junio 16). "Lingüistica." from http://conceptodefinicion.de/linguistica/.

Peñalver, G., A. Meneses and S. García (2010). SXP, metodología ágil para el desarrollo de software.

Pressman, R. S. (2002). Ingeniería del Software. Un Enfoque Práctico. Quinta Edición. España, McGraw-Hill.

Pressman, R. S. (2003). Ingeniería del Software. Un enfoque práctico. Sexta Edición, Mc Graw Hill.

Pressman, R. S. (2010). Ingeniería del Software.

Ramos, S. T. (2009). "Optimización global de coherencia en la desambiguación del sentido de las palabras." from http://www.gelbukh.com/thesis/Sulema%20Torres%20Ramos%20-%20PhD.pdf.

Ramos, S. T. (2012). "Estudio sobre métodos de tipo lesk usados para la desambiguación de sentidos de palabras." from http://rcs.cic.ipn.mx/2012_47/Estudio%20sobre%20metodos%20tipo%20Lesk%20usados%20para%20la%20desambiguacion%20de%20sentidos%20de%20palabras.pdf.

Ruiz, F. (2011). "Software Architecture." from http://epf.eclipse.org/wikis/openupsp/openup_basic/guidances/concepts/software_architecture,_O7tAMVvEduLYZUGfgZrkQ.html.

Sanchez, A. (2014). "Entorno Desarrollo Integrado (IDE)."

Santibáñez, J. M. (2015). Fundamentos de las metodologías en la ingeniría del software.

Suárez Cueto, A. (2004). Resolución de la ambigüedad semántica de las palabras mediante modelos de probabilidad de máxima entropía. Alicante.

Tello Leal, E. (2009). La Desambiguación del Sentido de las Palabras: revisión metodológica.

Valderrama Guayan, F. E. and R. Benites Barrientos (2014). Desarrollo de un sistema informático web para la gestión de producción de calzados de la empresa Jaguar S.A.C. utilizando la metodología AUP y tecnología ASP.NET framework MVC3. Desarrollo de un sistema informático web para la gestión de producción de calzados de la empresa Jaguar S.A.C. utilizando la metodología AUP y tecnología ASP.NET framework MVC3.

Vallejos, S. J. (2009). Sistemas de BD en Dispositivos Móviles y su integración con las BD tradicionales, Universidad Nacional del Nordeste.

Zambrano (2014). ¿Qué son las tarjetas CRC?

Zapata, C., K. Palomino and R. Rosero. (2007). "Portal de Revistas UN.", from http://www.revistas.unal.edu.co/index.php/dyna/article/view/1764/11536.

Zapata, J. (2013). Niveles de prueba del software.

ANEXOS

Anexo 1: Diagrama de clases

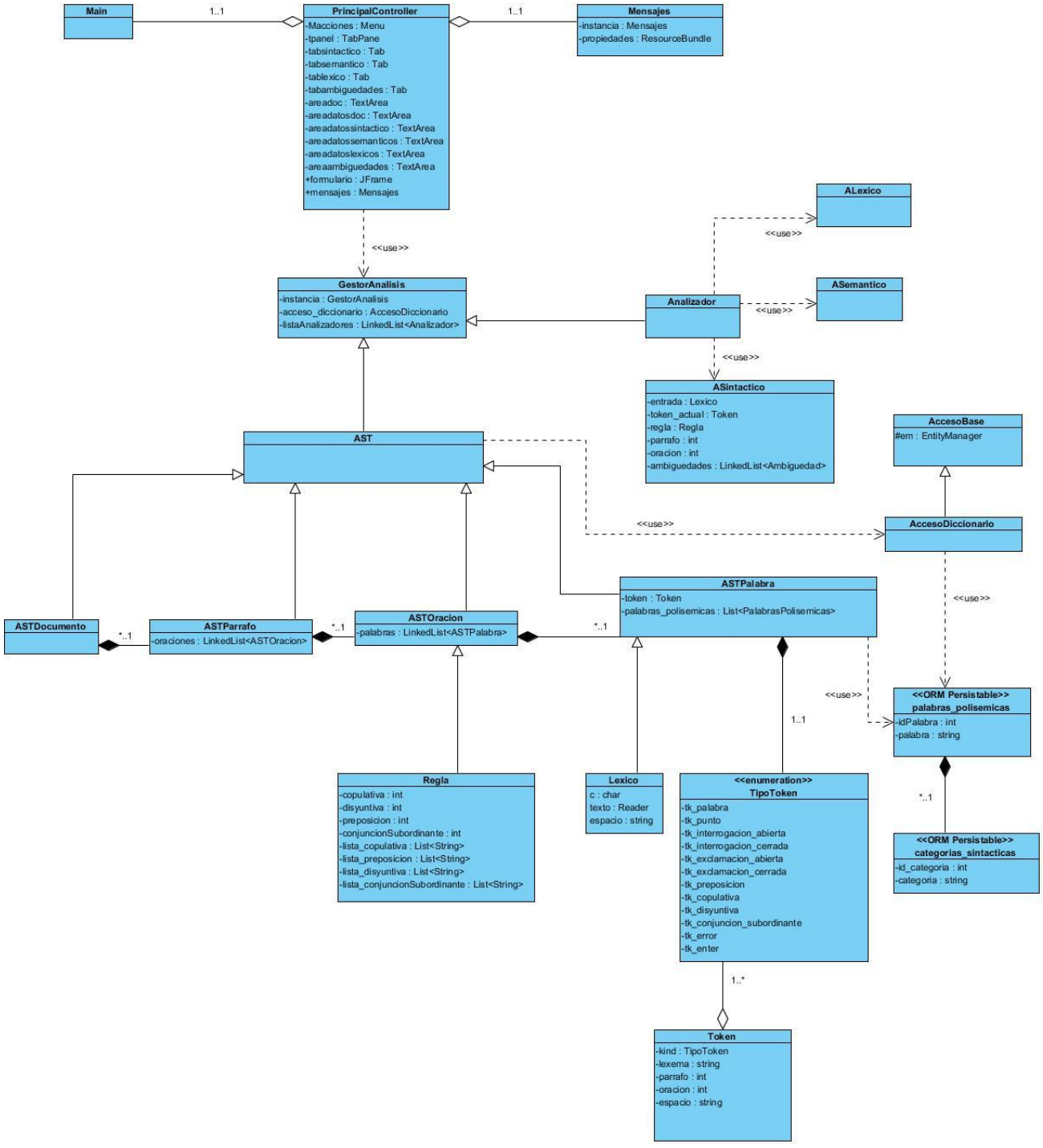
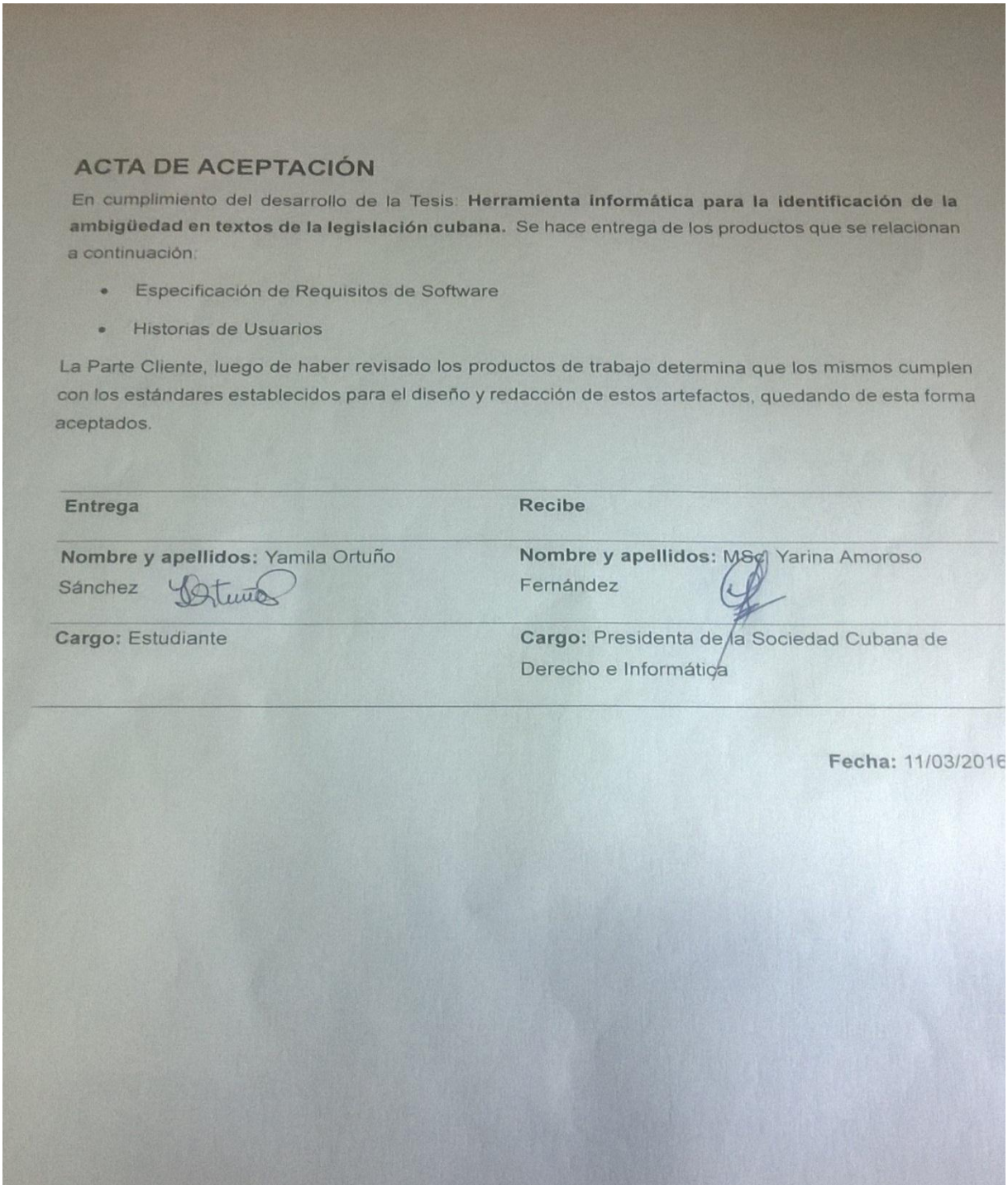


Figura 10. Diagrama de clases del diseño de la herramienta (Fuente: elaboración propia).

Anexo 2: Acta de Aceptación de la revisión de los requisitos.



ACTA DE ACEPTACIÓN

En cumplimiento del desarrollo de la Tesis: **Herramienta informática para la identificación de la ambigüedad en textos de la legislación cubana.** Se hace entrega de los productos que se relacionan a continuación:

- Especificación de Requisitos de Software
- Historias de Usuarios

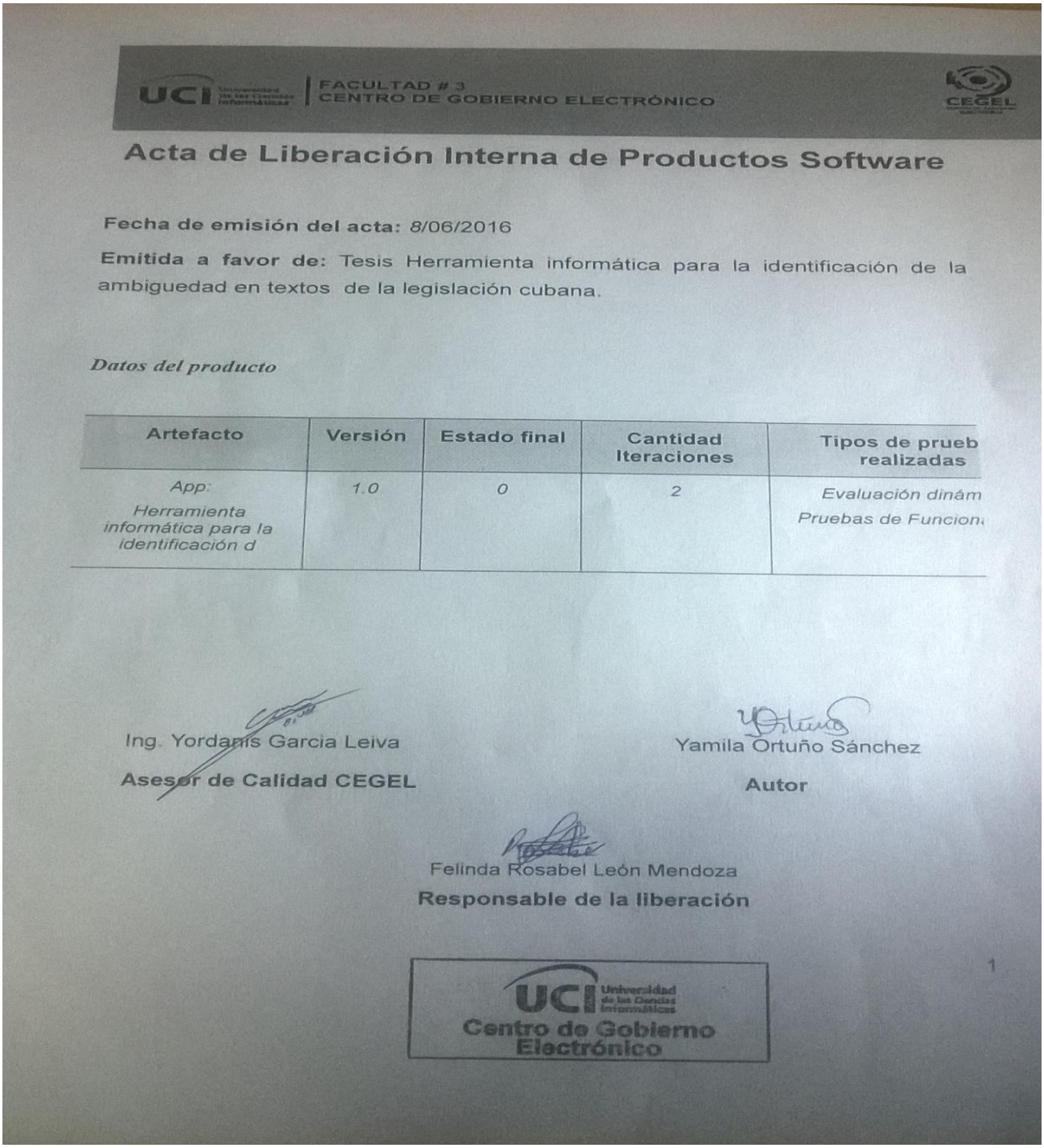
La Parte Cliente, luego de haber revisado los productos de trabajo determina que los mismos cumplen con los estándares establecidos para el diseño y redacción de estos artefactos, quedando de esta forma aceptados.

Entrega	**Recibe**
Nombre y apellidos: Yamila Ortuño Sánchez	**Nombre y apellidos:** MSc. Yarina Amoroso Fernández
Cargo: Estudiante	**Cargo:** Presidenta de la Sociedad Cubana de Derecho e Informática

Fecha: 11/03/2016

Figura 11. Acta de aceptación de los requisitos de software y las historias de usuario.

Anexo 3: Acta de Liberación Interna de Productos de Software.



UCI Universidad de las Ciencias Informáticas | FACULTAD # 3 CENTRO DE GOBIERNO ELECTRÓNICO — CEGEL

Acta de Liberación Interna de Productos Software

Fecha de emisión del acta: 8/06/2016

Emitida a favor de: Tesis Herramienta informática para la identificación de la ambiguedad en textos de la legislación cubana.

Datos del producto

Artefacto	Versión	Estado final	Cantidad Iteraciones	Tipos de prueb realizadas
App: Herramienta informática para la identificación d	1.0	0	2	Evaluación dinám Pruebas de Funcion

Ing. Yordanis Garcia Leiva
Asesor de Calidad CEGEL

Yamila Ortuño Sánchez
Autor

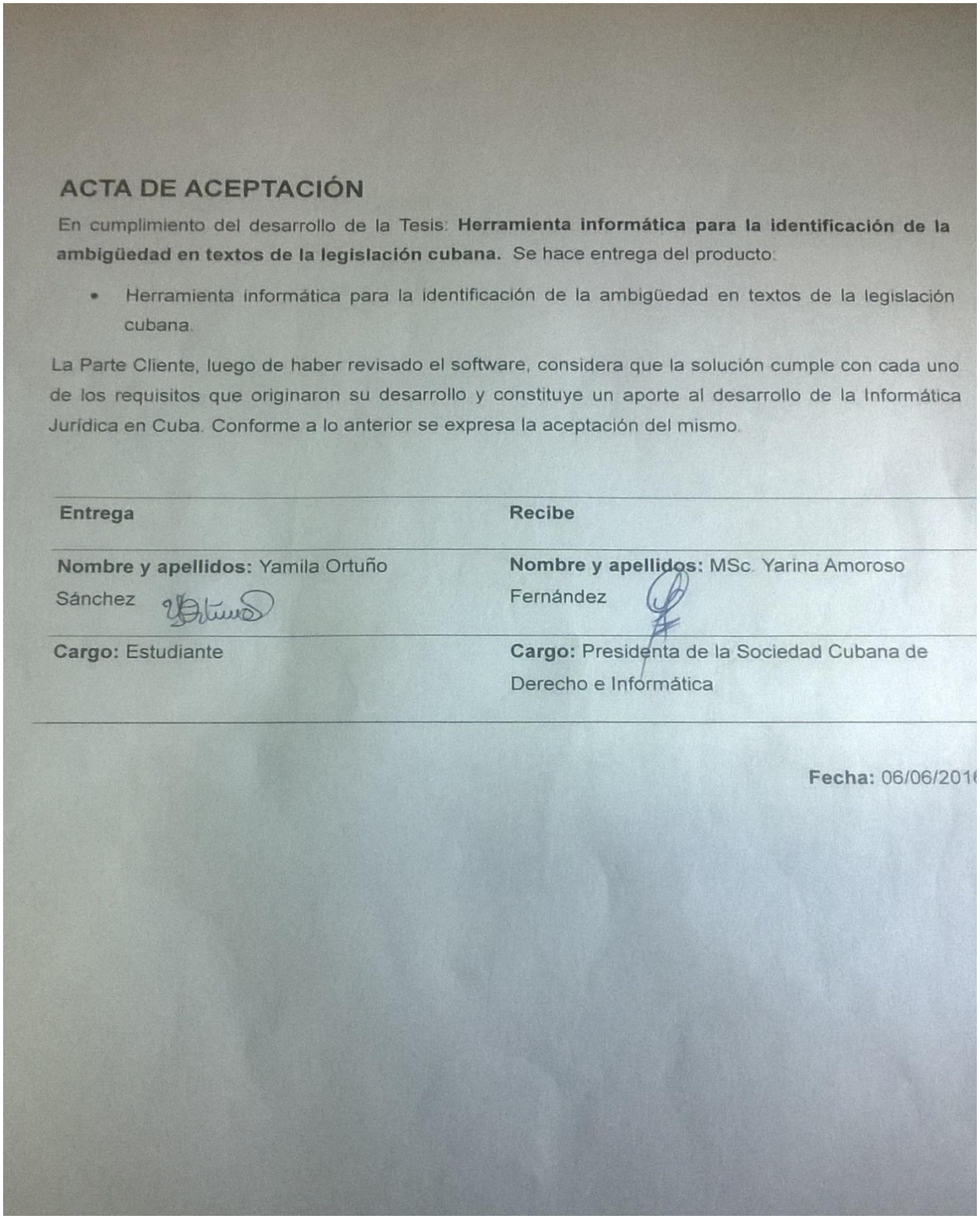
Felinda Rosabel León Mendoza
Responsable de la liberación

UCI Universidad de las Ciencias Informáticas
Centro de Gobierno Electrónico

1

Figura 12. Acta de liberación interna de productos de software de la herramienta.

Anexo 4: Acta de aceptación de la herramienta por parte del cliente.



ACTA DE ACEPTACIÓN

En cumplimiento del desarrollo de la Tesis: **Herramienta informática para la identificación de la ambigüedad en textos de la legislación cubana.** Se hace entrega del producto:

- Herramienta informática para la identificación de la ambigüedad en textos de la legislación cubana.

La Parte 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 aporte al desarrollo de la Informática Jurídica en Cuba. Conforme a lo anterior se expresa la aceptación del mismo.

Entrega	**Recibe**
Nombre y apellidos: Yamila Ortuño Sánchez	**Nombre y apellidos:** MSc. Yarina Amoroso Fernández
Cargo: Estudiante	**Cargo:** Presidenta de la Sociedad Cubana de Derecho e Informática

Fecha: 06/06/201

Figura 13. Acta de aceptación del cliente por parte del cliente.