UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA

MAESTRÍA Y DOCTORADO EN CIENCIAS E INGENIERÍA

"EVALUACIÓN AUTOMÁTICA DE REQUERIMIENTOS FUNCIONALES DE SOFTWARE MEDIANTE PROCESAMIENTO DE LENGUAJE NATURAL"

TESIS

QUE PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS

PRESENTA:

CARLOS ALBERTO HUERTAS VILLEGAS

TIJUANA, BAJA CALIFORNIA ENERO 2013

Prefacio

"Quien nunca se ha equivocado,
Nunca ha intentado algo nuevo"
Albert Einstein

Cuando ingrese a la carrera de Ingeniero en Computación (2003) siempre presenté un gran interés sobre la interacción avanzada con las máquinas; en ese momento no tenía bases para comprender cómo podría lograrse, pero conforme iba avanzando en la carrera se despertó más ese interés.

Fue en 2008, cuando después de ver la película de *Iron Man* en donde el personaje plática con una computadora *súper avanzada,* me di cuenta que me gustaría trabajar en el área del Procesamiento de Lenguaje Natural y resolver problemas de la computación con este enfoque.

En esta tesis planteo una propuesta utilizando Procesamiento de Lenguaje Natural para ayudar a minimizar problemas de calidad de requerimientos en al área de Ingeniería de Software, y así ayudar a realmente tener un desarrollo sistemático y formal en el desarrollo de requerimientos.

Resumen

Evaluación automática de requerimientos funcionales de Software mediante procesamiento de lenguaje natural

Investigaciones han demostrado que la mayor parte de los errores en el desarrollo de software pueden ser ligados a los requerimientos, por lo tanto, el aseguramiento de la calidad de los mismos es de vital importancia. La problemática surge cuando un proyecto crece tanto, que la cantidad de requerimientos y los recursos necesarios para su evaluación son demasiado grandes para hacer práctica su evaluación. Por esto mismo se han buscado formas más eficientes de realizar la evaluación de requerimientos.

En este documento se presenta una propuesta de solución para la evaluación automática de requerimientos funcionales, específicamente los atributos de calidad: ambigüedad, atomicidad, completitud. Con esto se busca poder reducir la carga de trabajo necesario, así como también eliminar el potencial error humano que siempre estará presente en una evaluación manual. Para lograr esto se ha definido una estructura gramatical de la información esperada en un requerimiento funcional. Una vez teniendo la estructura, se ha creado una gramática formal construida mediante expresiones regulares, que es evaluada mediante procesamiento de lenguaje natural de forma automática.

Los resultados obtenidos hasta el momento han sido satisfactorios logrando un alto índice que detección de problemas, y una buena relación entre la evaluación automática y una evaluación humana. Los problemas atendidos en esta investigación son ambigüedad, conjunción e incompletitud.

Summary

Automatic Software Functional Requirement
Evaluation using natural language processing

Researchers have proved that most of software development cycle errors can be traced to requirements, therefore, the quality of these requirements are very important. The main problem arises when a project grows up to the point where the amount of requirements and the required resources to ensure their quality become more than the developers are willing to spend. Given the stated problematic, developers have seek for more efficient ways to review and check requirements.

In this document is reviewed a proposal for automatic requirements evaluation for software functional requirements, specifically for the quality attributes: ambiguity, atomicity, completeness. The main goal is to reduce the amount of work and resources needed for this task, besides the error prone human evaluation can be backed up with a formal approach. To achieve this goal, a formal structured grammar based on regular expressions has been defined; later this grammar is checked by natural language processing techniques to evaluate requirements.

The results so far have been successful, a high ratio of detected problems has been achieved, and a good relationship between the automatic and the human evaluation has been detected too. The main problems in review under this research are ambiguity, conjunction and incompleteness.

Agradecimientos

Sin ningún orden particular primeramente quisiera agradecer a mis padres por todo el apoyo que siempre he recibido para completar mis estudios.

A mi tutor el Dr. Reyes Juárez Ramírez por saberme guiar no solo en este paso de maestría sino durante toda mi formación profesional en la licenciatura.

Al Consejo Nacional de Ciencia y Tecnología por otorgarme la beca necesaria para poder realizar mis estudios de una manera mucho mas enfocada gracias al apoyo otorgado.

A la Universidad Autónoma de Baja California por la infraestructura incondicional para realización de estudios, experimentos, consultas y otras labores invaluables para la realización de esta investigación

Al grupo de ayuda de Natural Language Tool Kit quien desinteresadamente ofrecen su ayuda a los nuevos integrantes del desarrollo con procesamiento de lenguaje natural.

Tabla de Contenidos

Lista de Figuras

Lista de Tablas

CAPÍTULO 1

INTRODUCCIÓN

1. Introducción

La Ingeniería de Requerimientos (IR) es la base de los proyectos de desarrollo de software, pues en la gran mayoría de los casos es el primer paso para el desarrollo de un proyecto y por tanto las demás etapas heredan la calidad o los defectos posiblemente generados desde un principio. La idea básica de la ingeniería de requerimientos es definir los objetivos, funciones y limitantes de un sistema (Laplante, 2007).

Existen numerosos problemas que aún están en proceso de ser resueltos por la IR; el problema más grande según algunos autores es la ambigüedad (Berry, 2003) (Nikora, 2011) y la raíz de este problema la encontramos en el hecho de que actualmente el uso del Lenguaje Natural (LN) es la forma más común para expresar requerimientos de software (Berry, 2005).

Aún cuando todos los problemas del LN se heredan a los requerimientos, el uso de lenguajes formales no ha logrado un éxito en la comunicación cliente – desarrollador debido a que no todas las personas dominan estos lenguajes y por lo tanto al utilizarlos se limita la cantidad de usuarios que pueden utilizarlos lo cual representa un problema aún más grande que el que se intentaba solucionar (Nikora, 2011)

Los requerimientos de software sufren de muchos más problemas que solo la ambigüedad, en esta tesis no se tiene pensado atacarlos todos sino los tres que se han considerado más significativos como son la *ambigüedad*, *conjunción e incompletitud*.

La ambigüedad representa un interés especial de solución, debido a que como se mencionó antes, es de los problemas más comunes y también de los más peligrosos, ya que pueden ocasionar que se continúe el proceso de diseño y desarrollo con una idea totalmente incorrecta que puede ocasionar problemas graves, mismos que en etapas tempranas del proceso son mucho más sencillas y económicas de corregir (McConnell, 2004).

La conjunción es otro problema que afecta muchísimo sobre todo a los ingenieros de calidad de software en donde un requerimiento conjugado puede generar casos de prueba parcialmente verificables, mismos que deben ser evitados a toda costa (Davis, 1993).

La incompletitud es de los factores más complicados de verificar, ya que previamente se tiene que hacer toda una labor de investigación para definir lo que completitud significa en nuestro contexto; detalles de cómo fue esto realizado se explican más adelante en esta tesis, pero en un resumen general la idea principal es poder tener una gramática que represente los elementos mínimos con los que debe contar un requerimiento funcional y entonces poder hacer una comparación del enunciado contra la gramática para determinar si este está o no completo.

De la misma forma como lo describe Nikora (Nikora, 2011), el problema no es la revisión del requerimiento, el verdadero problema es como atacar dos principales factores:

1) El error humano, no importa que tan bueno sea el desempeño de un trabajador, siempre existe la posibilidad de error
2) Consumo de recursos, cuando se tienen proyectos grandes de miles de requerimientos, es prácticamente imposible revisarlos a detalle.

Para apoyar en resolver estos problemas se plantea utilizar técnicas de Procesamiento de Lenguaje Natural (PLN) para una evaluación formal y automática de requerimientos de software, esto apoyándose de ambientes de desarrollo como el Natural Language ToolKit (NLTK) para el lenguaje de programación Python.

1.1 Problema de investigación

El más grande reto en esta investigación no es el cómo detectar si un requerimiento sufre de un problema o no, ya que la evaluación humana hace un excelente trabajo en esta actividad, el problema más grande es que el trabajo y esfuerzos necesarios para la evaluación de un gran número de requerimientos se vuelve una tarea no apta para la evaluación humana y por esta razón organizaciones con proyectos grandes de desarrollo como NASA están apostando por el PLN como una ayuda a reducir esta carga (Nikora, 2011).

De acuerdo a Brooks (Brooks, 1987), tan pronto como se descubra un problema más barato será su solución, es por esto la importancia de atender los requerimientos pues es una de las etapas más tempranas. Errores en los requerimientos ocasionan que se propague el defecto a las demás etapas del ciclo de desarrollo causando con esto un enorme gasto de recursos. Varios de estos problemas son causados indirectamente por el uso del lenguaje natural, aun así es el más utilizado a la fecha (Nikora, 2011)

El lenguaje natural tiene numerosas ventajas sobre lenguajes formales como se describe posteriormente en esta tesis, sin embargo, al momento de intentar realizar una evaluación automática son precisamente estas ventajas las que causan los mayores retos. El lenguaje natural es tan diverso que ocasiona que un idéntico texto pueda ser interpretado por múltiples lectores, aquí es donde se plantea poder definir una gramática que siga permitiendo un uso extenso del vocabulario natural mientras se pueda mantener la objetividad del texto.

El objetivo no es solo poder hacer una evaluación sino hacerlo en una manera eficiente que permita la optimizan de recursos y no entorpezcan el flujo ágil del desarrollo de software de la actualidad.

1.2 Contribución Principal

En esta tesis se describe el proceso de diseño de una estructura formal de requerimientos funcionales de software así como también el diseño, programación y evaluación de una herramienta titulada NLARE.

Para el desarrollo de la estructura formal de requerimientos se han tomado investigaciones previas (Reyes, 2011) del grupo de investigación “Grupo de Investigación e Innovación en Procesos de Ingeniería de Software” (GIIPIS) donde se realiza esta tesis. Algunas versiones de estructura de requerimiento, propuestas en estos trabajos previos, han sido tomadas y adaptadas a un formato más computable para el procesamiento de lenguaje natural.

Más adelante en esta tesis se abordaran a detalle las tres principales problemáticas a tratar las cuales son: ambigüedad, atomicidad y completitud. Es de suma importancia que una especificación de requerimientos esté tan libre de defectos como sea posible, ya que como se indica más adelante, el costo de resolver problemáticas crece exponencialmente entre más adelante en el ciclo de desarrollo se detecten los problemas.

Para atacar el problema de completitud se ha desarrollado una expresión regular que atiende a la gramática previamente definida, misma que es evaluada y procesada por técnicas de PLN que son soportadas por la librería Natural Language Toolkit (NLTK) iniciada desde 2001.

Para atacar los problemas de ambigüedad y verificar la correcta atomicidad se han definido reglas gramaticales igualmente evaluadas automáticamente por la aplicación desarrollada para una evaluación imparcial y a una mucho mayor velocidad que la que pudiera proporcionar la evaluación manual humana.

Para probar la validez de los resultados se han corrido experimentos en el ámbito académico e industrial. Los experimentos en cuestión se describen posteriormente en esta tesis. En un sentido resumido, la metodología empleada es de carácter investigación-acción (Lewin, 1973), en la que se realiza una etapa regular de requerimientos para observar el estado en el que los mismos seguirían su flujo sin la utilización de un análisis automático. Posteriormente, en la experimentación, se realiza una evaluación automática de los requerimientos haciendo uso de la gramática construida para identificar cuáles de los potenciales futuros problemas pudieron haber sido evitados con una buena redacción de requerimientos.

1.3 Hipótesis

En base a mi experiencia profesional como ingeniero de calidad de software, en la que estuve familiarizado con todo el proceso de desarrollo, desde el inicio de los requerimientos hasta la corrección de errores, me di cuenta de numerosas problemáticas que surgían desde los requerimientos.

El problema inicia con un requerimiento mal redactado, principalmente de carácter ambiguo que al ser programado es interpretado de una forma, y al ser probado (mi actividad laboral) era probado de cierta manera, al no cumplir con las especificaciones se realizaba un documento explicando toda la situación. En un considerable número de ocasiones la solución a dicho problema reportado se rastreaba a que el requerimiento estaba incorrecto, una sencilla modificación al mismo y todo se volvía más claro lo que me ayudo a la elaboración de la siguiente hipótesis que se fundamenta en investigaciones previas a esta tesis sobre las capacidades actuales del PNL y su aplicación en el área de requerimientos.

Hipótesis #1:

Haciendo uso de expresiones regulares es posible construir una gramática capaz que evaluar una sentencia de un requerimiento funcional, siempre y cuando ésta obedezca a un conjunto de patrones propios de un requerimiento como son: actor, función y detalle de función.

1.4 Objetivos y metas

Los objetivos planteados para este trabajo de tesis están directamente relacionados con el planteamiento del problema descrito en la sección anterior (1.1), ya que se pretende poder generar una versión formal de cómo contruir un requerimiento funcional asi como también una manera de poder realizar la revisión del mismo de manera formal y automatizada.

Estos objetivos se clasifican en dos niveles: Objetivos generales y objetivos específicos. Tambien en la siguiente sección se describen las metas para cada objetivo específicos, asi como la hipótesis que será guía para el desarrollo de la investigación.

1.4.1 Objetivos generales

- Generar una estructura de composición de los requerimientos funcionales, expresada de manera formal.
- Generar un mecanismo para la revisión automatizada de requerimientos funcionales.

1.4.2 Objetivos específicos

OE1: Integrar un patrón de elementos estructurales para un requerimiento funcional, de tal manera que este pueda ser evaluado formalmente.

OE2: Definir una gramática mediante expresiones regulares que permita evaluar la estructura sintáctica y semática de un requerimientos funcional.

1.4.3 Metas

Las metas por alcanzar en esta investigación son las siguientes:

M1: Determinar los elementos mas básicos de un requerimiento funcional buscando los que sean la clave para el entendimiento del mismo.

M2: Generar una gramática que cubra los patrones encontrados en los requerimientos basándose en los elementos básicos de la M1.

M3: Desarrollar un conjunto de expresiones regulares capaz de atender a la gramática definida en la M2.

M4: Utilizar técnicas de procesamiento de lenguaje natural para proveer una via de evaluación para las expresiones definidas en la M3.

M5: Construir una herramienta que pueda utilizar las técnicas evaluadas enn la M4 para realizar una evaluación rápida y automática de los requerimientos funcionales.

Para cumplir con este bloque de objetivos y metas se propone la siguiente metodología de trabajo.

1.5 Metodología de trabajo

Para el desarrollo de la presente investigación se uso la siguiente metodología.

1. La primera fase del desarrollo de la investigación fue al estudio del estado del arte sobre la situación de la etapa de requerimientos en el desarrollo de software, así como también las metodologías y técnicas empleadas en la misma. También se analizaron herramientas actuales para ayudar a los ingenieros de requerimientos a realizar mejor y más eficientemente su trabajo.

2. Formulación de la hipótesis a desarrollar en la investigación, con el propósito de validar los objetivos y metas para esta investigación.

3. Desarrollo de una gramática basada en expresiones regulares capaz de atender a una estructura previamente definida de elementos que deben estar presentes en un requerimiento funcional. Para determinar si cada requerimiento cumple o no con la gramática se desarrollo una aplicación utilizando PLN.

4. Realización de casos de estudio, usando la metodología investigación-acción (Lewin, 1973):
    a. Para la elección de los casos de estudio se optó por utilizar proyectos de distintos rubros como Industria y Educación para poder hacer pruebas que no estén cargados solo a un área.
    b. Para cada caso de estudio se realizaron las siguientes actividades:
        i. Se realizó una recopilación de los requerimientos del proyecto
        ii. Si estos estaban en otro idioma que no fuese inglés se traducen manualmente
        iii. Se evalúan manualmente los requerimientos para determinar sus problemas
        iv. Se utiliza la herramienta NLARE para determinar que tan cerca esta de una evaluación humana-manual.

CAPÍTULO 2

FUNDAMENTOS TEÓRICOS

2. Fundamentos teóricos

En este capítulo se hablará sobre la teoría que rodea a esta tesis para facilitar el entendimiento de futuros capítulos; los fundamentos teóricos vistos en este capítulo serán clave en los próximos capítulos.

Primeramente se tocará el tema de la Ingeniería de Requerimientos (IR), cómo esta ingeniería ha ido cambiando y la importancia de la misma en el desarrollo de software. Las problemáticas de esta ingeniería es una sección de suma importancia para comprender la razón fundamental de esta investigación.

Como se verá más adelante en este capítulo, el uso del lenguaje natural es el método más común de redactar requerimientos de software así que se abordaran temas de su relación con la ingeniería de software así como también las dificultades que éste presenta para el procesamiento computación del mismo.

Finalmente se tocarán temas relacionados con la teoría del lenguaje, como está éste construido, cómo se entiende por el lector, así como técnicas actuales para el procesamiento del mismo.

2.1 Ingeniería de Requerimientos

El desarrollo de software puede ser iniciado en distintas etapas, puede ser un trabajo nuevo o bien la continuación de uno, en cualquier caso es necesaria una idea de lo que se va a realizar. Estas ideas son plasmadas en un documento llamado *Especificación de Requerimientos* mismo que ayuda a continuar las siguientes etapas del desarrollo de software, como el diseño y la misma programación.

En la ingeniería de requerimientos es de vital importancia que los requerimientos de software estén lo mas correctos posibles, libres de problemas como ambigüedad, conjunción e incompletitud, mismos que si se dejan pasar pueden crear problemas que son más costosos de reparar que haber hecho una correcta evaluación de requerimientos y correcciones donde estas fueran necesarias (Lauesen, 2000).

Podemos decir entonces que la ingeniería de requerimientos consiste en describir el sistema que será construido a partir de las especificaciones de requerimientos. Estos requerimientos hablan de cuestiones variadas como rendimiento, confiabilidad, usabilidad, mantenibilidad y por supuesto la funcionalidad del sistema (Grandy, 1992).

Dentro de la ingeniería de requerimientos también existe la parte de búsqueda de información, en donde el analista de requerimientos es responsable de extraer la información del cliente empleado distintos métodos y técnicas para lograr comprender la idea del cliente y convertir las ideas en un producto real.

Inicialmente se previa que la etapa de requisitos era solo al inicio de un proyecto y era una actividad que una vez concluida se abandonaba; esto ha ido cambiando con diferentes modelos de desarrollo al grado que ahora se entiende que esta etapa de requerimientos no debe abandonarse y puede estar sujeta a algunos cambios conforme avance el proyecto, aunque cabe destacar que en la medida que mejor se hagan los requerimientos desde un inicio, menor será el impacto negativo y necesidad de ajustes conforme avance el ciclo de desarrollo, para mostrar gráficamente este flujo se muestra la imagen a continuación.

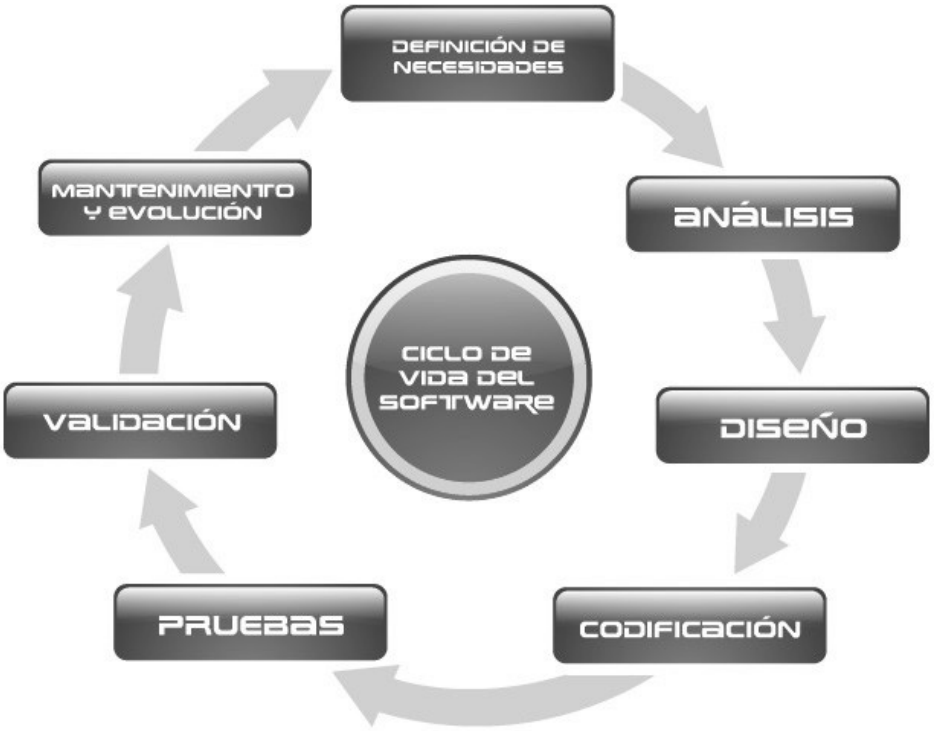

Figura 2.1. Ciclo de vida del software

En la Figura 2.1 podemos observar la etapa definida como “Definición de Necesidades” que es todo un proceso y el tema central de esta investigación la cual también está formada por un ciclo interno el cual se muestra en la Figura 2.1.

Figura 2.2. Ciclo de vida de Requerimientos

En la Figura 2.2 podemos observar de una manera general como ocurre el ciclo de vida de los requerimientos que a su vez es un proceso que se encuentra ciclado dentro de un ciclo más grande que es el del desarrollo de software mostrado en la Figura 2.1, a continuación se explicará de manera general las etapas en este ciclo de vida.

Etapa 1. Elicitación

Esta es la primera etapa para empezar el ciclo de vida. En esta primera actividad lo que se realiza principalmente es la comunicación con el cliente, con el objetivo de poder extraer esas necesidad que posteriormente serán los requerimientos del futuro software a construir, para esto se utilizan diferentes técnicas que no necesariamente una es mejor que otra, pero entre las más utilizadas podemos ver la *entrevista* en donde los futuros usuarios del software son cuestionados por el ingeniero de requerimientos en busca de información clave para los requerimientos. Esta actividad suele

combinarse también con la técnica de observación en donde sin intervenir con la producción, el ingeniero de requerimientos asiste al área de trabajo a identificar como se realiza actualmente las tareas para poder detectar esas oportunidades a mejorar o incluir en el software a desarrollar (Zowghi, 2007)

Entre las técnicas más comunes para recabar los requerimientos tenemos las siguientes (Wiegers, 2003):

Entrevistas

Las entrevistas son un método común. Por lo general no se entrevista a toda la gente que se relacionará con el sistema, sino a una selección de personas que represente a todos los sectores críticos de la organización, con el énfasis puesto en los sectores más afectados o que harán un uso más frecuente del nuevo sistema.

Las entrevistas son una buena herramienta ya que si se realizan con buena planeación, se puede extraer información importante en forma de enunciados que representan las respuestas del cliente. Estas respuestas posteriormente pueden ser utilizadas para construir los requerimientos a evaluar.

Talleres

Los requisitos tienen a menudo implicaciones cruzadas desconocidas para las personas implicadas individuales y que a menudo no se descubren en las entrevistas o quedan incompletamente definidas durante la misma. Estas implicaciones cruzadas pueden descubrirse realizando en un ambiente

controlado, talleres facilitados por un analista del negocio, en donde las personas implicadas participan en discusiones para descubrir requisitos, analizan sus detalles y las implicaciones cruzadas.

En los talleres, el resultado puede ser un poco menos tangible en cuestión de la información necesaria para construir los requerimientos. Si se utiliza esta opción en conjunto con la entrevista se podrían obtener datos que no se habían logrado recabar.

Contrato

En lugar de una entrevista, se pueden llenar formularios o contratos indicando los requisitos. Este podría ser el método que más se asemeja a los requerimientos necesarios para continuar con el desarrollo. Para esta investigación este artefacto representa una gran importancia debido a que errores en este documento, tienen un impacto muy parecido al que los requerimientos tendrán en el desarrollo del software.

Prototipos

Un prototipo es una pequeña muestra, de funcionalidad limitada, de cómo sería el producto final una vez terminado. Ayudan a conocer la opinión de los usuarios y rectificar algunos aspectos antes de llegar al producto terminado.

Los prototipos pueden ser: diagramas, aplicaciones operativas con funcionalidades sintetizadas. Los diagramas, en los casos donde se espera que el software final tenga diseño gráfico, se realizan en una variedad de

documentos de diseño gráficos y a menudo elimina todo el color del diseño del software (es decir utilizar una gama de grises). Esto ayuda a prevenir la confusión sobre la apariencia final de la aplicación.

Para esta investigación, es la metodología que menos proporciona información textual, a no ser que se definiera un documento de requerimientos del prototipo como tal, esta técnica no representa mucha ayuda

Casos de uso

Un caso de uso es una técnica para documentar posibles requisitos, graficando la relación del sistema con los usuarios u otros sistemas.

Los casos de uso también pueden estar acompañados de información textual, en teoría se esperaría que los casos de uso fueran resultantes de un previo requerimiento y ayudaran a la consolidación del mismo. Realizando casos de uso para cada requerimiento se pueden encontrar detalles que no se hayan visto en una primera etapa y poder aspirar a una mejor calidad.

Etapa 2. Especificación y Documentación

Una vez que se realiza la recolección de las necesidades y detalles particulares del sistema, sigue el segundo reto y trata sobre cómo plasmar lo observado en palabras para poder generar un documento que debe ser tan perfecto como sea posible pues en términos legales éste representará un contrato entre el cliente y el desarrollador sobre qué es lo que se tiene que realizar. Entre los atributos de calidad que se desean lograr para un requerimiento, según Davis (1993), son:

- **Unitario:** que el requerimiento hable de solo una cosa.
- **Completo:** incluya toda la información necesaria para su comprensión.
- **Consistente:** que no contradiga lo definido en cualquier otro requerimiento.
- **Atómico:** no debe contener conjunciones.
- **Trazable:** debe cumplir con alguna necesidad para el usuario.
- **Actual:** no ha dejado de ser válido por el paso del tiempo.
- **Factible:** posible de realizar de acuerdo a los presupuestos y tecnología
- **Claro:** sin ambigüedades que lo hagan susceptible a interpretaciones
- **Relevante:** debe estar ligado estrictamente con el proyecto en cuestión
- **Verificable:** debe ser posible verificar si fue cubierto o no en la programación

Otras literaturas como el estándar IEEE 830-1998 (IEEE 830-1998) proporcionan lineamentos muy parecidos también para definir como debe estar compuesto un buen requerimiento.

Esta es la parte central de la tesis. En el caso de esta tesis solo se trabajaran 3 parámetros de calidad como lo son *Atomicidad*, *Ambigüedad* y *Completitud* que más adelante serán explicados.

Etapa 3. Validación y Verificación

Esta es una etapa en la que hay que asegurarse que cada requerimiento haya cumplido con los atributos de calidad definidos anteriormente. El problema no es tanto el hecho de revisarlos, sino la

cantidad de esfuerzo que esto requiere conforme el numero de requerimientos va aumentando, y unos de los principales problemas es que no solo depende de un involucrado sino de varios, por ejemplo, ambos, el cliente y la empresa deberán estar de acuerdo en que los requerimientos plasman correctamente las ideas y el entendimiento de ambos para llevar a cabo el proyecto exitosamente.

Existen normalmente tres diferentes técnicas de evaluación de requerimientos (Gilb and Graham, 1993; Ebenau and Strauss, 1994) las cuales se explican a continuación:

- **Prototipo**: la cual consiste básicamente en irse directo a la programación y asumir los requerimientos como correctos y evaluar entonces el prototipo en vez de los requerimientos.
- **Modelado**: se puede generar una versión de los requerimientos en un lenguaje formal, para en el proceso analizar posibles errores en los requerimientos y presentar algo formal al cliente, el detalle aquí y una gran limitante es que no todos los clientes dominarían un lenguaje como UML.
- **Casos de Uso**: se puede redactar un caso de uso que describa la funcionalidad del sistema desde un punto de vista del usuario, y con esto poder encontrar problemas o inconsistencias en los requerimientos.

Como parte de la propuesta de esta tesis se pretende agregar una opción más de validación que bien podría incluso combinarse con las actuales. En capítulos más adelante se hablará más en detalle de la propuesta de validación.

2.2 Problemáticas en Ingeniería de Requerimientos

La investigación en campos de ingeniería de requerimientos ha indo creando diferentes técnicas, modelos y herramientas para la incrementar la facilidad y mejorar el producto resultante que son los requerimientos. Ejemplos de estas técnicas son UML y OCL, sin embargo, hasta el momento el *lenguaje natural* es una parte esencial e irreemplazable de esta ingeniería (Nuseibeh, 2001), por lo que aún después de numerosos avances y buenas prácticas recolectadas a lo largo de los años, la Ingeniería de Requerimientos continúa siendo una de las partes más complicadas de todo el desarrollo de software.

Los problemas en los requerimientos desde mucho tiempo han sido estudiados, pues se ha demostrado que requerimientos que sufren de problemas como incompletud y ambigüedad tienen un crítico impacto en la calidad final del software (Bell, 1976), tanto es así que problemas en requerimientos han causado defectos multi-millonarios, mismos defectos que pudieron haber sido detectados y corregidos en etapas tempranas en donde el costo sería inmensamente menor (Brooks, 1987).

De acuerdo a una investigación realizada en el año 2000 (Lamsweerde, 2000a) a 350 empresas con más de 8000 proyectos evaluados se encontraron los siguientes resultados que se muestran en la Tabla 2.1.

Tabla 2.1. Evaluación de proyectos USA-2000

Resultado	Porcentaje
Proyecto Exitoso	16 %
Proyecto Fracaso	33 %
Proyecto Regular	51 %

De acuerdo a dicha investigación en el caso de los proyectos etiquetados como *Regulares* se refiere a proyectos que solo lograron funcionalidades parciales o bien sufrieron de exceso de re trabajo y por tanto retardos en entrega.

De acuerdo a los estándares buscados en esta investigación, agruparíamos los Proyectos Regulares junto con los de Fracaso pues consideramos que no existe el "éxito parcial" los que nos daría un enorme porcentaje de fallos de 84% lo que sin duda es alarmante y justifica una investigación y propuestas de mejora. Como soporte adicional a esta problemática también podemos citar las encuestas realizadas por el Instituto de Software Europeo (ESI, 1996) en el que se encuestaron 3800 diferentes organización de 17 países en donde se encontró que más del 50% de los encuestados mencionaban ya sea a los requerimientos o al manejo de los mismos como los principales problemas enfrentados.

De acuerdo a una investigación realizada por el grupo Standish se determinó que en Estados Unidos de América se gastan más de $250 billones de dólares cada año en el desarrollo de aproximadamente $175,000 proyectos, de entre los cuales se encontró que el 31.1% de los proyectos

serían cancelados antes de ser completados y un impactante 53.7% de ellos costarían 189% de lo que originalmente se estimó que costarían. Otro ejemplo de proyecto costoso es el del aeropuerto de Denver, en el que la falta de un buen software para el control de equipaje le costaba a la ciudad $1.1 millones diarios. Las problemáticas en parte fueron causadas por malos requerimientos.

Uno de los aspectos más importantes de esta investigación son las las opiniones de los líderes de proyectos que muestran los factores que ellos consideran claves en el éxito y culpables de los fracasos según sea el caso, dichas tablas se muestran en la Tabla 2.2 (Standish, 2005):

Tabla 2.2. Factores Clave en el Existo de Proyectos

Factor de Éxito	**Porcentaje**
Involucrar al usuario	15.9 %
Soporte de ejecutivos	13.9 %
Buenos requerimientos	**13.0 %**
Buena planeación	9.6 %
Expectativas realistas	8.2 %
Entregas cortas	7.7 %
Equipo competente	7.2 %
Compromiso	5.3 %

Ahora bien según las opiniones de los líderes de proyecto mostramos la lista resultante de las causas por las cuales los proyectos fracasan.

Tabla 2.3. Factores Causa en el Fracaso de Proyectos

Factor de Fracaso	Porcentaje
Falta de información del usuario	12.8 %
Requerimientos incompletos	**12.3 %**
Requerimientos cambiantes	**11.8 %**
Falta de soporte ejecutivo	7.5 %
Incompetencia tecnológica	7.0 %
Falta de recursos	6.4 %
Expectativas irreales	5.9 %
Objetivos confusos	5.3 %

En la Tabla 2.3 podemos identificar que los tres principales problemas en los fracasos de los proyectos están derivados de los requerimientos, así como también dos factores más para sumar un porcentaje de 48.1% de factor de fracaso únicamente de requerimientos, lo que representa un valor que indica la necesidad del estudio de este paso tan importante en el desarrollo de software, y la necesidad de tecnologías para ayudar en el desarrollo de esta complicada tarea que son los requerimientos.

Entonces entre los principales problemas destacamos (Standish, 2005):

Causa #1: Falta de Información del Usuario:

Este problema va más que nada relacionado a que en muchos proyectos se tiene una sesión inicial con el usuario y no se le vuelve a involucrar en el proyecto hasta que éste está ya prácticamente terminado, y ésta resultó ser la causa numero uno de fracaso (Standish, 2005). Es muy importante mantener al usuario siempre involucrado en todo el proceso para poder descartar futuros problemas de visiones diferentes y atenderlas en etapas tempranas

Causa #2: Requerimientos Incompletos

Esta podría ser uno de los errores más fáciles de cometer ya que es muy fácil omitir información, no por intensión sino debido a que incorrectamente se llega a asumir que el lector de los requerimientos mantiene ese mismo conocimiento que el escritor sobre ciertos puntos, lo que lleva a descripciones imprecisas de lo que se desea lograr con las funcionalidades del sistema en cuestión. Si se tienen requerimientos incompletos estos pueden causar que en futuras etapas se desarrollen funcionalidades incorrectas o bien con comportamientos distintos a los esperados por el usuario (Standish, 2005).

Este problema en particular es uno de los que en esta investigación se intentan atacar debido a que sin una estructura formal es complicado argumentar si un requerimiento está o no completo.

Causa #3 Requerimientos Cambiantes

Este es un problema que se origina principalmente por dos causas, la que mas impacta a esta tesis sería el hecho de una mala construcción de requerimientos, ya que si estos están mal hechos al llegar a otras etapas podrían no tener el sentido correcto y necesitar reformularse nuevamente (Standish, 2005). En esta investigación se pretende que los requerimientos estén correctos desde un inicio para que no haya necesidad de modificarse, sin embargo, existe un segundo factor de cambio, referente al hecho de cuestiones administrativas internas de la empresa cliente, en donde podría haber desacuerdos en lo que se quiere y por tanto estos requieran cambios en los requerimientos que previamente sometieron a desarrollo. En esta tesis no se pretende abordar este problema. En este caso habría que implementar mejores técnicas para extracción de requerimientos, mismas que tampoco son el enfoque de esta investigación.

Conclusión de las problemáticas

La ingeniería de requerimientos es una etapa del desarrollo en la que no se puede asumir información sobre el proyecto si no se está seguro, ya que pequeños detalles como esos son los que al final resultan causantes de que el producto resultante no se comporte como se esperaría por parte del cliente. Continuar un ciclo de desarrollo con requerimientos que no son los correctos es una actividad peligrosa y puede tener enormes consecuencias conforme el proyecto avanza (Lamsweerde and Letier, 1998).

2.3 El Lenguaje Natural en los Requerimientos

Actualmente una gran mayoría de los requerimientos se expresan en lenguaje natural aunque también pueden estar ayudados de otros métodos como por ejemplo diagramas de flujo UML.

El *Lenguaje Natural* al igual que cualquier otro medio de comunicación tiene sus ventajas y desventajas. En el caso particular del lenguaje natural nos encontramos con el hecho de que por su naturaleza misma es ambiguo y por lo tanto, los requerimientos expresados por este medio son susceptibles a heredar tal problema también.

El problema de la ambigüedad en requerimientos no es un tema nuevo pues ha sido intentado de resolver desde los años 80s por investigadores como Gause and Weinberg (1989) quienes realizaron investigaciones para poder detectar y eliminar la ambigüedad en requerimientos. Entre estas investigaciones se descubre el hecho de que para un lector, un requerimiento puede no ser ambiguo sobre todo si este lector es el mismo que escribió el requerimiento. Este fenómeno se da porque aunque el requerimiento tenga problemas, el lector resuelve esta ambigüedad y le da el significado correcto al mismo, sin embargo, esta resolución de ambigüedad podría no tener el mismo efecto cuando una persona distinta es la que tiene que resolverla y entonces pudiera darle otro significado que no era el esperado por el escritor.

De momento podría parecer que el lenguaje natural no es una buena idea para la redacción de requerimientos, sin embargo, como se comentó antes es lo más utilizado; la pregunta obvia surgiría de la razón de ser el más

usado con tantos problemas. El detalle es que aunque los lenguajes formales evitan la ambigüedad por su sintaxis, no son los suficientemente descriptivos ni dominados por el público en general, motivo que podría agregar incluso más problemas que los que está intentando solucionar, mientras que el lenguaje natural es tan vasto que es más fácil describir una idea y encontrar quien pueda entenderla por el amplio dominio que se suele tener del lenguaje natural comparado con los lenguajes formales (Sawyer and Kotonya, 2001).

Otro detalle importante a considerar a favor del lenguaje natural es que el desarrollo de requerimientos basados en un lenguaje formal no es una solución pues en algún momento, alguna idea o incluso apuntes informales estarán en lenguaje natural antes de pasarse a un lenguaje formal, y un incorrecto entendimiento de este lenguaje natural podría resultar en un incorrecto desarrollo del requerimiento en lenguaje formal, lo que nos llevaría exactamente al mismo problema inicial.

Investigaciones (Sommerville, 2001) han mostrado que la ambigüedad es una de las propiedades más susceptibles a problemáticas del lenguaje natural y causante de numerosas problemáticas.

2.4 Problemática de ambigüedad

Una propiedad inherente del lenguaje natural es precisamente la ambigüedad; en su concepto más simple es que ocurre cuando una palabra puede tener más de una posible interpretación lo que provoca que una oración entonces tenga múltiples sentidos (Tinholt & Nijholt, 2007).

Cuando leemos una oración, principalmente nos basamos en el contexto para intentar darle el significado correcto a una palabra, sin embargo, no en todas las ocasiones es tan fácil determinar el significado esperado por el escritor desde la primera leída, esto para complicar un poco mas este problema de ambigüedad. Existen distintos tipos de ambigüedad que afectan al lenguaje natural (Grenat & Taher, 2008), entre estos tipos tenemos el *léxico*, *referencial*, *alcance* y *estructural*. A continuación se describirán brevemente cada uno de ellos.

Ambigüedad léxica

Esta ambigüedad se origina en una sola palabra (Quiroga-Clare, 2003), a este tipo de palabras se les conoce como *polisemias* que viene del griego y significa “muchos significados”, pero entre ellos guardan alguna relación; al contrario con los homónimos que aunque la palabra se escribe de la misma forma esta puede tener significados completamente distintos.

Ambigüedad referencial

Este tipo de ambigüedad es causada cuando una palabra no define explícitamente a que persona u objeto hace referencia (Grenat & Taher, 2008). Un ejemplo común de esto son palabras como “este” o “aquel”, a este tipo de palabras se le conoce como anáforas, dentro de este tipo de ambigüedades también se encuentran en las que la información no es suficientemente para determinar la referencia, por ejemplo “Carlos gano un premio”, pero Cual ¿Carlos?, ¿Carlos Huertas?, ¿Carlos Villegas?, si el elector conoce a más de un Carlos, seguramente puede haber problemas para resolver esa ambigüedad.

Ambigüedad de Alcance

De acuerdo a Grenat y Taher (2008), esta ambigüedad se genera cuando otros componentes en su contexto estructural determinan el significado de un constituyente dentro de una oración, tal vez si no se está muy familiarizado con la lingüística, será complejo entender la definición. Para esto entonces proporcionamos el siguiente ejemplo: “En la fiesta había hombres y mujeres respetables”. En este sencillo ejemplo se puede ver con claridad la ambigüedad referencial pues “respetables” tiene un alcance hasta “mujeres” es decir ¿que no había hombres respetables? O el alcance es entonces hasta “hombres” y entonces ambos eran respetables.

Ambigüedad Estructural

Esta ambigüedad es una de las más complicadas, pues se puede dar incluso aunque no exista una sola palabra ambigua en la oración, es decir, la oración como tal puede ser interpretada por distintas formas; un ejemplo podría ser “Carlos vio a Alberto enojado”, si bien las palabras no son ambiguas por si solas, no es posible determinar si Carlos estaba enojado cuando vio a Alberto, o bien si Alberto estaba enojado y fue visto por Carlos.

En otro tipo de literaturas (Levinson, 1983) podemos encontrar otra clasificación más de ambigüedad que explicamos a continuación.

Ambigüedad pragmática

Este tipo de ambigüedad trata sobre la relación que existe entre el significado propio de la oración y el contexto en el que esta ocurre, la diferencia principal entre la pragmática y la semántica, es que la primera depende del contexto y la segunda es independiente.

Sin importar el tipo de ambigüedad o dónde se genera en la comunicación ésta es compensada de diferentes formas según el involucrado.

Por el emisor: que por medio del contexto del escrito puede disminuir la potencial ambigüedad existente en su escrito.

Por el receptor: este puede asumir los correctos significados de los elementos potencialmente ambiguos basado en el contexto así como también en previas experiencias con el emisor.

El gran problema sin embargo se presenta cuando tenemos una comunicación formal por escrito en donde asumir que se seleccionó el significado correcto a una expresión podría causar numerosos problemas y por tanto, la práctica de asumir significados es muy peligrosa y poco recomendada en el área de especificación de requerimientos (Brooks, 1987).

Otro potencial problema interesante con la ambigüedad es el hecho de que aún cuando un texto sea claramente ambiguo tras una evaluación formal, como humanos podemos pasar por alto esta ambigüedad al resolverla inconscientemente, situación que de igual forma es muy peligrosa sobre todo en una evaluación manual de requerimientos, en donde incluso con la intención de detectar ambigüedades, estas sean pasadas por alto al resolver la ambigüedad inconscientemente.

2.5 Teoría del Lenguaje

A lo largo de la historia se han desarrollo métodos automáticos para la detección de la ambigüedad, sin embargo es prudente iniciar esta sección hablando un poco de la teoría que envuelve a estos métodos.

Un lenguaje está definido por un conjunto de sentencias en donde cada sentencia tiene una longitud finita (Chomsky, 2002), de esta misma manera, las sentencias están formadas de palabras las cuales también tienen una longitud finita.

Una vez que hemos aprendido a leer este es un proceso bastante simple para nosotros, sin embargo investigaciones (Buzan, 2008) han demostrado que en efecto es un proceso complejo que involucra distintos procesamientos en el cerebro como se describen a continuación:

Reconocimiento

En este primer paso nuestro cerebro debe reconocer un símbolo como un carácter y así entonces un conjunto finito de símbolos como una palabra y así continúa. Siendo este el primer paso en el conjunto que comprende la lectura, este procesamiento cerebral ocurre incluso antes el efecto físico de percepción que se explica a continuación

Asimilación

Este es el proceso físico en el que el ojo humano capta la imagen de las palabras y entonces son transmitidas al cerebro.

Comprensión

Este el proceso de ligar todas las palabras, o bien todas las sentencias del texto que se lee para lograr crear un contexto del texto y comprender la idea, esta etapa también es conocida como intra-integración.

Extra-integración

Este cuarto paso incluye etapas de análisis, selección y reflexión en donde el lector debe conectar la información previamente leído e integrarla con su conocimiento anterior

Almacenamiento

El lector interpreta la información y almacena en su cerebro cierta parte de lo leído

Memoria

Aquí se determina la capacidad del lector para poder extraer información del cerebro de la información previamente leída

Comunicación

Este séptimo y último paso se refiere a la capacidad del lector para poder retransmitir la información previamente leída.

Otra parte de la teoría desarrollada previa a poder generar una evaluación automática hace referencia al lenguaje como si mismo, por ejemplo la estructura de una palabra, o más formalmente conocidas como “elemento léxico” (Stubbs, 2001), que se puede definir como la unidad mínima con un significado.

De acuerdo con las investigaciones de Field (2003) quien se encargó de estudiar qué necesita un humano para reconocer y entender un elemento léxico; encontró que para lograr entender un elemento léxico es necesario primero entender los significados y funciones internas del elemento mismo, las cuales de acuerdo a Levelt (Levelt, 1989) son definidas por 4 elementos de información que son fonología, morfología, sintaxis y significado, que a su vez quedan agrupados en dos grupos los cuales son elementos de forma y elementos de función los cuales se explican a continuación.

Forma

El primer elemento es la fonología es decir, cuando la palabra es hablada, como es pronunciada, en otros términos, como es el sonido producido por dicha palabra. De acuerdo a Levelt (Levelt, 1989) esta información fonológica en las palabras escritas está definida por su representación ortográfica. Por ejemplo la palabra computadora, puede ser representada como **computadora**, *computadora*, computadora o también entre muchos otros tantos ejemplos como ~~computadora~~ en donde el reconocimiento de esta entidad léxica puede complicarse un poco por su representación sin embargo su significado es el mismo.

Ahora, para poder utilizar un elemento léxico en el lenguaje natural es necesario que almacenemos la información morfológica del mismo, de tal forma que necesario conocer como modificar dicho elemento, por ejemplo, conjugar un verbo en distintos tiempos o bien modificar géneros o números como plural y singular.

Función

Además de la información de la forma léxica del elemento, el lector almacena información de la función de dicho elemento, los dos principales componentes de este grupo consisten en la sintaxis y el rango de significados del mismo, el conocimiento de sintaxis es el que dicta como construir una frase con los elementos léxicos, como cuando un elemento es verbo o sustantivo, etc. y cómo estos elementos se relacionan en la gramática.

Ahora bien, el otro elemento restante, es el *significado* mismo del elemento. Es importante notar que el significado de algunos elementos puede variar según el contexto en que se presenten, incluso si la variación no es muy grande, ya es suficiente para generar un cierto grado de ambigüedad, de tal forma que cuando accedamos al significado de un elemento léxico, realmente accedamos al rango completo de significados que este posee y seleccionamos el que creemos más adecuado.

Otro aspecto importante se deriva de estudios realizados por Westhoff (2002) y se conoce como los niveles de conocimiento, en donde se descubrió que los humanos podemos leer más rápido una secuencia de caracteres en ordenes conocidos como palabras que la misma secuencia de caracteres en un orden aleatorio, por ejemplo, nos es más fácil leer la palabra “simulador” que “oamlrsridu” esto prueba que leer no solo es una combinación de letras que forman palabras sino que utilizamos 5 distintos niveles de conocimiento al leer para poder hacerlo de una forma más rápida y agregarle el significado a las palabras.

Nivel 1

El primer nivel de conocimiento es el que define a probabilidad que tiene un carácter para ser combinado con otro, por ejemplo es más fácil encontrar la combinación “ca” que “wz”, al menos en el idioma español.

Nivel 2

En este nivel también se definen probabilidades, pero en este caso hacen referencia a la probabilidad de combinación de palabras dentro de una oración. Dada la experiencia que vamos adquiriendo al leer notamos que por ejemplo un artículo es comúnmente seguido por un sustantivo, este tipo de conocimiento nos permite leer frases bien ordenadas más fácilmente, por ejemplo “ A mí me gusta correr en autos” se lee más fácil y rápido que “Me autos mi correr en a gusta”.

Nivel 3

En el tercer nivel se involucra el conocimiento que tenemos para predecir la probabilidad de una secuencia de palabras en una frase, lo que nos ayuda a leer con mayor velocidad, este fenómeno se puede apreciar en expresiones conocidas como “estamos tirando la casa por la _____” en donde la mayoría completaría con la palabra “ventana”. Es importante hace notar que este conocimiento se une con el de Nivel 2 pues es muy poco improbable que la frase se completara con un verbo como “correr”; esto también ayuda a resolver ambigüedades.

Nivel 4

Este nivel involucra un aspecto un poco más avanzado del lenguaje, y hace referencia al conocimiento de estructuras lógicas de lenguaje, por ejemplo, al utilizar la palabra “también” hacemos referencia a que el menos

hay 2 entidades léxicas con alguna similitud de algún tipo, lo que nos ayuda a entender más fácilmente una frase.

Nivel 5

El ultimo nivel de conocimiento es el más extenso y complejo, y es muy particular de las experiencias personales del lector y su conocimiento del mundo, por ejemplo es de conocimiento general que los perros no pueden volar. Así como esos conocimientos tenemos muchos otros que nos ayudan a comprender mejor un texto y eliminar potenciales ambigüedades gracias a nuestro conocimiento personal.

Como hemos visto con los niveles de conocimiento, muy poco del total de información que nos ayuda a determinar el significado de una palabra u oración se provee en como la palabra se escribió, lo cual nos lleva a la conclusión que es nuestro conocimiento el que determina ese significado y he aquí la raíz de la ambigüedad, pues el escritor y el lector pueden tener un conocimiento distinto y por tanto aplicar diferentes significados a un texto que se escribe de una sola forma.

2.6 Teoría de la detección automática de ambigüedad

Antes de poder revisar algunos métodos para resolver ambigüedades automáticamente iniciaremos con revisar cómo la resolvemos los humanos. El investigador David Swinney (1979) trabajó en descubrir el proceso de cómo los humanos resolvemos la ambigüedad y en el caso particular de su investigación. El tipo de ambigüedad que fue objeto de estudio fue el de la ambigüedad léxica.

La idea principal del estudio era determinar si cuando leemos una palabra potencialmente ambigua accedemos a todos los significados conocidos de la misma o solo a algunos. Los resultados de dicha investigación revelaron que incluso cuando el contexto que rodea a una palabra indica un significado obvio, nuestro cerebro revisa todos los posibles significados antes de poder asignar el correcto. Un ejemplo claro de esto se puede ver en el ejemplo “Compré un nuevo ratón para mi computadora”, aquí aunque es obvio que “ratón” hace referencia al dispositivo, nuestro cerebro evaluó la posibilidad del animal que se llama ratón como potencial significado.

Leer, reconocer y resolver ambigüedades son trabajos realizados por el cerebro que prácticamente son inconscientes para el lector pues se realizan en un tiempo muy corto. El proceso para reconocer una ambigüedad es parecido al proceso de asignar un significado a un texto, de tal forma que al leer una palabra ambigua el cerebro asocia esa palabra a múltiples significados, lo que implica que entonces podemos saber cuándo una palabra es ambigua y una vez reconocida como ambigua en cuestión de milisegundos nuestro cerebro intenta buscar el significado correcto y la

mayoría de veces lo consigue.

Sin embargo, también existe situaciones externas al lenguaje que pueden afectar estas capacidades de detección de ambigüedad, por ejemplo si para un elemento léxico únicamente conocemos un significado, para nosotros no habrá ambigüedad, sin embargo, dicho elemento si puede contener distintos significados y que el que desconocemos sea el que el escritor quiso utilizar.

Otra situación posible para no detectar correctamente la ambigüedad puede ser que el contexto no es lo suficientemente claro o explicito, de tal forma que aunque sepamos los significados del elemento, no estemos seguros de cuál significado es el que debería ir en ese texto en particular, por ejemplo, aunque es poco probable, existe la posibilidad de que un texto que habla sobre el animal ratón, en algún punto del texto hablará del ratón como dispositivo de una computadora.

Existen en el área de procesamiento de lenguaje natural tres principales métodos de detección de ambigüedad: *sistemas basados en conocimientos*, *basados en estadísticas* y *los parsers*. Cada uno de estos diferentes sistemas se explicará a continuación.

Sistemas basados en conocimiento

Un sistema basado en conocimientos necesita ser previamente entrenado con conocimiento, más específicamente es conocimiento léxico el que se le debe transmitir a este sistema. Generalmente este conocimiento está derivado de una especie de diccionario, el cual se compone de palabras con su respectiva información léxica. Un ejemplo de este tipo de diccionarios

es *WordNet*, que es del tipo léxico-semántico y se compone por relaciones, definiciones y sinónimos, lo cual es un recurso muy valioso si te tiene un sistema que está buscando ambigüedades y puede acceder a todo este conjunto de léxicos (Klapaftis, 2005).

Sistemas basados en estadísticas

A diferencia de los sistemas basados en conocimiento, este tipo de sistemas no necesitan de un conjunto de conocimiento como bases de datos en los cuales buscar información para trabajar, por el contrario, estos son entrenados con información. Este tipo de información está construida con elementos léxicos previamente etiquetados con un resultado y sacando relaciones que le sirven para encontrar ambigüedades, incluso en contenidos que nunca se había entrenado previamente. La forma como estas estadísticas son construidas generalmente es con el uso de *parsers* que analizan información y alimentan al sistema con estructuras para poder construir nuevos textos, de tal forma que tanto mas grande sean los textos de entrenamiento, más preciso puede llegar a ser el sistema. Las máquinas de aprendizaje utilizan este tipo de enfoques y resultan ser eficientes (Song, 2004).

Parsers

Otra opción común utilizada es el uso de un *Parser*, que viene del inglés “parse” y significa analizar gramaticalmente a un léxico. Este tipo de herramientas generan una estructura gramatical a partir de la cadena de entrada. En el procesamiento de lenguaje natural, la salida producida por estas herramientas es muy valiosa ya que ésta refleja información muy importante sobre cómo está construida la oración, y esta información es incluso utilizada en otras aplicaciones como son traducciones o

reconocimiento de voz.

Una de las ventajas principales de los *parsers* es que han sido desarrollados desde hace mucho tiempo y están de cierta manera más pulidos que otras técnicas, por lo mismo tienen un gran soporte para ser implementados en muchos sistemas y son muy útiles para distintas actividades de procesamiento de lenguaje natural. En el caso particular de esta tesis ayuda en tareas como detección de ambigüedad e identificación de patrones en los requerimientos como se explicará en capítulos futuros.

Distintas metodologías proporcionan ventajas y desventajas, en capítulos posteriores hablaremos de la metodología seleccionada en esta tesis y un enfoque más detallado por qué seleccionamos esta metodología.

CAPÍTULO 3

TRABAJOS RELACIONADOS

3. Trabajos Relacionados

Como se ha revisado en capítulos anteriores, la mejor opción de representar los requerimientos es aún el lenguaje natural, esto a pesar de sus problemas también previamente mencionados; sin embargo, existe esa incertidumbre de no saber si el requerimiento está correcto, si la idea en la mente del escritor es realmente la que se está logrando transmitir al lector, y el lenguaje formal aunque asegura que el resultado del lenguaje tiene una sola interpretación. Esto no asegura que la descripción previa a este lenguaje formal fue comprendida correctamente lo que de igual manera agrega incertidumbre, y por tanto tampoco se puede decir que es una técnica a prueba de fallos.

El lenguaje natural sin embargo ha sido utilizado desde mucho tiempo como la mejor forma de comunicación, por lo que numerosas investigaciones se han realizado en busca de avanzar en corregir o minimizar los inherentes problemas de dicha forma de comunicación (Denger, 2001).

De acuerdo a una investigación realizada en 2006 estos enfoques de la búsqueda de calidad se dividen en tres grupos de los cuales se hablará en este capítulo.

3.1 Herramienta ARM

En términos generales, está herramienta se basa en atributos de calidad, modelos e indicadores que determinan si un requerimiento esta o no correcto.

Los aspectos de calidad en general en los requerimientos están divididos en dos bloques (Wilson, 1996):

Atributos de Calidad

Estos son los que definen aspectos como atomicidad, no ambiguo y completitud que son los principales problemas a atacar entre esta tesis, desde luego existen otros aspectos más como rastreabilidad, congruencia y demás pero no serán atacados en esta primera investigación.

Indicadores

Estos nos revelan la falta de calidad de algún atributo de calidad como los que utiliza la herramienta ARM desarrollada por el Centro Tecnológico de Aseguramiento de Software (SATC, por sus siglas en inglés).

A continuación se muestra una tabla con la clasificación detallada de los indicadores utilizados por la ARM que significa Medidor Automático de Requerimientos (Wilson, 1996):

Imperativos: palabras o frases que mandan la necesidad de algo

Continuación: palabras o frases que sirven de introducción a un nivel más bajo

Directiva: estas palabras son las que preceden a información ilustrativa

Opción: grupo de palabras que ofrecen un grado de libertad al desarrollador

Débil: grupos de palabras que son susceptibles a múltiples interpretaciones

Tabla 3.1. Estructura de indicadores en ARM

Imperativo	Continuación	Directiva	Opción	Débil	Incompleto
deberá	debajo	ejemplo	puede	adecuado	TBD*
debe	como sigue:	Es decir	pudiera	apropiado	TBS*
requiere	Siguiente:	Por ejemplo	opcional	poder	TBE*
aplicables	Listado:	imaginar		capaz	TBC*
son de	En particular:	tabla		Capacidad de	indefinido
responsables	Soportar:	Notar:		Facilidad de	indeterminado
desearía	y			efectivo	No limitado a
debería	:			requerido	Como minimo
				normal	
				prever	
				oportuno	

Una de las principales diferencias de éste enfoque con la propuesta de solución de esta investigación, es que aquí se utilizan un conjunto de indicadores fijos mientras que, en nuestra investigación se plantea atacar el problema por la gramática que compone a un requerimiento. La gramática por tanto no está ligada a ningún lenguaje.

3.2 Herramienta QuARS

En las investigación de Fabbrini (Fabbrini, 2000) él distinguió entre dos importantes factores, lo que es la calidad en la oración del requerimiento (RSQ por sus siglas en inglés Requirement Sentence Quality) y lo que es un aspecto diferente conocido como calidad en el documento de requerimientos (RDQ por sus siglas en inglés Requirement Document Quality), descripción de los mismo se explica a continuación:

RSQ: estos indicadores hacen referencia explícitamente al texto y sus oraciones, como por ejemplo oraciones opcionales, subjetivas, vagas y débiles gramaticalmente
RDQ: estos indicadores incluyen frecuencia de comentarios, índice de facilidad de lectura, frases sin referencia o sin explicación.

Basados en estas investigaciones se desarrollo una herramienta automática llamada QuARS por sus siglas en inglés Quality Analyzer of Requirements Specification que de acuerdo a sus investigaciones soporta el análisis y evaluación de calidad de las especificaciones de requerimientos.

En la siguiente tabla se muestra un detalle de los atributos de calidad utilizados por la herramienta QuARS.

Tabla 3.2. Atributos de Calidad QuARS

Tipo de Oraciones						
Implícita	Múltiple	Opcional	Débil	Subjetiva	Vaga	Referencia
Adjetivos Demostrativos	>1 Actor	posible	puede	Tener en mente	Fácil	De acuerdo a

Pronombres	>1 Verbo	eventualmente	debería	Tómese en cuenta	Fuerte	Basado en
Preposiciones	>1 complemento directo	En caso de	podría	Tómese en consideración	Bueno	Relativo a
Adjetivos	>1 complemento indirecto	Si es posible		Similar	Malo	Cumpliendo con
		Si es apropiado		Parecido	Útil	Conforme a
		Si es necesario		Mejor	Significante	
				Peor		
				Si es posible		

En resumen podemos notar que tanto la herramienta ARM como QuARS presentan atributos de calidad, un modelo e indicadores usados para la evaluación de de la calidad en los requerimientos expresados en lenguaje natural.

Los elementos que pueden causar problemas en los requerimientos son agrupados y contados por programas de computadoras, y de acuerdo a las investigaciones consultadas parecen ser efectivos en la búsqueda de defectos y ambigüedades que normalmente se encuentran.

El sistema QuARS es, en gran medida parecido al anteriormente evaluado ARM, y por tanto comparten la misma diferencia básica con el enfoque de ésta investigación; siendo está diferencia que en ésta propuesta no se manejan atributos fijos.

3.3 Sistemas de Reglas

Otro sistema para la evaluación de requerimientos se deriva de las investigaciones de Gotz y Rupp (Gotz, 1999); en este caso hablamos de un sistema basado en reglas, que de acuerdo a los investigadores incluye todas las reglas necesarias para detectar defectos, ambigüedades y palabras débiles en los requerimientos, en esta investigación de determinan tres importantes transformaciones que sufre un requerimiento comparado con la idea original que se deseaba transmitir.

Estas transformaciones se explican a continuación:

Omisión: esto ocurre cuando se reduce percepción de un fenómeno a un contexto propio de la persona.

Generalización: esto implica una separación de la experiencia sobre un contexto al asumir que el conocimiento sobre un fenómeno es del todo valido.

Distorsión: esto hace referencia a la transformaciones que pueden sufrir la ideas y representan el punto clave de donde se generan las reglas para intentar determinar los problemas que originan los defectos en los requerimientos. De acuerdo a las investigaciones mencionadas se cree que tienen un considerable grado de efectividad en la detección de defectos.

De acuerdo a investigaciones de Juristo (2000) se clasificaron los requerimientos en estáticos y dinámicos. En estas investigaciones se muestra como convertir un conjunto de requerimientos estáticos a un lenguaje denominado Static Utility Language (SUL, por sus siglas en inglés) y de la misma forma el procedimiento para convertir los que son dinámicos en Dynamic Utility Language (DUL, por sus siglas en ingles).

Cada unos de estos SUL y DUL resultantes esta expresado por una gramática formal y se compone de distintas estructuras de lenguaje natural, misma que según los autores puede ser convertida a lógica de predicados, de tal forma que cualquier declaración en cualquiera de los dos lenguajes no es ambigua. En esta investigación también se definen diferentes directrices para que los requerimientos estáticos y dinámicos se adapten y reformar la especificación de requerimientos.

En resumen, estos enfoques definen reglas para ser adaptadas en preparar requerimientos en lenguaje natural, mientras que por otro lado las directrices evitan la incorrecta construcción de requerimientos en lenguaje natural y las reglas sirven como funciones para el ingeniero de requerimientos en revisar la correctitud de los requerimientos resultantes, sin embargo, estos enfoques al final de cuenta están restringiendo el nivel de libertad con el que se desarrollan los requerimientos

Los sistemas de reglas tienen un ligero parecido con la solución propuesta en esta tesis, principalmente en el sentido en que una expresión regular, bien podría ser considerada una regla a seguir. Es por medio de expresiones regulares que la gramática está construida para la evaluación de los requerimientos.

3.4 Lenguajes de Patrones y Modelos

Un lenguaje de patrones de lenguaje es descripción ideada de un lenguaje de una forma más restringida. Existen distintos tipos de patrones como pueden ser los arquitecturales que definen la arquitectura de alto nivel

de un sistema de software, también existen otros patrones que se centran en detalles de programación, cosas administrativas del proyecto entre otros (Martinez, 2004)

Entre las investigaciones más relevantes de estos enfoques encontramos la realizada por Ohnishi (1994) quien desarrolló un analizador para el lenguaje japones de título X-JRDL el cual está basado en un concepto llamado modelo de marcos para requerimientos en el contexto de sistemas de archivos. El anterior modelo está compuesto por tres distintos marcos llamados Marco de Sustantivos, Marco de Casos y Marco de Funciones, cada uno de esos marcos restringe el vocabulario y el contexto de cada afirmación de los requerimientos. De acuerdo a los investigadores se afirma que con el modelo de marcos en requerimientos es posible transformarlos a una representación interna llamada CRD por sus siglas en inglés Conceptual Requirement Description, una vez en este formato, la herramienta X-JRDL puede analizar automáticamente la descripción de cada requerimiento.

Otra investigación importante es la realizada por Rolland (1992) el cual consiste en un modelo del esquema conceptual del futuro sistema, este modelo se construye a base de mecanismos lingüísticos los cuales sirven para abstraer el fenómeno del mundo real. Empujados por la necesidad de un desarrollo orientado a base de datos los investigadores definieron patrones y casos basándose en los casos originalmente definidos por Fillmore (1968), de eso se derivaron seis diferentes tipos de casos los cuales son: Agentivo, instrumental, dativo, factitivo, locativos, y objetivo.

Adicional a los casos, en la investigación también se categorizaron distintos clases de verbos y se distinguieron dos principales patrones lingüísticos que son un con junto de patrones formados por casos y tipos de verbos que se indican a continuación:

Patrones principales: estos permiten la asociación de casos a unidades de clausulas sintácticas

Patrones de oración: estos permiten la asociación de casos a clausulas de una oración

Como parte de esta investigación se desarrollo una herramienta llamada OICSI que adopta el enfoque propuesto, dicha herramienta está basada en el idioma Francés y busca como principal objetivo ser una herramienta de apoyo para el ingeniero de requerimientos.

Finalmente, tenemos las investigaciones realizadas por Barr (1999), las cuales se enfocan en patrones del lenguaje orientado a sistemas embebidos. En esta investigación se identifican dos tipos de patrones, los de especificaciones y los de oración que a su vez sirven de apoyo para la transformación de requerimientos en lenguaje natural sin estructura a una especificación de lenguaje formal. En esta investigación encontramos diferentes clases de patrones como:

Patrones si-entonces descritos por un esquema de reglas:

La estructura de un sistema de una oración dentro de un esquema de reglas está definida por el patrón ***si condición B, entonces consecuencia K*****,** por lo tanto, cada regla tiene una condición y su respectiva consecuencia en los que dentro de un sistema en tiempo real la condición y la consecuencia tienen una relación temporal

Patrones de hechos:

Estos patrones expresan un hecho valido dentro de la especificación de esquemas de consecuencias sin condiciones en donde la consecuencia que es parte de la regla es realizada si y solo si la condición a evaluar es verdadera.

De las investigación de Denger (2002; 2003), se desarrolló un enfoque para atacar el problema de la imprecisión en los requerimientos, esto logrado con el uso de patrones del lenguaje natural, un conjunto de reglas así como también formatos para los documentos.

Dentro de dichas investigaciones podemos encontrar diferentes patrones que fueron encontrados como *Patrones* en las oraciones de requerimientos funcionales, patrones de eventos, patrones de reacción, patrones computacionales, patrones de relación, patrones de excepción, patrones de aspectos especiales y finalmente patrones de oraciones para requerimientos no funcionales. En esta investigación también se formó un metamodelo para requerimientos funcionales adaptable para el dominio de sistemas embebidos.

Dentro de este mismo ramo existen otras investigaciones como las realizadas por Funchs (1996), en las que se intenta tener un lenguaje controlado restringido pero con un gran parecido al lenguaje natural, en efecto este lenguaje viene siendo un sub conjunto del lenguaje natural con sintaxis restringida con un vocabulario limitado a un dominio específico. En esta investigación se desarrolló un sistema llamado *Attempto,* el cual se encarga de traducir requerimientos en lenguaje controlado a estructuras de

primer orden de lógica de predicados que opcionalmente pueden utilizarse con *Prolog*.

Los enfoques de patrones podrían ser los más parecidos a esta investigación. Ya que es precisamente patrones los que se están buscando que se cumplan en los requerimientos. Esto ocurre atreves de expresiones regulares que conforman la gramática que se pretende cumplir para cada requerimiento.

3.5 Otros enfoques

Dentro de otros enfoques que buscan una calidad en los requisitos están los que proporcionan guías para la buena construcción de los mismos, dentro de estas investigaciones podemos citar las (Hooks, 1994), donde se describen problemas comunes y guías para cómo evitarlos. También se realizaron numerosas encuestas en busca de identificar las principales causas de los defectos en los requerimientos. Un trabajo similar lo podemos encontrar en las investigaciones de Firesmith (2003) en las que se describen las características de un buen requerimiento así como los potenciales problemas a ocurrir.

Otros enfoques como el presentado por Ambriola (2003) utilizan un método llamado Cooperative Interactive Requirement-Centric Environment (CIRCE, por sus siglas en inglés) o bien, Ambiente Cooperativo e Interactivo centrado en los requerimientos, en el cual se intentan concentrar en requerimientos de alta calidad. La idea principal de CIRCE es ir adaptando los requerimientos hasta poder extraer vistas concretas de los mismos.

CAPÍTULO 4

PROPUESTA DE SOLUCIÓN

4. Propuesta de Solución

En este capítulo hablaremos de la propuesta de solución a la problemática anteriormente planteada, así como también algunas de las posibilidades previamente evaluadas para llegar a la propuesta final.

Uno de los retos más grandes es encontrar un equilibrio entre el lenguaje natural, y a su vez poder generar una representación formal del mismo, en un contexto reducido para expresar requerimientos de software, pero sin recurrir a lenguajes restringidos que en la opinión del autor solo es cambiar los problemas actuales por otros nuevos sin llegar realmente a un progreso en el proceso de la elaboración y validación de requerimientos.

Existen numerosas problemáticas que atacar; dichas problemáticas en causa son generadas por la enorme complejidad del lenguaje natural. En el caso de esta investigación se utilizará el idioma inglés, debido a la disponibilidad de herramientas de *parsing* que existen para analizarlo sintácticamente, sin embargo, se intenta presentar una propuesta con suficientes fundamentos teóricos que hacen que la misma pueda ser extensible a diferentes lenguajes.

4.1 Bases de la Solución

Como hemos estado manejando a lo largo de este documento, es el lenguaje natural la parte central que forma a los requerimientos de software, sin embargo, antes de entrar en detalles como la estructura de un requerimiento en términos de lenguaje natural, es prudente comentar sobre distintos elementos léxicos y gramaticales del lenguaje natural mismo.

El elemento léxico más básico con significado lo conocemos como una *palabra*. Antes de abundar en temas como las diferentes combinaciones de las mismas o estructuras que pueden ser definidas, es conveniente revisar en este caso para el idioma inglés los grupos de palabras que tenemos, y mismos que suelen ser muy parecidos a otros idiomas aunque con escrituras y gramáticas muy diferentes.

Sustantivos

En inglés se conoce como “Noun”, y en un contexto simple, es una palabra que se utiliza para indicar una persona, un lugar, una cosa o alguna idea.

Sin embargo, estudiando mas a detalle este grupo de palabras, y viéndolas desde un enfoque de lingüística un sustantivo es un elemento de una categoría léxica abierta, es decir que admite nuevos elementos, y dichos elementos pueden ser utilizados como la palabra principal en el sujeto de una oración, el objeto de un verbo, o bien el objeto en una preposición (Loos, 2003).

El hecho de pertenecer a un grupo lingüístico define patrones en común del uso de sus elementos y como estos se combinan con distintas expresiones, estas reglas en los sustantivos son variadas y dependen explícitamente del lenguaje en cuestión. En el idioma inglés, los sustantivos pueden ser definidos como aquellas palabras que pueden aparecer con artículos y adjetivos o bien como el elemento principal en un sintagma nominal. Para comprender un poco mas este punto es conveniente definir que un *sintagma nominal* se refiere a una frase la cual se basa en un sustantivo, un pronombre y otro conjunto de palabras parecidas a los

sustantivos que suelen ser acompañados por modificadores como los adjetivos.

Algunos ejemplos de adjetivos (en color **negrita**) se muestran a continuación, mismos que están en lenguaje inglés por ser el idioma en cuestión bajo objeto de estudio.

The **cat** sat on the **mat.**
Please hand in your **assignments** by the **end** of the **week**.
Cleanliness is next to **Godliness**.

Pronombres

En el área de lingüística un pronombre es una palabra funcional es aquella que expresa el mismo contenido que otra palabra, frase, oración o sentencia en donde el significado de la misma (del pronombre) puede ser recuperado gracias al contexto que la rodea y son comúnmente utilizados para evitar la excesiva repetición de palabras.

En el idioma Inglés por ejemplo tenemos las palabras como "*it*" con la cual podemos substituir el nombre de algún objeto u otro ejemplo seria la palabra "*she*" con la que podemos sustituir el nombre de una mujer, a estos elementos remplazados se les conoce en el área de la lingüística como antecedente (Postal 1966).

Un aspecto importante a considerar es que los pronombres son potencialmente ambiguos cuando estos no tienen un antecedente, evaluemos la frase "*Lisa gave the coat to Phil*", misma que puede ser

remplazada por "*She gave it to him*" siempre y cuando los antecedentes hayan sido previamente mencionados en el texto, ya que entonces se puede inferir el significado, si esto no es así, entonces se estaría cayendo en un problema de ambigüedad como ya lo hemos explicado anteriormente.

Algunos ejemplos de pronombres son:
I love **you**.
Take **it** or leave **it**.

Adjetivos

En gramática un adjetivo es aquella palabra que describe a otra en donde su principal rol sintáctico consiste en calificar a un sustantivo o bien a un sintagma nominal para poder dar más información sobre el objeto de acción (Capital, 2012).

En la actualidad en el área de la lingüística los adjetivos ya son separados en un grupo separado ya que previamente estaban comprendidos junto con los determinantes que también eran considerados adjetivos.

En muchos lenguajes, como lo es el caso de idioma inglés, se hace una marcada distinción entre los adjetivos y los adverbios ya que mientras los adjetivos califican adjetivos, los adverbios modifican verbos, adjetivos o incluso otros adverbios.

A continuación mostramos diferentes ejemplos de adjetivos, y un interesante ejemplo con la palabra "*interesting*" y como esta puede cumplir diferentes roles de adjetivo.

That's an **interesting** idea. (attributive)
That idea is **interesting**. (predicative)
Tell me something **interesting**. (postpositive)
The **good**, the **bad**, and the **ugly**. (substantive)

Verbos

Los verbos son palabras cuya sintaxis lleva a principalmente al de una acción, como por ejemplo, jugar, correr, patear, brincar, etc. Sin embargo también expresan otras cosas como ocurrencias o bien estados de ser y estar. En muchos lenguajes los verbos tienen diferentes formas que indican si la acción ya se realizó, se está realizando o bien será realizada en un futuro.

Los verbos tienen diferentes valencias que se define por la cantidad de argumentos sobre los cuales este actúa, o bien en otras palabras podríamos decir que define el nivel de detalle sobre el objeto de acción. Las valencias posibles para el idioma Inglés son:

Intransitiva (valencia 1): Se da cuando el verbo solo tiene un sujeto como por ejemplo: "*he runs*", "*it falls*"
Transitiva (valencia 2): Esto ocurre cuando se tiene el verbo con su sujeto y además un objeto directo, por ejemplo: "*she eat fish*", "*we hunt nothing*"
Transitiva (valencia 3): Este es un nivel mas grande de detalle en donde no solo se tiene un sujeto y objeto directo, sino que se cuenta también con uno indirecto, por ejemplo: "*he gives her a flower*".

Adverbio

Estas palabras se encargan de modificar principalmente el significado de un verbo, aunque también pueden modificar a otro tipo de palabras, siempre y cuando estas no sean sustantivos ya que para ellos existen los adjetivos y determinantes.

Entre los elementos lingüísticos que pueden ser modificados por un adverbio encontramos verbos, adjetivos, clausulas, oraciones e incluso otros adverbios. Los adverbios principalmente contestan preguntas del tipo, ¿cómo?, ¿de qué manera?, ¿cuándo?, ¿dónde? Y ¿en qué medida? A este fenómeno se le conoce como función adverbial y se lleva acabo no solamente por una simple palabra sino por todo un conjunto de ellas que pueden formar lo que se conoce como "sentencias adverbiales" o bien "clausulas adverbiales".

Algunos ejemplos de adverbios en inglés serian "*incredibly*", "*extremely*" y "*sideways*", para mostrar más claro su uso mostramos los siguientes ejemplos en oración.

I found the film **incredibly** dull.
Crabs are known for walking **sideways**.
I **often** have eggs for breakfast.

Preposiciones

Este es un grupo de palabras cuya característica principal es expresar una relación de espacio o bien servir para marcar distintas funciones sintácticas o roles semánticos (Huddleston, 2002).

La función primaria entonces de estas palabras es la de relacionar, para lograr esto generalmente se combina con otro constituyente conocido como “complemento” para formar entonces una frase preposicional, esto para lograr una relación entre el complemento y el contexto en el cual este ocurre.

Las preposiciones generalmente son palabras que se escriben antes del complemento y entonces de ahí su nombre pre-posición, sin embargo, es importante destacar que podemos tener preposiciones después del complemento conocidas como post-posición o bien en ambos lados entonces llamadas circu-posición, para referirse a ellas en general se debería utilizar ad-posición, aunque lo más común es utilizar el término pre-posición para referirse a cualquiera de los casos (Huddleston, 2002,1).

Algunos ejemplos de preposiciones del idioma inglés serian entonces:
the, of, and, to, a, in, that, it, is, was, I, for, on, you.

Conjunciones

Este tipo de palabras son utilizadas para conectar dos palabras, oraciones, frases o incluso clausulas. Un discurso conjuntivo es aquel que se utiliza para unir oraciones, sin embargo, la definición concreta de conjunción necesita ser definida particularmente para cada lenguaje en cuestión. En términos generales podemos entonces decir que una conjunción es un elemento invariable que puede o no posicionarse entre los elementos que une.

Existen numerosas cantidades de conjunciones, entre las más utilizadas en el idioma Inglés, tenemos “for, and, nor, but, or, yet, so” (Paul, 2009).

A continuación se presentan algunos ejemplos de conjunciones y como son utilizadas en frases.

For, (razon): He is gambling with his health, **for** he has been smoking far too long

And, especifica ideas no contrastada: They gamble, **and** they smoke

Nor, especifica idea negativa: They do not gamble, **nor** do they smoke

But, especifica ideas contrastada: They gamble, **but** they don't smoke

Or, especifica idea alternativa: Every day, they gamble, **or** they smoke

Yet, especifica excepcion: They gamble, **yet** they don't smoke

So, (consecuencia): He gambled well last night, **so** he smoked a cigar to celebrate

Interposición

También conocido como exclamación, este grupo de palabras se utilizan para expresar emociones o sentimientos por parte del escritor. Complementos de pausas también son consideradas interposiciones, tales como “*hu, er, um*” y otras tantas del idioma Inglés.

Convenciones como “*Hi, Bye, Goodbye*” también se pueden interpretar como interposiciones, así como también exclamaciones como “*Cheers, Hooray*”.

De hecho, al igual que sustantivos o pronombres, suelen ser acompañados por símbolos de exclamación según el nivel de emoción con el que estos deseen ser expresados.

A continuación mostramos algunos ejemplos de interposiciones comunes:
Shh: se interpone por una exclamación de **silencio.**
Psst: suele utilizarse para llamar la atención, como indicando "**aqui**"
Ugh: generalmente expresa un **disgusto** por algo
Phew: suele utilizarse para expresas una sensación de **alivio**

Ahora que ya hemos visto los diferentes tipos de palabras que se manejan, es necesario ver como esto está ligado a los requerimientos funcionales de software que al final de cuentas son sentencias como otras que se forman de este conjunto de palabras.

4.2 Especificación de Requerimientos

El bloque más grande de jerarquía entre los requerimientos sería lo que se conoce como Especificación de Requerimientos de Software, el cual está compuesto de la completa descripción de todo el sistema que será desarrollado y puede incluir incluso casos de uso que describan las interacciones del usuario con el sistema. Este documento incluye entonces las funciones del sistema pero también debe incluir sección de requerimientos no-funcionales en los cuales se marcan las limitantes de diseño e implementación del mismo, tales como especificaciones de

rendimiento, estándares de calidad. En base a lo anterior entonces podemos decir que la ingeniería de requerimientos es una rama de la ingeniería de software (Pierre, 2004).

Estas especificaciones son enunciados compuestos por lo que se conoce en inglés como *Part-of-Speech,* y es el conjunto de grupo de palabras previamente definidas que juntas forman una oración con un significado interpretable por un agente distinto al escritor, y en donde se espera transmitir una idea de lo que se está requiriendo que el sistema efectué una vez que este se ha terminado.

En la siguiente sección mostraremos desde un punto de vista matemático como el LN juega un papel en los enunciados para posteriormente explicar cómo se presenta la solución propuesta.

4.3 Descripción Formal de la Solución

Como hemos visto en secciones anteriores, existen diferentes tipos de requerimientos, sin embargo, la problemática a abordar únicamente incluye a los de tipo Funcional, los cuales son los encargados de proporcionar la información requerida para crear las funcionalidades que se incluyen en cualquier sistema de software (Laplante, 2007).

Anteriormente mencionado tenemos que los requerimientos no son más que enunciados comunes que están compuestos al igual que cualquier otro enunciado por elementos del *Part-of-speech*, por lo tanto, podemos olvidarnos de su contenido y verlos estrictamente como un conjunto finito de

distintos tipos de palabras que entre si forman la idea que se pretende transmitir. Esto entonces expresado con teoría de conjuntos nos daría lo siguiente:

$$R = \{W1, \ldots, Wn\} = W$$

En donde podemos encontrar que un Requerimiento R está definido entonces por ***n*** cantidad de palabras $W_1 \ldots . W_N$, que al final resultan ser un conjunto finito de Palabras *W*.

Ahora continuamos a ver las palabras no como un simple grupo de letras, sino como un elemento léxico que tiene un grupo al cual pertenece según el *Part-of-Speech* y no solo eso sino que el mismo grupo de letras, es decir la palabra, puede tener distintos significado según la interpretación del lector, y este significado no precisamente puede ser el que el escritor quiso dar a entender y ahí es donde inician los problemas. Para tener un panorama más claro de cómo quedaría esto entonces formulado con teoría de conjuntos presentamos lo siguiente:

$$W_i = \{t, \{M\}\}$$

En donde mostramos ahora la representación formal de como se compone cada palabra *W* que se utilizo en la formulación anterior, en este caso una palabra se compone de una *tupla* formada por un elemento *t* que representa el tipo de palabra dentro del *Part-of-speech* a la cual pertenece, y un conjunto *M* que está formado por todos los potenciales significados que esta palabra tiene.

Si esta definición entonces la pasamos a la primera formalización de representación de un requerimiento entonces tendremos una versión más compleja del mismo, que sirve para ver más explícitamente como un simple enunciado puede ser visto en términos de conjuntos, con esto entonces tenemos:

$$R = \{\{t, \{M\}\}_1, \ldots, \{t, \{M\}\}_n\}$$

En esta representación podemos observar que se tiene una formulación un tanto más compleja en donde se puede apreciar como se ve en conjunto una secuencia de palabras y sus componentes, y entonces analizar las problemáticas a atacar en esta tesis y como se presentan las mismas.

El primer problema que presentamos en esta formalización es entonces, sin ningún orden en particular el de *ambigüedad*. Como hemos comentado en secciones anteriores, existen numerosas formas de *ambigüedad*, de las cuales por el momento únicamente se atacará la *léxica* que reside en la palabra misma y sus potenciales significados.

Definimos para cuestiones de esta tesis entonces que la ***ambigüedad***, en este caso denotada como *B* se genera cuando el número de elementos del conjunto *M* de significados para una palabra *W* es mayor a 1 o bien denotado en teoría de conjuntos tenemos lo siguiente:

$$B = \{W_i\ (M)\} > 1$$

Ahora bien, otro problema que se intenta atacar es el hecho de la falta de ***atomicidad***, o también conocido como *conjunción* esta bajo nuestra investigación se genera cuando se tienen tipos de palabras especificas dentro del grupo de *Part-of-speech,* las cuales se conocen como conjunciones y su presencia causa la posibilidad de separar una sentencia en sentencias más pequeñas, algo que es fundamental en un requerimiento pues debe ser lo más preciso posible y hablar solo de una sola cuestión.

Este problema de conjunción **C** lo expresamos como sigue:

$$C = \{W_i\ (t) \rightarrow conjunction\} > 0$$

Aquí podemos notar que nuevamente se hace referencia al conjunto palabra denotado con la letra *W*, sin embargo, en este caso, en donde se presta atención es al valor del elemento *t*, que en este caso hace referencia al tipo de palabra dentro del bloque *Part-of-Speech*, si cualquiera de las palabras *W* en el requerimiento *R* es parte del grupo conjunciones, entonces decimos que se presenta el problema anteriormente mencionado.

Ahora bien, para abordar el siguiente problema que sería el de completitud, es necesario nuevamente redefinir el concepto de requerimiento pero esta vez por una versión más completa y compleja que nos permita definir espacios requeridos dentro de los conjuntos para poder tener un elemento cuantitativo que defina el nivel de completitud.

Para esta validación entonces aunque se siguen cumpliendo las proposiciones anteriores como la que define a un requerimiento como *un conjunto finito de palabras*. Es necesario hacer una separación y definir

grupos de palabras que forman diferentes secciones de un requerimiento de software que si bien no hay patrón que forzosamente tiene que seguirse, si existen elementos importantes que se esperan en dichos requerimientos, y basados en las investigaciones de Juárez-Ramírez (2011), se ha definido un grupo de elementos que se esperan existan en un requerimiento funcional para en cierta medida garantizar que cumplen con su objetivo de especificar funcionalidades del sistema.

Se han detectado entonces 3 elementos básicos que son: *actor, función, detalle de función*. Estos elementos se explican a continuación:

Actor

Este es un elemento primordial que debe ser incluido para evitar incertidumbres del tipo “quién hace qué?” Es importante entonces definir explícitamente quien es el encargado de realizar cualesquiera será la función que se exprese en el requerimiento, pudiera este ser el usuario, el sistema, o demás potenciales actores y es importante definir el responsable del rol.

Función

Un elemento que es imposible no tener presente en un requerimiento funcional es entonces precisamente la “función” a la cual se está haciendo referencia, generalmente aquí encontraremos entonces un verbo, que representara la acción que debe ser ejecutada por el elemento anteriormente descrito que fue el *actor* en cuestión.

Detalle de función

Finalmente tenemos este elemento que va muy íntimamente ligado con la “función” en cuestión, pues hace referencia a “detalles de la función”, ya que muchas veces una misma función puede ser realizada bajo distintas circunstancias o bien variar la funcionalidad de acuerdo a diferentes condiciones, mismas que se esperaría se especifiquen explícitamente en el requerimiento, pues como ya hemos visto anteriormente, se debe cuidar la atomicidad, y no incluir más de una circunstancia en el mismo requerimiento.

Entonces, una vez entiendo estos 3 elementos básicos de un requerimiento funcional podemos definir un requerimiento como una secuencia más compleja de conjuntos como:

$$R = \{\{A\}, \{F\}, \{D\}\}$$

Aquí podemos observar entonces que ahora el requerimiento *R* ya esta mas ordenado y ahora se compone de 3 diferentes conjuntos que representan los elementos mínimos esperados que vimos anteriormente, sin embargo, la primera proposición en donde se definía como un conjunto finito de palabras *W*, se sigue cumpliendo pues los conjuntos *A*, *F*, y *D* se forman únicamente de palabras *W* y todos ellos son requeridos para determinar el requerimiento como **completo**, para esto entonces se definen las siguientes condiciones:

$$\text{Condición \#1: } \{A \subset W\} \neq \emptyset$$
$$\text{Condición \#2: } \{F \subset W\} \neq \emptyset$$
$$\text{Condición \#3: } \{D \subset W\} \neq \emptyset$$

Las anteriores condiciones deben cumplirse, es decir, deben existir los 3 elementos mínimos en el requerimiento funcional, de otra forma será catalogado como incompleto.

Para un mejor entendimiento de esta propuesta entonces se prosigue al siguiente ejemplo de su funcionamiento:

Teniendo el requerimiento:

```
The system will check if password match user in database
```

Se puede decir que tenemos entonces el conjunto de palabras denotadas W de tal forma que W_1 es para *The*, W_2 para *system*, y así continuando hasta W_{10} para *database*

Como se explicó anteriormente, las palabras se componenen de un tipo y sus significados, por ejemplo para *system* tenemos:

```
t =  noun (sustantivo en español)
{M} = Un conjunto de significados para la palabra.
```

De acuerdo al diccionario Oxford, en la rama de computación, solo tiene un significado el cual es *“a group of related hardware units or programs or both, especially when dedicated to a single application”.*

Una vez realizado este tipo de análisis para cada palabra en cuestión, se procede entonces a buscar patrones de ayuden a identificar las diferentes problemáticas. En el siguiente capítulo entonces hablaremos de cómo esta estrategia de solución ha sido desarrollada.

CAPÍTULO 5

ARQUITECTURA DE SOLUCIÓN

5 Arquitectura de Solución

En este capítulo hablaremos de cómo se ha diseñado la propuesta de solución. Los temas serán abordados desde un punto de vista más técnico y se intenta que esta sección sirva como guía, incluso para posteriores desarrollos que bien podrían ser llevados a cabo bajo esta misma arquitectura o bien un nuevo desarrollo.

En este capítulo entonces definiremos cómo está formada la arquitectura de la herramienta denominada en inglés “Natural Language Automatic Requirement Evaluator” (NLARE) que se está desarrollado como medida para demostrar la viabilidad de la propuesta de solución y su factibilidad.

El sistema se compone básicamente de 5 módulos de programación, cada uno con una funcionalidad en específico con la idea de poder tener una arquitectura que sea fácilmente escalable a múltiples lenguajes aunque de inicio esta investigación está centrada únicamente en el idioma inglés.

5.1 Introducción a NLARE

La herramienta titulada NLARE también traducida en español como Evaluador Automático de Requerimientos en Lenguaje Natural.
Más adelante en esta tesis se explicará a detalle el funcionamiento de NLARE, sin embargo, la idea principal consiste en poder definir una estructura formal de elementos mínimos que compongan a un requerimiento funcional y a su vez un programa de cómputo capaz de entender esa estructura y reglas para poder así evaluar el requerimiento de una forma

automática.

Para lograr el objetivo principal de NLARE se utilizan tecnologías de procesamiento natural, en específico el *framework* de desarrollo llamado NLTK por sus siglas en inglés *Natural Language Tool Kit*.

Entre otras tecnologías también se incluyen *parsers*, evaluadores de expresiones regulares, diccionarios y cuerpos de texto de entrenamiento para formar en conjunto la solución propuesta que explicamos a continuación.

5.2 Arquitectura NLARE

En esta sección exploraremos cómo está diseñada la herramienta NLARE con un enfoque independiente a cualquier lenguaje de programación, se espera que esta sección sirva para comprender la lógica detrás de NLARE y pueda servir para posibles replicas del funcionamiento de la herramienta en arquitecturas distintas a NLTK.

Para ayudar a tener una visión global de la arquitectura mostramos la Figura 5.1, la cual ha preservado títulos en inglés debido a que algunos elementos no tienen traducción directa o bien son muy poco conocidos en su potencial traducción.

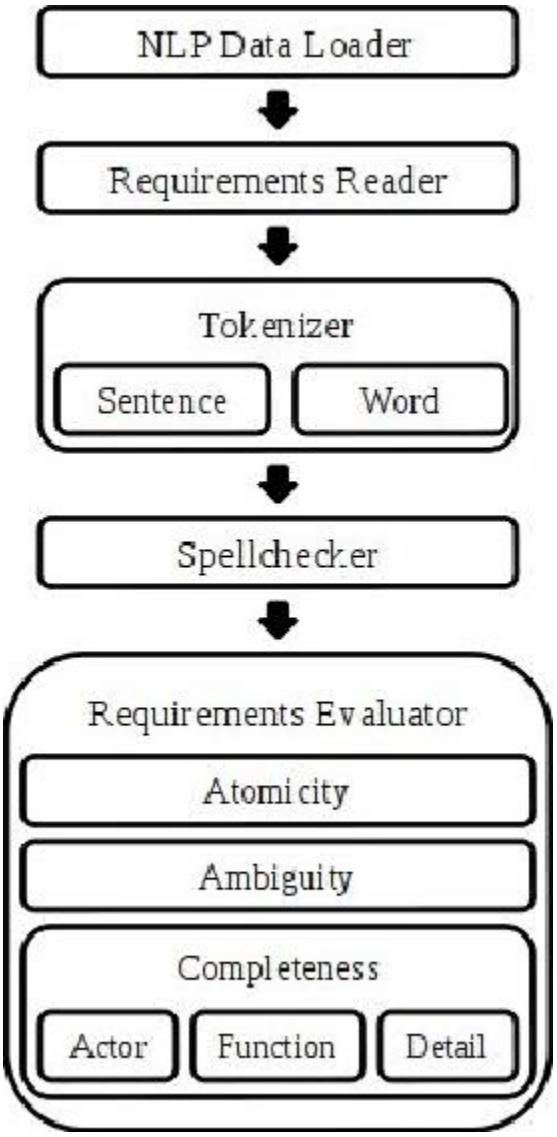

Figura 5.1. Arquitectura NLARE

Ahora se continuará describiendo cada uno de los módulos que componen a la arquitectura.

5.2.1 NLP Data Loader

Esta es la sección encargada de cargar toda la colección de información proporcionada por el ambiente de desarrollo NLTK, con la cual

se configurara la aplicación para poder entender el lenguaje inglés, si se desea manejar otro lenguaje, aquí implicaría entonces entrenar a la aplicación con otro conjunto de información para reconocer un grupo distinto de palabras y su agrupación en Part-of-speech.

En la versión 1.0b de NLARE se distribuye un conjunto de programas y texto de casi 500 MB, que se distribuye de la siguiente manera:

Chunkers (10.8 MB) (maxent_ne_chunker)

Un chunker es un programa que se encarga de realizar precisamente la tarea llamada “chunk” también conocida como “shallow parsing” o “light parsing” es decir un *parser* que funciona muy rápido y se encarga de identificar elementos léxicos dentro de una oración sin evaluar su estructura interna. Esta técnica es ampliamente utilizada en procesamiento de lenguaje natural y se podría comparar con la tarea de análisis léxico que realiza un compilador a un lenguaje de programación.

Corpoa (458.7 MB)

Esta sección sin duda es la más extensa y ocupa casi el 92% de toda la información cargada por NLARE, y se compone de millones de palabras de diferentes tipos de textos que sirven como método de configuración y entrenamiento para NLARE con respecto al lenguaje va a necesitar evaluar, en este bloque se puede entonces dar una idea a NLARE como se estructura el lenguaje inglés. Este conjunto de textos está dividido 53 secciones con diferentes categorías y tipos de textos.

En ambientes Unix es posible sacar estadísticas de que tan grande es cada bloque con el comando “du -h | sort -n”, sin embargo aquí mostramos los

catálogos más grandes de información con los que cuenta NLARE 1.0b.

Brown_tei (54.2 MB)

Esta es una versión en XML directamente derivada del “A Standard Corpus of Present-Day Edited American English, for use with Digital Computers” que fue desarrollada por W. N. Francis and H. Kucera en 1964 en el departamento de lingüística de la universidad Brown en Estados Unidos.

Wordnet (34.7 MB)

Esta es una recopilación de distintos fragmentos de Wordnet la cual es una reconocida base de datos léxica para el idioma inglés (Miller, 1990), la base de datos ha ido creciendo durante el tiempo, en 2006 se contabilizaron que contenía más de 155,287 palabras organizadas en 117,659 grupos de semántica equivalente (synset en inglés)

Senseval 2 (15.7 MB)

Este es un conjunto de texto en formato XML el cual ha sido previamente etiquetado para ayudar a procesamiento de eliminación de ambigüedad en textos, y se especializa en cuatro palabras básicas únicamente: "hard", "interest", "line", "serve". Esta colección ha sido realizada por Ted Pedersen de la Universidad de Minnesota, Estados Unidos.

Gutenberg (11.3 MB)

Esta es una colección realizada por el proyecto Gutenberg que reune fragmentos de libros de reconocidos autores William Blake, Lewis Carroll, Maria Edgeworth, John Milton, William Shakespeare, entre otros. La colección cuenta con fragmentos de 18 libros que sirven como entrenamiento

del lenguaje inglés para NLARE 1.0b.

Treebank (5.7 MB)

Esta es una recopilación del 5% del árbol Penn realizado en 1995 en donde encontramos diferentes secciones ya etiquetadas, sin etiquetar, mixtas, parseada y plena.

Y así entonces se cuentan con otros catálogos de información para completar un total de 53 que incluso incluyen de distintos idiomas, mismos que de momento no se están utilizando pero podrían ser de gran utilidad al intentar portar NLARE a múltiples lenguajes.

Ahora bien pasamos a otro bloque del conjunto de información que carga NLARE, en este caso es la sección de gramáticas, la cual se explica a continuación:

Grammars (4.0 MB)

En esta sección encontraremos configuraciones de gramáticas y ejemplos de sentencias etiquetadas que pueden servir para mejorar el rendimiento y la efectividad de las aplicaciones que utilizan procesamiento de lenguaje natural, incluso se tiene un bloque del lenguaje español para potencial futuro uso.

Se tiene también otro bloque que es de ayuda (Help (20.3 KiB)) el cual de momento no se ha utilizado. Dentro de lo que si se utiliza está el siguiente bloque.

Stemmers (15.1 KiB) (rslp)

Aquí se cuentan con un conjunto de reglas y pasos para poder realizar el proceso precisamente conocido en inglés como “stemming” el cual es el proceso de reducir una palabra a su “stem” que en español significa “tallo” es decir, a la versión más corta de la palabra en donde se pueda sacar su contenido léxico más representativo que luego es transportado a distintas variantes de palabras, esto ayuda a reducir la complejidad de evaluación mientras aún se mantiene un concepto básico, por ejemplo para el idioma Inglés tenemos:
“fishing”, “fished” y “fisher”, estas palabras al ser pasadas por un stemmer, esperamos como resultado “fish” que es la composición básico de la cual estas otras fueron derivadas.

Otra sección incluida en el conjunto de información proporcionada por NLTK y de uso crítico para NLARE es la sección de etiquetadores o “taggers” en inglés que comentamos a continuación.

Taggers (8.9 MB)

Una herramienta de este tipo es encargada de etiquetar cada palabra de una oración con el nombre del grupo que le corresponde según la definición del Part-of-speech que ya hemos previamente discutido en esta tesis, eso se consigue basándose en su definición a si como también el contexto que rodea a una palabra, un ejemplo práctico es como en la primaria se enseña a los niños a identificar diferentes secciones en una oración como el sujeto y verbo.

En el caso de NLTK contamos con 2 diferentes taggers, que son hmm (2.0 MB) y el otro que es el que se utiliza actualmente en NLARE que es el

Maxent (8.2 MB), el nombre esta derivado del inglés Maximum Entropy, que significa Maxima Entropia, que es en términos más generales es la medición de la incertidumbre que rodea a una variable aleatoria.

Finalmente la última sección de la información cargada y también de vital importancia para NLARE es la de la separación por “tokens”.

Tokenizers (16.8 MB) (punkt)
En el caso de las herramientas proporcionadas para NLTK solo disponemos actualmente de un solo “tokenizer” en este caso es el punkt el cual esta previamente entrenado con información de Kiss y Strunk (2006) y soporta hasta 16 idiomas, en el caso de NLARE utilizamos la versión entrenada para el Inglés, la cual tiene como fuente Penn Treebank y fue entrenada con textos de Wall Street Journal y se compone aproximadamente de 469,000 “tokens”

Con esto concluye entonces la sección de NLP Data Loader, es importante mencionar que es posible configurar NLARE para poder acceder a esta información en diferentes rutas de acuerdo a las necesidades específicas del uso que se le dé a la misma.

5.2.2 Requirements Reader

Este es un modulo muy simple por el momento sin embargo por cuestiones de diseño se decido que aunque chico tenga su propia sección pues es potencialmente extensible en muchas formas, podría programarse para leer diferentes formatos de documentos de requerimientos, talves XML, Json, o cualquier otro formato que se requiriera y se vea una utilidad de extender la funcionalidad de este modulo a su lectura, una vez realizada esta, el flujo de NLARE seguiría intacto sin importar la fuente de la cual proviene la información.

En la versión actual NLARE 1.0b únicamente se cuenta con un formato aceptable de documento de requerimientos este es un archivo de texto plano que utilice la codificación de caracteres UTF-8 el cual fue diseñado para tener compatibilidad con ASCII y evitar problemáticas de ordenamiento de bits que ocurrían con UTF-16 y UTF-32, otras codificaciones podrían funcionar sin embargo no han sido probadas y su potencial resultado es entonces desconocido.

Durante el diseño de NLARE se estuvieron evaluando diferentes posibles formatos para representar los requerimientos, por ejemplo etiquetas del tipo RF_1 para indicar el Requisito funcional 1, o RU_1, que bien podría ser el requerimiento de funcionalidad 1, pero finalmente se ha decidido que no haya ningún elemento interfiriendo con el requerimiento y sea este integro como se ha redactado que será evaluado.

El formato debe entonces incluir un (1) requerimiento por cada renglón, siendo entonces el carácter de salto de línea el que define donde termina y

donde empieza el siguiente requerimiento. Es prudente comentar que durante las pruebas se noto como los documentos de requerimientos realizados con el editor Kate bajo Linux se generaban correctamente pero Microsoft Notepad no era capaz de identificar correctamente el salto de línea, problema que no ocurría con editores más preparados como NotePad++ una excelente herramienta de edición que se utilizo para las pruebas bajo Windows.

Un ejemplo entonces del archivo de texto resultante sería el siguiente:

Nombre del archivo: req0

Contenido:

```
The System will generate a random color for every piece
The System will send pieces from the server to the game
window
```

El nombre del archivo puede ser cualquiera que siga con los estándares definidos por el sistema operativo en este caso, el nuestro lleva por título req0 (sin extensión) y se puede ubicar en cualquier ruta pues esta también es configurable por NLARE.

5.2.3 Tokenizer

Como ya habíamos visto anteriormente aquí se está utilizando el tokenizador Punkt, y existen dos principales procesos de los que este modulo se encarga, primeramente hace una revisión preliminar sobre de cuantas sentencias se compone el requerimiento, ya que si este se compone de más de 1, caería en nuestra categoría de no-atómico (o conjugado) pues se está buscando que no tenga variantes y solo hable de una cuestión, y la potenciales cuestiones adicionales sean entonces manejadas en un requerimiento separado.

Si el resultado de la detección de sentencias da el valor esperado (uno, 1), entonces se precede a una segunda “tokenización” en donde el requerimiento no es visto mas como una cadena de texto, sino como un arreglo en donde cada palabra guarda su posición independiente esto con fines de un manejo mas optimo del procesamiento que se requiere aplicar al mismo.

Si quisiéramos ver este pasó en función de la teoría que rodea a la solución, estaríamos entonces en la separación de los requerimientos por partes, como se definió anteriormente en el capitulo 4.

Requerimiento:

```
R = {W1,..., Wn} = W
```

5.2.4 Spellchecker

Este modulo cumple una función simple desde el punto de vista de desarrollo pero que sería un enorme trabajo si no fuera porque se ha aprovechado todos los recursos que proporciona Enchant para la revisión ortográfica del texto, ya que errores ortográficos pudieran generar ambigüedad si el lector intenta asumir la palabra correcta y esta no coincidiera con lo que realmente intentaba expresar el escritor.

Es importante realizar esta separación como modulo independiente aunque de momento solo se tenga un idioma pues se planea poder extender este trabajo para aceptar múltiples idiomas y entonces será requerido ligar a diferentes librerías de corrección.

Dentro de la lógica de NLARE 1.0b de momento cualquier palabra que no sea reconocida por la librería Enchant y que no sea parte de un conjunto de palabras propias del proyecto que pueden ser definidas en un archivo de palabras reservadas será entonces marcada como “desconocida” y por tanto el requerimiento o requerimientos que la contengan serán catalogados como ambiguos.

El archivo de palabras reservadas será un archivo texto plano de nombre cualesquiera en codificación UTF-8 o ASCII, que contenga una palabra por renglón que definirá palabras propias del proyecto en cuestión.

5.2.5 Requirements Evaluator

Previas todas las configuraciones y validaciones anteriores si se ha determinado que un requerimiento es apto para las pruebas realmente que definen a esta investigación entonces son enviadas a este modulo que es el encargado precisamente de realizar las evaluaciones previamente definidas en capítulos anteriores las cuales son Atomicidad, claridad (no-ambigüedad) y Completitud.

Lo primero que se realiza en este modulo es entonces una revisión gramatical del enunciado que define al requerimiento para entonces poder realizar un proceso conocido como POS-tagging, que se deriva del inglés Part-of-speech tagging, que es un etiquetado de cada elemento del arreglo, en este caso cada palabra, la cual según su contexto se determina a qué grupo del Part-of-speech pertenece y en nombre de este grupo es entonces agregado a cada elemento del arreglo para generar un arreglo de tuplas en donde la tupla se compone entonces de (a,b) en donde 'a' representa la palabra en cuestión y 'b' entonces al grupo del Part-of-speech a la cual pertenece según la oración en cuestión.

Posteriormente se hace una búsqueda de elementos que puedan generar conjunción dentro de la oración, esto se consigue revisando el arreglo tokenizado y que previamente ya fue etiquetado, entre los elementos principales de búsqueda tenemos entonces a los que pertenecen al grupo Part-of-speech denominado “Coordinating conjunction”. De acuerdo a la teoría desarrollada en el capitulo 5, esto es equivalente a:

$$C = \{W_i\ (t) \rightarrow conjunction\} > 0$$

Posteriormente se realiza una búsqueda más exhaustiva para determinar elementos ambiguos dentro del requerimiento, en este caso los grupos de Part-of-speech que intentan evitar son los siguientes:

Tabla 5.1. Grupos Part-of-speech buscados para evitar ambigüedad

Etiqueta NLTK	**Descripción Original**	**Descripción Español**
FW	Foreign word	Palabra extranjera
JJ	Adjetive	Adjetivo
JJR	Adjective, comparative	Adjetivo comparativo
JJS	Adjective, superlative	Adejtivo superlativo
RB	Adverb	Adverbio
RBS	Adverb, superlative	Adverbio superlativo

Si se detecta entonces alguna palabra que pertenezca a cualquier de los grupos Part-of-speech definidos en la Tabla 5.1, entonces se hace una recopilación de la palabra en cuestión para tener una referencia de la palabra causante de la potencial ambigüedad dentro del requerimiento evaluado.

En esta etapa, entonces se estaría realizando la etapa que se definió en el capitulo 4 como búsqueda de multiples significados, expresado formalmente como:

$$B = \{W_i\ (M)\} > 1$$

Durante este proceso se continúa revisando toda la cadena del requerimiento y si hubiera más palabras que generan ambigüedad se van

marcando todas para mostrar al final un resumen con el detalle de toda la evaluación del requerimiento.

Para la evaluación de la completitud fue necesario definir una gramática pues la evaluación palabra por palabra en este caso no ayuda pues es necesario poder identificar patrones de elementos que nos indiquen si los elementos que buscamos, que en este caso son Actor, Función y Detalle, se encuentran en el requerimiento.

La gramática definida que realizada en base a expresiones regulares, para poder tener una mejor compresión de los símbolos utilizados a continuación mostraremos alguna explicación de los mismos.

Tabla 5.2. Elementos Básicos de Expresiones Regulares

Símbolo	Significado
{n}	Coincidir “n” veces
<x>	Coincidir elemento “x”
?	Coincidir 0 o 1 vez
|	Coincidir el elemento anterior o el siguiente
+	Coincidir 1 o más veces.
*	Coincidir 0 o más veces.

Existen muchos más elementos para la construcción de expresiones regulares pero los mostrados en la Tabla 5.2 son los esenciales que se utilizaron en la construcción de la gramática que utiliza NLRARE para la revisión de completitud de los requerimientos.

La gramática de NLARE se esperar vaya mejorando y siendo más robusta conforme se realizan más experimentos pero después de los ya realizados se obtuvo como resultante la siguiente gramática:

```
"""
    Function:{<MD>?<VB><IN>?<DT>?(<NN>|<NNP>|<NNS>)+}
    Detail:{<IN><DT>?(<NN>|<NNP>)+}
    Actor:{<DT><JJ>*<NN>+}
"""
```

Con esta gramatica entonces se pretende determinar si se encuentran los 3 elementos principales, *actor*, *función* y *detalle* que se definieron como parte de un requerimiento completo, expresado formalmente como:

```
R = {{A}, {F}, {D}}
```

Para tener una idea más clara de los elementos que forman a estas expresiones regulares presentamos entonces la siguiente tabla con las descripciones de los elementos Part-of-speech que forman parte de la gramática.

Tabla 5.3. Elementos Part-of-speech utilizados en la gramática NLARE.

Etiqueta NLTK	Descripción Original	Descripción Español
MD	Modal	Modal
VB	Verb	Verbo
IN	Preposition	Preposición

DT	Determiner	Determinante
NN	Noun	Sustantivo
NNP	Proper Nouns	Sustantivos Propios
NNS	Plurar Nouns	Sustantivos plurales
JJ	Adjective	Adjetivo

Una vez desarrollada esta gramática es necesario pasar cada requerimiento para completar la evaluación, esta se realiza por medio de una RegeX Parser o en español un equivalente podría ser “evaluador de expresiones regulares”.

Finalmente, todos los resultados recopilados durante la evaluación es agrupada y se presenta un resumen de la evaluación para cada requerimiento como se puede observar en la siguiente tabla.

Tabla 5.4. Información individual de cada requerimiento

Etiqueta	Detalle
Processing Time	Tiempo de procesamiento requerido en evaluar
FR<n>	Enumera el requerimiento e imprime su contenido
PoST	Representacion Part-of-speech
RegX	Resultado de la evaluación gramatical
<Atomic Test Result>	Resultado de la prueba de atomicidad
<Ambiguous Test Result>	Resultado de la prueba de ambigüedad
<Completness Test Result>	Resultado de la prueba de completitud

Posteriormente cuando se han revisado todos los requerimientos y se ha impreso su información correspondiente definida en la Tabla 5.4, el sistema procede a mostrar un balance general de lo que ocurrió en la revisión con totales sobre la misma, esta información se puede observar a detalle en la siguiente tabla.

Tabla 5.5. Información General resultante de la evaluación

Etiqueta	Detalle
Total Runtime	Tiempo total requerido en la evaluación
Total Loading Time	Tiempo invertido en cargar información a memoria
Total Requeriments Processed	Número total de requerimientos evaluados
Total Good Requirements	Número total de requerimientos correctos
Total Wrong Requirements	Número total de requerimientos incorrectos
-not atomic requirements	Número total de requerimientos no atómicos
-ambiguous requirements	Número total de requerimientos ambiguos
-incomplete requirements	Número total de requerimientos incompletos
List of good requirements	Desglose de requerimientos correctos
List of not-atomic requirements	Desglose de requerimientos no atómicos
List of ambiguous requirements	Desglose de requerimientos no ambiguous
List of incomplete requirements	Desglose de requerimientos incompletos

Con la información mostrada en la tabla 10, entonces finaliza la ejecución de NLARE y la evaluación de los requerimientos, se espera que esta herramienta sirva como apoyo a ingenieros de requerimientos para detectar más rápidamente problemas, y evitar el hecho de que hasta el mejor ingeniero puede equivocarse, una vez concluida esta evaluación se corregirían los requerimientos que así necesitaran y se correría una nueva prueba esperando ahora tener cero defectos o bien ciclar el proceso hasta tener una certeza de que todos los requerimientos están de cierta manera libres de defectos al menos en el contexto que se plantea en esta investigación.

5.3 Entradas y Salidas de la Arquitectura

A continuación, se describirá a manera general las entradas y salidas que tiene cada sección de la arquitectura. Para un enfoque más técnico se puede consultar el Capitulo 6. Programación de la Solución.

Como se vio anteriormente en la Figura 3. El flujo de la arquitectura inicia en la etapa denominada como *"NLP Data Loader",* y termina con la evaluación final de los requerimientos, un ejemplo de cómo se va transformando la información por cada etapa se muestra a continuación:

Etapa #1: NLP Data Loader
Etapa técnica transparente al usuario, véase Capitulo 6.

Etapa #2: Requirements Reader
Entrada:
```
The system will check if password match user in database
```
Salida:
```
[The system will check if password match user in
database]
```

Etapa #3:Tokenizer
Entrada:
```
[The system will check if password match user in
database]
```
Salida:
```
['The', 'system', 'will', 'check', 'if', 'password',
'match', 'user', 'in', 'database']
```

Etapa #4: Spellchecker
Etapa técnica transparente al usuario, véase Capitulo 6.

Etapa 5: Requirements Evaluator
Paso #1: Part-of-speech
Entrada:
```
['The', 'system', 'will', 'check', 'if', 'password',
'match', 'user', 'in', 'database']
```
Salida:
```
[('The', 'DT'), ('system', 'NN'), ('will', 'MD'),
('check', 'VB'), ('if', 'IN'), ('password', 'NN'),
('match', 'NN'), ('user', 'NN'), ('in', 'IN'),
('database', 'NN')]
```

Paso #2: RegeX
Entrada:
```
[('The', 'DT'), ('system', 'NN'), ('will', 'MD'),
('check', 'VB'), ('if', 'IN'), ('password', 'NN'),
('match', 'NN'), ('user', 'NN'), ('in', 'IN'),
('database', 'NN')]
```
Salida:
```
(S(Actor The/DT system/NN)
(Function will/MD check/VB if/IN password/NN match/NN
user/NN)
(Detail in/IN database/NN))
```

5.4 Evaluación de Ambientes de Desarrollo

Un paso muy importante que requiere bastante consideración en el desarrollo de esta investigación fue evaluar distintos ambientes de desarrollo, y las capacidades de los mismos para determinar según los objetivos propuestos, cual podría ser la opción más viable para un desarrollo exitoso.

En la siguiente tabla mostramos un listado de los que de acuerdo a nuestra investigación preliminar resultaron ser los frameworks de desarrollo más completos para cubrir las necesidades que estábamos buscando.

Tabla 5.6. Frameworks de desarrollo para NLP

Nombre	Lenguaje	Licencia	Creadores
Apertium	C++, Java	GPL	Varios
Delph-in	Lisp, C	LGPL	Grupo HPSG
Distinguo	C++	Comercial	Ultralingua
GATE	Java	LGPL	Comunidad GATE
LBJ	Java	BSD	UIUC
LinguaStream	Java	Investigativa	CAEN
Mallet	Java	CPL	UMass
MARF	Java	BSD	Grupo MARF
MontyLingua	Python, Java	Investigativa	MIT
NLTK	Python	Apache	Grupo NLTK
NooJ	.NET	Investigativa	UFC

OpenNLP	Java	Apache	Comunidad NLP
UIMA	Java, C++	Apache	Apache

De acuerdo a las características buscadas, existen algunos frameworks que si bien son importantes y merecen un reconocimiento considerable, no necesariamente son los adecuados para cumplir los objetivos previstos, a continuación hacemos un recuento de todos los frameworks evaluados

Apertium

Existen dos principales limitantes que encontramos en este framework, el primero de ellos y el más importante es que esta mucho muy enfocado al tema de traducción de lenguajes y si bien el framework es de codigo abierto, involucraría mucho trabajo adaptarlo para cumplir los objetivos buscados. Otro detalle importante también es que solo cuenta con soporte para sistemas operativos compatibles con el estándar POSIX lo que implica mucha problemática de compatibilidad al intentar integrarlo con Windows.

Delph-in

Este framework de desarrollo parece tener considerable potencial sin embargo, la documentación del mismo no está tan bien desarrollada como en otros frameworks, y por tanto no detallan con facilidad el cómo iniciar un desarrollo utilizando dicho ambiente.

Distinguo

Una de las principales características que mas llamo la atención de este ambiente es que su interfaz es para uso con C++, sin embargo una limitante considerable es el hecho de que este es un ambiente propietario, aspecto que lo descalifica como una potencial opción ya que en esta investigación se plantea utilizar tecnologías libres.

GATE

Este ambiente de desarrollo fue ampliamente considerado, para empezar tiene una gran historia pues sus inicios datan desde 1995, por lo que a la fecha de 2012, estaríamos hablando de 17 años. El principal detalle es la forma de programación que presenta no es acorde a lo esperado, ya que se está buscando un ambiente minimalista y de más bajo nivel, así como también, el enfoque del mismo es más orientado a Extracción de información, lo cual no es necesariamente el punto central de esta investigación.

LBJ

Este framework no fue evaluado en su totalidad, sobre todo por una peculiaridad y es el hecho de que utiliza un lenguaje de programación propio. Esto implica que se gastarían considerables recursos de tiempo solo aprendiendo dicho lenguaje en comparación con otros frameworks que utilizan lenguajes ya ampliamente conocidos. Finalmente la salida resultante parece producirse en una traducción a la maquina virtual de JAVA, lo que no necesariamente implicaría un código muy optimo.

LinguaStream

Este framework de desarrollo presenta un gran potencial, sin embargo, el tiempo requerido para entenderlo debido a su limitada documentación en comparación con otros ambientes, y al hecho de que incluso en la página oficial se muestra una tendencia a trabajar más con francés, no se ha seleccionado como el indicado para esta investigación. Otro punto importante por el cual se ha decidido no utilizar este ambiente es que limita la libre distribución, a la fecha de este escrito, en su página oficial no es posible descargarlo sin antes previa autorización de los desarrolladores.

Mallet

Este ambiente de desarrollo tiene una presentación muy adecuada pero sufre el mismo problema que se ha vuelto común entre diferentes frameworks, y es el no tener una suficiente documentación que permita alcanzar los logros buscados con la suficiente rapidez. Adicional a esto el framework esta más enfocado al área de aprendizaje de maquinas.

MARF

Este ambiente de desarrollo parece ser una buena opción para futuras versiones de la solución propuesta ya que manejan el área de reconocimiento de audio, una funcionalidad que sería de gran ayuda al proyecto propuesto, sin embargo para una primera instancia en donde el procesamiento de texto es la primordial función, no se considera que sea apto este ambiente, adicional a esto, aunque la sintaxis y funciones tienen una excelente documentación, otras áreas como tutoriales y ejemplos no están a la par con otros ambientes.

MontyLingua

Este framework de desarrollo es uno de los que mejor se podrían utilizar en esta investigación, sin embargo presenta algunas limitantes que lo descalifican como su potencial uso en este proyecto. Primeramente está muy enfocado al idioma inglés sin oportunidad aparente de una fácil transición a otro lenguaje puse no hay fase de entrenamiento del lenguaje. Adicionalmente la documentación no está a la par con otros ambientes e incluso está bastante limitada, la distribución oficial está enfocada para Windows y aunque éste se puede correr en otros sistemas, es necesario ajustes manuales lo que también lo deja en desventaja frente otras opciones.

NLTK

Sin duda uno de los mejores ambientes de desarrollo para procesamiento de lenguaje natural, la documentación es fantástica, incluso se puede contar con libros físicos publicados con extensos tutoriales, el ambiente utiliza el lenguaje Python que es mas practico de manejar que Java y está disponible en múltiples plataformas, la cantidad de funciones de procesamiento de este ambiente son generales al contrario de otros muy específicos para una tarea.

NooJ

Este ambiente utiliza la plataforma de .NET la cual si bien es una excelente plataforma tiene la peculiaridad de presentar problemas al intentarse ejecutar en ambientes distintos a Windows. A pesar de sus buenas características, uno de los ideales es que se pueda ejecutar el programa en múltiples plataformas, lo que no nos permite usar este ambiente.

OpenNLP

Este ambiente de desarrollo es de lo mejor que hay disponible, no solo en su presentación, sino también funciones y soporte, su documentación es muy buena. Incluso su licencia es buena ya que es Apache 2.0, realmente no hay mucho más que pedirle a este ambiente dadas las necesidades de esta investigación, sin embargo es necesario continuar evaluando las demás posibilidades para determinar la mejor opción disponible.

UIMA

Es uno de los mejores ambientes disponibles, y tal vez el mejor documentado, realmente cubre a gran detalle el cómo trabajar con él, sin embargo su enfoque es mas el de extracción de información que tareas más generales del área de procesamiento de lenguaje natural, sin embargo, para lo que está enfocado, es de lo mejor que hay disponible en este momento.

5.5 Ambiente Seleccionado: NLTK

Después de la evaluación de 13 distintos frameworks de desarrollo para procesamiento de lenguaje natural, se decidió utilizar NLTK como la plataforma de desarrollo por distintos puntos a favor que se comentan a continuación.

El ambiente de desarrollo NLTK fue iniciado en 2001, si bien no es de los más antiguos, ha tenido una vida considerablemente activa lo que a la fecha suma más de 11 años de trabajo que se ven reflejados en una excelente documentación.

El hecho de utilizar la Licencia Apache en su versión 2.0 también fue una motivación, esto debido a que algunos de los frameworks evaluados utilizaban políticas mucho menos abiertas en donde incluso descargar el framework presentaba problematicas.

El lenguaje Python es ampliamente utilizado y está disponible en numerosas plataformas, y para esta investigación el uso de Python proporciona un uso más ágil y rápido que Java, factor que incluyo en la decisión final.

5.6 Ambiente de Programación

Es importante definir las condiciones bajo las cuales el desarrollo de esta herramienta se está llevando a cabo, para lo cual mostramos a continuación las especificaciones de hardware y software del ambiente para que pueda servir de referencia para posibles trabajos que se deriven de esta herramienta.

Características del Equipo

Equipo: Laptop Acer 5750-9292
Procesador: Intel Core i7 2630QM 2.0 GHz (Turbo a 2.9 Ghz)
Memoria RAM: 4GB DDR3
Pantalla: 15.6” a 1366x768
Video: Intel HD Graphics 3000
Almacenamiento: Vertex 3 SSD 120GB
Sistema Operativo: Kubuntu 12.04 AMD 64 Kernel 3.2.0-26.

A continuación definiremos entonces aspectos de software que fueron necesarios completar antes de poder iniciar este desarrollo y que son de vital importancia pues también son requeridos para ejecutar la solución terminada.

Python

Lo primero a destacar aquí es la necesidad de tener un ambiente Python para el desarrollo, este lenguaje de programación es de uso general de alto nivel (Kuhlman, 2009) y uno de los principales objetivos de este lenguaje es que su código sea muy fácil de leer. Además este lenguaje facilita labores de programación comunes gracias a su gran librería de funciones (Python web, 2012).

Python es un lenguaje que soporta múltiples paradigmas de programación principalmente, mas no limitado a orientado a objetos, imperativo y en menor medida funcional y es un lenguaje dinámico lo que implica que puede ser ejecutado sin necesidad de un proceso de compilación pues sus funciones se procesan en tiempo de ejecución, sin embargo también es posible generar ejecutables independientes utilizando distintos software sin embargo ese proceso no está contemplado en esta investigación.

Es importante establecer que Python ha sufrido considerables cambios de la versión 2.x a la 3.x en donde la incompatibilidad es el tema principal, para este desarrollo y como se ha comentado en capítulos anteriores se ha escogido la librería NLTK la cual solo funciona de momento con Python 2.5 al 2.7 por lo cual es importante tener esta especifica versión instalada.

Como se comento anteriormente el ambiente de desarrollo utilizado es Linux y para la distribución utilizada ya se encuentra instalado Python por default por lo que no es requerida mayor intervención del usuario en este aspecto, sin embargo para ambientes Windows es muy seguro que no se cuente con Python por lo que se necesitara instalarlo, por lo que quedaría de la siguiente forma:

Python en Unix

Se recomienda utilizar la versión 2.7.3, en ambientes basados en Debian se puede verificar la versión instalada con “python -V”

Python en Windows

Basados en experiencias con otras dependencias necesarias recomendamos para usuarios de esta plataforma permanecer en la versión 2.6.6, misma que puede encontrarse en la siguiente dirección http://www.python.org/getit/releases/2.6.6/

NLTK

La siguiente dependencia y de igual importancia seria entonces instalar el ambiente de desarrollo de procesamiento de lenguaje natural el cual nos proporcionara todo un conjunto de librerías así como también todo un conjunto de “corpoa” que se utilizara para ayudar al programa a entender el lenguaje para el cual está programado.

En este caso a diferencia de la instalación de Python, tanto ambientes Unix como Windows estarían utilizando la misma versión de la librearía que sería la 2.0b9, versiones posteriores a estas se espera que sigan

funcionando sin embargo no hay garantía de que esto ocurra pues no ha sido probado.

Para la instalación quedaría entonces de la siguiente manera:

NLTK en Unix

Para ambientes basados en *Debian* únicamente es necesario abrir la terminal e instalar con la siguiente instrucción “sudo apt-get install python-nltk”, si se quiere verificar la versión que se tiene instalada se puede hacer con “apt-cache policy python-nltk” que en este ambiente de desarrollo arroja:
python-nltk:

```
  Installed: 2.0~b9-0ubuntu3
  Candidate: 2.0~b9-0ubuntu3
  Version table:
 *** 2.0~b9-0ubuntu3 0
        500 http://us.archive.ubuntu.com/ubuntu/ precise/universe amd64
Packages
        100 /var/lib/dpkg/status
```

NLTK en Windows

Para la plataforma de Windows la instalación es ligeramente más extensa pues no se tiene un repositorio como en Unix, por lo que es necesario descargar e instalar el paquete que se puede encontrar en la siguiente ruta: http://nltk.googlecode.com/files/nltk-2.0b9.win32.msi

NumPy

Una extensión que es necesaria para NLARE también es NumPy la cual se encarga de agregar soporte para grandes arreglos y matrices de múltiples dimensiones así como también un gran grupo de funciones matemáticas para realizar operaciones sobre estos elementos. Esta librería está basada en la originalmente creada por Jim Hugunin la cual tenía como titulo Numeric, misma que fue mejorada por Travis Oliphant desde 2005 y es lo que conocemos hoy como NumPy.

En este caso también se maneja la misma versión de NumPy 1.5.1 sin importar la plataforma, y la instalación se realiza como se indica a continuación:

Numpy en Unix

Basta con abrir la terminal y ejecutar el siguiente comando “sudo apt-get install python-numpy” para tener la versión más actual de NumPy, se recomienda la 1.5.1 o bien se puede optar hasta por la 1.6.1 la cual se ha confirmado que funciona perfectamente, posteriores versiones se desconoce su comportamiento.

Numpy en Windows

Como hemos estado manejando en Windows es necesario descargar y manualmente instalar esta extensión, misma que puede ser encontrada en el siguiente link:
http://downloads.sourceforge.net/project/numpy/NumPy/1.5.1/numpy-1.5.1-win32-superpack-python2.6.exe?r=&ts=1340672868&use_mirror=voxel

PyEnchant

Esta a una librería esencial si se está trabajando con idioma inglés, con PyEnchant podemos enlazar Python con el excelente evaluador ortográfico de la librería Enchant que fue diseñada para ser utilizara por el procesador de palabras AbiWord, y gracias a esta librería entonces podemos incluir el avance tecnológico de Enchant en nuestra aplicación de evaluación de lenguaje natural.

PyEnchant no ha recibido actualizaciones desde Diciembre 2010, así que la versión más nueva al momento de esta redaccion es la que se ha probado en ambas plataformas, la cual es 1.6.5 y se instala como se muestra a continuación:

PyEnchant en Unix

La forma más sencilla de instalarlo en ambientes basados en *Debian* sería entonces vía la terminal con una simple instrucción como “sudo apt-get install python-enchant”

PyEnchant en Windows

Para esta plataforma nuevamente se necesario extender un poco más el proceso, el cual iniciaría primeramente descargando la aplicación la cual puede encontrarse en la ruta: http://pypi.python.org/packages/any/p/pyenchant/pyenchant-1.6.5.win32.exe

Finalmente como último requisito de ambiente de desarrollo y ejecución tenemos PyYAML del cual daremos una breve descripción a continuación:

PyYAML

Lo que se consigue con esta librería es una interfaz via Python para poder manejar YAML el cual es un formato para serializar información de una manera entendible para un humano, de ahí su nombre recursivo derivado de inglés “YAML Aint Markup Language”, con lo que los desarrolladores intentan hacer muy marcada la diferencia entre el enfoque de información dado por YAML y otros lenguajes que son mas de documentos como lo es XML.

Para la instalación, se recomienda la versión 3.09 para Windows y 3.10 para Unix.

PyYAML en Unix

Para instalar la versión más nueva, basta con abrir la terminal y en sistemas basados en debían realizar la operación “sudo apt-get install python-yaml” para obtener la versión más nueva, la ultima probada ha sido 3.10, que al momento de este escrito es la más nueva.

PyYAML en Windows

En esta plataforma la versión 3.09 compilada para Python 2.6 es la ultima que ha sido satisfactoriamente probada, la más reciente puede

funcionar sin embargo, no se garantiza la efectividad de la misma, para obtener la versión recomendada en Windows es necesario descargarla de la siguiente ubicación: http://pyyaml.org/download/pyyaml/PyYAML-3.09.win32-py2.6.exe

Con esto finalizamos la sección de ambiente de desarrollo, es importante recordar que como se menciono al principio, esto es necesario no solo para la programación sino para la ejecución de la aplicación.

Una última nota práctica para los usuarios de sistemas Unix basados en Debian, es de utilidad mencionar que una sola línea de instrucción en la terminal podría preparar todo el ambiente necesario, ya que las acciones pueden ser concatenadas como sigue:
“sudo apt-get install python-nltk python-numpy python-enchant python-yaml”

CAPÍTULO 6

PROGRAMACIÓN DE LA SOLUCIÓN

6 Programación de la Solución

Este capítulo tendrá un enfoque completo a la programación de NLARE, cómo se realizó en términos de código, y servir como una guía en programación de variantes de NLARE.

Es importante tener un conocimiento de programación orientada a objetos, así como de conocimientos básicos del lenguaje de programación Python 2, se espera que este capítulo sirva también para posibles versiones de NLARE desarrolladas para Python 3.

NLARE es una aplicación sin interfaz gráfica de usuario y su interacción es únicamente vía parámetros de línea de comandos al iniciar la aplicación.

El uso de NLARE se describe de manera más extensa en la documentación del programa, pero para fines informativos, se provee una versión resumida en esta tesis. En términos generales la aplicación se debe iniciar con 3 parámetros de la siguiente forma.

```
nlare.pyc <datos> <palabras_reservadas> <requerimientos>
```

O bien un ejemplo completo seria:

```
nlare.pyc /uabc/nltk_data/ mywords.txt rekerimientos.txt
```

6.1 Importación de Módulos

La primera sección de código corresponde entonces a la importación de módulos de Python, esto se realiza mediante las siguientes instrucciones:

import <modulo>

Con esta instrucción importamos modulo a nuestro código, aunque las definiciones del mismo no son importadas y deben ser accedidas explícitamente.

From <modulo> import <funcion>

Con esta instrucción se pueden definición funciones explícitamente para poder ser utilizadas directamente en el código sin necesidad de hacer referencia a que modulo pertenecen.

En el caso de NLARE es necesario importar los módulos y funciones mostrados en la siguiente tabla.

Tabla 6.1. Módulos y funciones importados a NLARE

Modulo	Función
__future__	division
nltk.corpus	wordnet
time	clock
time	time
nltk	
re	
pprint	

fileinput	
enchant	
sys	

A continuación explicamos la utilidad de los módulos y funciones importados.

División:

Existe una confusión son el operador de división “/” y el resultado que se espera de este, por ejemplo 1 / 2 da como resultado 0, pues es división de enteros. En futuras versiones de Python se espera que el operador “/“ siempre de el resultado real sin importar el tipo de dato, y utilizar “//” si realmente se quiere una división de enteros, para habilitar el comportamiento planeado a futuro entonces utilizamos el modulo __future__.

Wordnet:

Con esto tenemos acceso al “corpus reader” de NLTK con el que podemos obtener distinta información de elementos léxicos o palabras. En el caso de NLARE se utiliza en forma de *DEBUG* para analizar palabras mediante la función *synset(),* que se agrega al conjunto de funciones disponibles, y con las cuales podemos obtener información léxica de cualquier palabra en cuestión que se esté analizando dentro del requerimiento, esto puede ayudar a un mejor desarrollo de las expresiones regulares y sobre todo, al análisis del lenguaje si se quiere extender el uso a distintos lenguajes y no solo el Inglés.

Clock y time:

La definición de estas funciones ambas pertenecientes al modulo "time" no influyen directamente con la funcionalidad de NLARE, pero proporcionan un apoyo a la usabilidad del mismo, y son responsables de las pruebas de rendimientos que se realizan en NLARE para calcular el tiempo de procesamiento que tomo la evaluación de los requerimientos.

De acuerdo a la documentación de Python (Python Docs) la función clock() es la recomendada para hacer "*benchmark*" pues es la encargada de regresar el tiempo de procesamiento, mismo que está definido en la función de C que lleva el mismo nombre.

En sistemas Windows, esta función se comporta un poco diferente, por esto hemos definido también la función time en NLARE para posibles modificaciones en otras plataformas.

Nltk:

La importación de este modulo en el contexto de NLARE es muy obvia, pues es, como se ha comentado a lo largo de esta tesis, NLTK el framework de desarrollo bajo el cual está construida la propuesta de solución.

Entre las funciones más importantes que se utilizan en NLARE están las que mostramos en la siguiente tabla:

Tabla 6.2 Conjunto de principales funciones utilizadas del modulo NLTK

Función	Detalle
data.path.append()	Agrega una nueva ruta para buscar las fuentes de información NLTK
nltk.data.load()	Carga información o recursos para asignarlos a un objeto
nltk.word_tokenize()	Con esto pasamos de una oración a un arreglo para procesarlo
nltk.PorterStemmer()	(Solo Debug) Permite llegar a la raiz léxica de una palabra
nltk.LancasterStemmer()	(Solo Debug) Permite llegar a la raiz léxica de una palabra
nltk.pos_tag()	Realiza todo el etiquetamiento Part-of-speech
nltk.help.upenn_tagset()	(Solo Debug) Proporciona información sobre la forma en que una palabra fue etiquetada.
nltk.FreqDist()	(Solo Debug) Muestra la frecuencia de distribución para una tupla palabra-etiqueta.
nltk.RegexpParser()	Este es un evaluador de gramáticas basadas en expresiones regulares

Existen más referencias al modulo NLTK, sin embargo las anteriores muestran en esencia el núcleo de NLARE.

Re:

Este módulo de corto nombre “re” esta derivado del inglés “regular expression”, es importante tener definido este módulo al momento de trabajar con expresiones regulares, ya que de acuerdo a la documentación de Python (Python Docs 2) el carácter “\” en expresiones regulares es utilizado para invocar una forma especial, este mismo uso se consigue con el mismo carácter (“\”) en Python por lo que se genera entonces un conflicto. Para corregir este problema se puede utilizar la notación “raw” de Python para indicar que una cadena es literalmente los elementos que la contienen.

Pprint:

Derivado del inglés “pretty printer”, este modulo nos permite imprimir estructuras de información, que puedan ser evaluadas como entradas al intérprete de Python. Es más que nada utilizada en el modo debug de NLARE para poder realizar evaluaciones más prácticamente en tiempo de ejecución.

Fileinput:

Desde su nombre en inglés se puede intuir el manejo que se le da a este modulo, y es precisamente para la entrada de archivos a NLARE. En este caso el archivo a procesar es el conjunto de requerimientos, cuando solo se desea leer un archivo es recomendable la función *open()* la cual no es dependiente de este modulo sin embargo NLARE no solo necesita leerlo sino realizar procesamiento interno de evaluación mientras lo recorre, es por esto que este modulo entra en juego y permite la realización de ese tipo de ciclo.

Enchant:

El nombre de este modulo esta derivado de la librería que lleva el mismo nombre Enchan, y es utilizada para la revisión ortográfica del texto en los requerimientos.

Sys:

Con éste modulo accedemos a funciones básicas del sistema, en el caso en particular de NLARE, nos interesa tener acceso a la información de los parámetros con que éste fue ejecutado para poder configurar las 3 condiciones con las que trabaja la aplicación, que son:

ruta de datos: que indica donde se encuentra la información de entrenamiento y utilerías del ambiente NLTK.
palabras reservadas: que definen el grupo de palabras propias del proyecto que deben agregarse al vocabulario del mismo.
requerimientos: que indica el archivo a procesar.

6.2 Inicialización del Programa

Lo primero que se ejecuta en NLARE es el sistema de evaluación de rendimiento, en este caso se lanza la instrucción para determinar el tiempo al cual inicia la aplicación, misma que se inicia a calcular posterior a cargar los módulos.

```
tiempototal = clock()
```

Una vez marcado ese tiempo en la variable se prosigue a recopilar los

argumentos con los que fue lanzada la aplicación, los cuales se pueden encontrar en el arreglo `sys.argv` y se utilizan de la siguiente forma.

`sys.argv(1)`: se asigna a la ruta de fuentes de NLTK

`sys.argv(2)`: se agrega a Enchant como grupo de palabras adicionales al diccionario

`sys.argv(3)`: se utiliza con open() para crear un objetivo tipo archivo en modo lectura “r”

Posteriormente se inicializan las variables de estado que serán utilizadas a lo largo del programa.

6.3 Procesamiento de Requerimientos

Esta sección es la que contempla casi todo el código generado en NLARE, en este caso se necesita un ciclo para recorrer cada uno de los requerimientos a evaluar, el ciclo es de la siguiente forma:

```
for line in f:
```

Una vez dentro de este ciclo, se procede a una nueva prueba de rendimiento para evaluar cuanto tomo la evaluación de cada uno de los requerimientos. Para eso volvemos a hacer uso de clock() como se muestra a continuación:

```
start = clock()
```

Posteriormente, para limpiar los requerimientos de caracteres en blanco al principio o final de estos, los pasamos por un filtro de la siguiente forma:

```
s = line.strip()
```

Para continuar, se realiza la tokenización de sentencias y se evalúa que únicamente haya 1 sentencia por requerimiento de la siguiente forma:

```
sents = sent_tokenizer.tokenize(s)
for cada in sents:
    ...
```

Continuando con el procesamiento, se procede a una nueva tokenización pero en esta ocasión por palabras, mismas que son revisadas con Enchant para detectar errores ortográficos, esto se realiza como sigue:

```
tokens = nltk.word_tokenize(s)
for feliz in tokens:
    if not (d.check(feliz)):
        ...
```

Una vez realizado el proceso anterior, se procede a determinar entonces a que Part-of-speech pertenece cada una de las palabras tokenizadas según la posición y contexto en el que esta se encuentra, para eso utilizamos:

```
etiketas = nltk.pos_tag(tokens)
```

Una vez realizada la tokenización se procede a realizar la evaluación de si están o no incluidos elementos que puedan causar conjunción y ambigüedad en el requerimiento, estos elementos ya fueron descritos con anterioridad, y lo conseguimos de la siguiente forma:

```
for info in etiketas:
     if (info(1) == "CC"):
          ... #Esto es para conjunción
     if (info(1) == "FW" ... "JJ" ... "RB" ... ...):
          ...#Esto es para ambigüedad
```

Una vez habiendo realizado esta validación se procede entonces a la definición de la expresión regular que será utilizada como gramática para evaluar la completitud del requerimiento, misma que se realiza de la siguiente forma:

```
grammar =
"""
     Function:{<MD>?<VB><IN>?<DT>?(<NN>|<NNP>|<NNS>)+}
     Detail:{<IN><DT>?(<NN>|<NNP>)+}
     Actor:{<DT><JJ>*<NN>+}
"""
```

Una vez definida la gramática se procede a enviarlo al RegeX Parser para su evaluación de la siguiente forma:

```
  cp = nltk.RegexpParser(grammar)
  result = cp.parse(etiketas)
```

Una vez completada esta evaluación se procede nuevamente a la toma del tiempo para determinar el rendimiento de cada requerimiento, esto lo hacemos de la siguiente manera:

```
elapsed = (clock() - start)
```

Posterior a esto se imprime todo el resultado de la evaluación de cada requerimiento y así se continúa hasta completar cada uno de ellos. Al final de la evaluación entonces se cierra la prueba de rendimiento general de todo el programa de la siguiente forma.

```
ttotal = (clock() - tiempototal)
```

Y se procede a mostrar el resumen que ya hemos comentado anteriormente en donde se imprime el resultado de las variables de estado que se fueron generando durante la ejecución con la información de resultados generales de la evaluación.

En resumen de este capitulo podemos concluir que el uso de NLTK como el framework de desarrollo realmente proporciona un ambiente que facilita la programación de la solución. Una de las instrucciones que más facilita el trabajo es la de *RegexpParser* que nos permite hacer toda la evaluación contra la gramatia en una sola línea de código.

Por otra parte, el uso de software libre como lo es python y nltk, permite distribuir el trabajo resultante sin problemas de licencias, lo que facilito en gran medida realizar pruebas de gran cantidad de requerimientos con diferentes personas y equipos.

CAPÍTULO 7

PRUEBAS Y RESULTADOS

7 Pruebas y Resultados

En este capítulo se describirán las pruebas realizadas así como también un análisis de los resultados obtenidos hasta el momento con la versión 1.0b de NLARE.

Se realizaron dos grupos de experimentos:

- Industria
- Educación

Industria:

En esta sección se trabajó con una empresa local en la que se hizo un análisis de los requerimientos de un sistema de hardware para poder determinar oportunidades de mejora en el proceso de requerimientos.

Educación:

Estos experimentos están basados en la Materia Programación Orientada Objetos Avanzada que se imparte en la Universidad Autónoma de Baja California, para la cual se solicitan requerimientos en los cuales se plantea basar el proyecto final de la materia.

7.1 Caso de estudio No. 1: Pruebas Industria

Esta investigación fue realizada gracias a la empresa Argus Tecnologías en Tijuana en donde se estuvo trabajando con un proyecto titulado "*Desarrollo de un módulo electrónico de bajo costo para localización vehicular y telemetría de parámetros con memoria de almacenamiento y descarga de datos por medio del estándar WiFi 802.11*". Es importante mencionar que este proyecto se sale ligeramente de lo acostumbrado ya que es un sistema embebido sin embargo no deja de ser software y por lo tanto tener diferentes requerimientos que deben ser documentados y validados.

Los resultados aquí plasmados están basados en el proyecto anteriormente mencionado en donde se aplicó esta herramienta para realizar el caso de estudio en el desarrollo de un sistema embebido.

Para contextualizar mejor los resultados a continuación mostramos una breve descripción de lo que es un sistema embebido y las características del mismo que son importantes resaltar al evaluar los resultados posteriormente mostrados.

De acuerdo a Michael Barr (2007) y Heath Steve (2003), un sistema embebido es sistema de computación diseñado para realizar una o algunas pocas funciones dedicadas frecuentemente en un sistema de computación en tiempo real. Los sistemas embebidos se utilizan para usos muy diferentes a los usos generales a los que se suelen someter a las computadoras personales. En un sistema embebido la mayoría de los componentes se encuentran incluidos en la placa base. Por lo general los sistemas embebidos se pueden programar directamente en el lenguaje ensamblador

del microcontrolador incorporado sobre el mismo, o bien, utilizando algún compilador específico; suelen utilizarse lenguajes como C, C++ y hasta en algunos casos BASIC.

El proyecto a ser evaluado en cuestión es precisamente esto, un sistema embebido en donde se planea tener un sentado en tiempo real de la ubicación GPS del vehículo, todo esto sin la necesidad de una computadora como las conocemos tradicionalmente sino creando un dispositivo que funcione por sí solo, de igual manera este realiza relativamente pocas funciones lo que en teoría debería facilitar la especificación de sus requerimientos al tener un dominio más chico de posibilidades

NLARE fue utilizada para realización de pruebas automáticas a los requisitos funcionales del sistema a desarrollar para la empresa. Para probar los resultados de la herramienta, fueron comparados contra una evaluación manual con el objetivo de verificar la veracidad y eficiencia con que los resultados son obtenidos y determinar si es viable la utilización para proyectos más extensos.

Cabe mencionar que la herramienta desarrollada funciona únicamente para el idioma inglés, de tal forma que los requerimientos del sistema, aunque inicialmente redactados en español, han sido traducidos manualmente para evitar errores derivados de alguna traducción automática

7.1.1 Condiciones del Experimento

En este experimento, las actividades de evaluación de requerimientos se llevaron a cabo sin modificar el flujo del desarrollo, es decir, permitiendo que los errores de los requerimientos iníciales continuaran su flujo.

7.1.2 Fases del Experimento

En resumen el experimento se llevó a cabo siguiendo las etapas descritas a continuación:

1) Se redactaron requerimientos para el proyecto siguiendo estándares y metodologías propias de la empresa, esta forma de recolectar requerimientos es indiferente a la investigación, pues solo nos interesa el producto resultante que es el documento.
2) Una vez teniendo los requerimientos se prosigue a una traducción al idioma Inglés, esto de manera manual para evitar problemas por traducción.
3) Se revisaron manualmente los requerimientos entre diferentes personas para determinar en conjunto de qué problema (ambigüedad, conjunción, incompletitud) sufría cada uno.
4) Se utilizó la herramienta NLARE para evaluar automáticamente los requerimientos y determinar que tan eficiente y eficaz es su funcionamiento.
5) Se realizaron encuestas para determinar cuáles eran los principales problemas que los involucrados en el proyecto enfrentaron.
6) Finalmente se creó una tabla relación entre los defectos de los requerimientos y los problemas encontrados

7.1.3 Materiales y Herramientas

Para este experimento se utilizaron las siguientes herramientas:
LibreOffice Writer: Editor de textos para redacción de documentos
Google Translate: Apoyo en dudas para la traducción de los requerimientos
NLARE: Herramienta de análisis automático para requerimientos
LibreOffice Calc: Generación de gráficas

7.1.4 Objetivos del experimento

Los principales objetivos en este experimento fueron:

1) Determinar cómo se compara la evaluación manual de requerimiento con el resultado objetivo automáticamente por NLARE
2) Establecer una relación entre las problemáticas encontradas en el desarrollo y los problemas encontrados en los requerimientos.

7.1.5 Requerimientos de Proyecto Industria

A continuación se muestra una recopilación de los requisitos funcionales de software del proyecto en su versión intacta en español y su traducción al idioma inglés. Es importante resaltar que esta traducción se realizó de manera manual y no incluye defectos generados por un traductor automático.

Primeramente mostramos el listado de requerimientos originalmente expresados en idioma español recopilados de los documentos finales del proyecto.

1. El sistema de Coordenada Móvil debe contar con la capacidad de leer los datos que se encuentran en la memoria del módulo, en caso de que esta sea insertada en una computadora personal
2. El sistema Coordenada Móvil podrá generar consultas y reportes de históricos, tiempos muertos y excesos de velocidad con la información contenida en la memoria Servidor
3. El servidor deberá contar con una nueva consola de recepción de ubicaciones especializada en tramas tipo Wi-Fi
4. La consola debe ser capaz de recibir una trama de datos del módulo
5. La consola debe ser capaz de interpretar y manipular cada una de las tramas enviadas por el módulo
6. La consola trabajará de manera asíncrona en el proceso de interpretación de cada trama, con el uso de hilos, para asegurar que toda la información que se recibe es procesada
7. La consola deberá almacenar las ubicaciones en bases de datos
8. El servidor deberá de contar con una nueva tabla para la configuración y almacenamiento de configuración de redes inalámbricas de todos los módulos
9. El servidor deberá de poder enviar configuraciones de redes inalámbricas a los módulos

Para ver un listado completo de los requerimientos véase sección 10.1

La aplicación desarrollada únicamente funciona con el idioma inglés. La traducción resultante se muestra a continuación, esta es la versión que fue utilizada como entrada para el programa. La documentación en español no tuvo más interacción con la aplicación.

1. The mobile coordinate system must have the ability to read the data in the memory module if this is inserted into a personal computer
2. The mobile coordinate system can generate queries and historic, downtime and speeding reports with the information contained in the server
3. The server must have a new console for location specializing receipts in Wi-Fi frames
4. The console must be able to receive a data frame from the module
5. The console must be able to interpret and manipulate each of the frames sent by the module
6. The console will work asynchronously in the process of interpretation of each plot with the use of threads to ensure that all information received is processed
7. The console will store the locations in databases
8. The server must to have a new table for storing configuration and wireless network configuration of all modules
9. The server must be able to send wireless network settings to the modules

Para una lista completa de los requerimientos véase sección 10.2

En la siguiente sección tendremos análisis de los resultados al ingresar los requerimientos previamente traducidos en la herramienta NLARE.

7.1.6 Resultados Prueba Industria

En la siguiente tabla se muestran los resultados de la evaluación que se realizó. En este caso, con el símbolo O se indican los resultados de la evaluación manual, mientras que con el símbolo K indicamos los resultados arrojados por la herramienta automática que coinciden con los calculados manualmente.

Tabla 7.1. Resultados de Pruebas a Industria

No. Req	No Atómico	Ambiguo	Incompleto	Correcto
1				O
2	OK			
3	O	OK	OK	
4		OK	O	
5	O	OK	OK	
6		OK		
7			OK	
8	OK	OK		
9		OK		
10		OK		
11		OK		
12		OK		
13	OK	OK		
14	OK		OK	
15		OK		

16	OK	OK		
17		OK		
18	OK	OK		
19			OK	
20		OK		
21		OK	OK	
22				OK
23	OK	OK		
24	OK	OK		
25	OK			
26				O
27	OK	OK	O	
28		OK	OK	
29			O	
30		OK	O	
31	OK		OK	
32		O	O	
33	OK		O	
34	OK		OK	
35	O			
36				O
37	OK	OK	O	
38		OK	OK	
39	OK		O	
40	OK			
Totales	O=19 / K=16	O=24 / K=23	O=18 / K=10	O=4 / K=1

Los resultados que la aplicación produce se muestran a manera de texto, el cual se incluye a continuación:

Total Runtime = 10.8 sec
Total Loading Time = 9.09 sec
Total Requirements processed = 40 @ 22.8 req/sec
Total Good requirements = 3 = 7.5 %
Total Wrong requirements = 37 = 92.5 %
-not atomic requirements = 17 = 42.5 %
-ambiguous requirements = 33 = 82.5 %
-incomplete requirements = 21 = 52.5 %

List of Good requirement:
FR22, FR32, FR35,
List of Not atomic requirements:
FR2, FR5, FR8, FR13, FR14, FR16, FR18, FR23, FR24, FR25, FR27, FR31, FR33, FR34, FR37, FR39, FR40,
List of Ambiguous requirements:
FR1, FR2, FR3, FR4, FR5, FR6, FR8, FR9, FR10, FR11, FR12, FR13, FR14, FR15, FR16, FR17, FR18, FR19, FR20, FR21, FR23, FR24, FR25, FR27, FR28, FR29, FR30, FR31, FR34, FR37, FR38, FR39, FR40,
List of Incomplete requirement: FR3, FR5, FR7, FR8, FR9, FR10, FR11, FR12, FR14, FR15, FR16, FR19, FR20, FR21, FR24, FR26, FR28, FR31, FR34, FR36, FR38,

A continuación mostramos gráficas de todas las pruebas realizadas y los resultados obtenidos.

Prueba de Veracidad

Lo más importante en esta herramienta es identificar que tan bien realiza su trabajo comparado con un humano con experiencia en el tema de requerimientos, para lo cual mostramos la siguiente gráfica con los errores detectados por un humano contra los detectados por la herramienta.

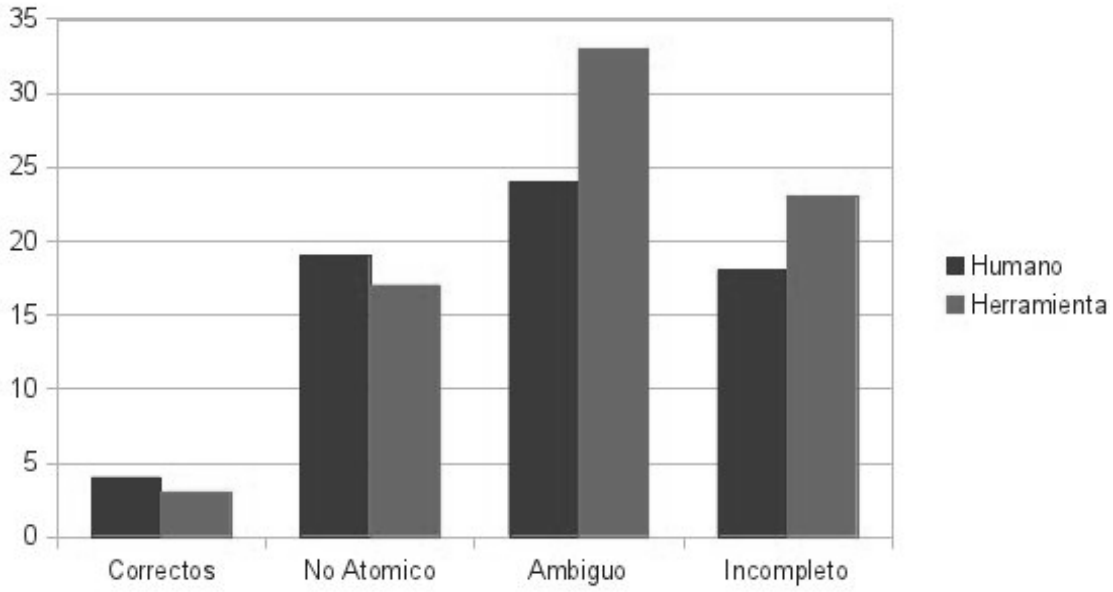

Figura 7.1. Prueba de Veracidad

La gráfica anterior nos indica que existe una buena aproximación entre los resultados obtenidos por la herramienta al ser comparados con la evaluación humana. Es evidente que de momento los resultados no son idénticos en ambas evaluaciones, sin embargo la “diferencia” no determina que uno sea mejor resultado que el otro, pues es posible también tener errores en la revisión humana.

La diferencia más grande se presenta al determinar la ambigüedad, pues la herramienta considera que hay otros 9 requerimientos más que deben ser catalogados como ambiguos, esto se debe principalmente a que la

herramienta es más estricta en su revisión y sus resultados son siempre los mismos, mientras que los resultados de la evaluación humana son susceptibles al revisor y muchas veces se tiende a tener un grado de gracia entre lo ambiguo, mientras que la herramienta es imparcial al evaluar este parámetro

Prueba de Similitud de Resultados

En esta prueba interesa saber en cuantos resultados la herramienta y la evaluación humana estuvieron de acuerdo con el problema detectado, para lo cual mostramos la siguiente gráfica.

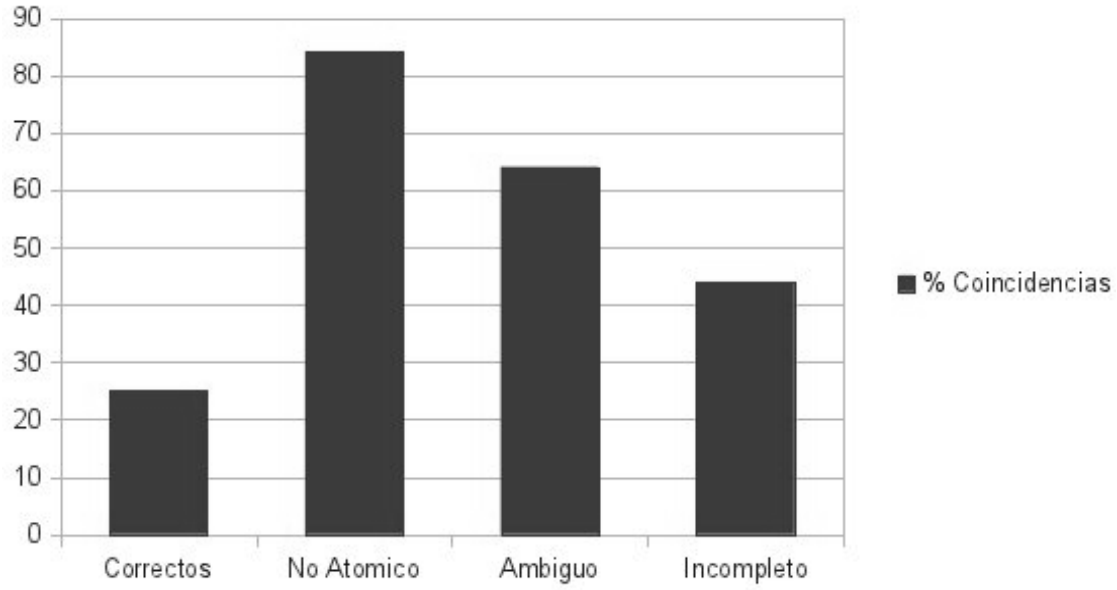

Figura 7.2. Prueba de Similitud de Resultados

En la gráfica se puede observar que el humano y la herramienta presentaron pocas diferencias en su evaluación salvo en la determinación de la correctitud en donde la herramienta fue mucho más estricta en cuanto a determinar qué requisitos estaban en su totalidad completos, pues solo

coincidieron en el 25% de los casos, mientras que el atributo de atomicidad fue el más efectivo con 84% de similitud en los resultados.

Prueba de Velocidad

Al evaluar requisitos funcionales de forma manual, no solo se corre el riesgo de error por el factor humano sino que es considerablemente más lenta la evaluación manual que lo que puede realizar una máquina promedio.

Para la siguiente prueba se tiene:
Humano: Ingeniero en Computación, con 5 años de experiencia en el manejo de requerimientos

Computadora:
Tipo: Laptop AMD Phenom II @ 1.8 Ghz
RAM: 4GB

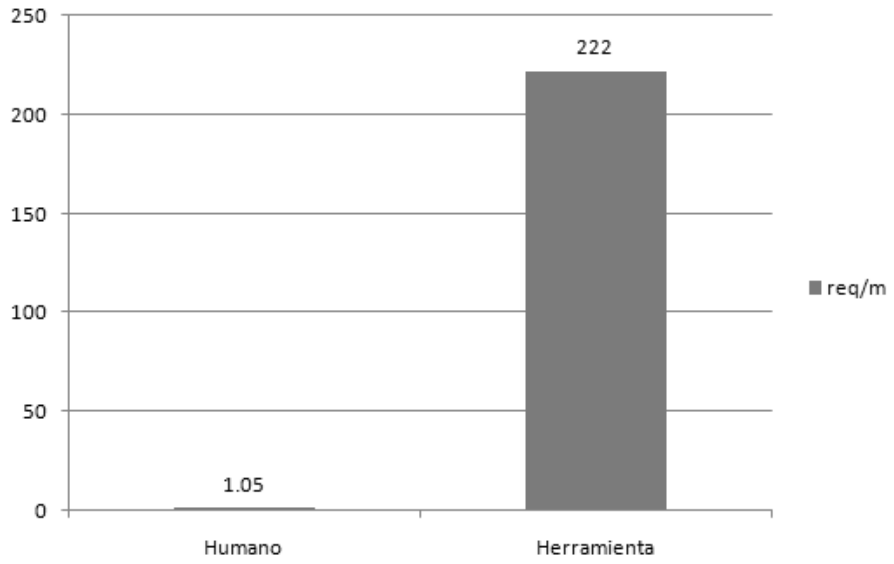

Figura 7.3. Prueba de Velocidad

En la gráfica se puede observar la enorme superioridad en cuanto a

velocidad que proporciona la herramienta sobre la evaluación manual.

El humano alcanzó una velocidad de 1.05 requerimientos por minuto, mientras que la aplicación alcanzo una velocidad en la computadora de prueba de 222 requerimientos por minuto, lo que implica una ganancia en velocidad considerablemente grande.

7.1.7 Conclusión pruebas Industria

El tema de requerimientos es un aspecto de la ingeniería de software que todavía tiene mucha oportunidad de investigación, sobre todo por la gran complejidad que representa el trabajar con Lenguaje Natural. Creemos que con este desarrollo se contribuye considerablemente para abrir la oportunidad de iniciar automatizaciones en esta etapa tan sensible del desarrollo de software. Y es un buen inicio para que se empiece a trabajar en el aseguramiento de la calidad desde el principio, tal como lo hace NASA, quien tiene unos de los más grandes adelantos en la materia.

Una ventaja de esta herramienta es que proporciona resultados muy rápidos que de otra forma dados los tiempos calendarizados por el proyecto no es posible agendar tanto tiempo para el análisis de requerimientos, pues es una tarea que requiere grandes esfuerzos y muchas veces no es reconocido pues no aporta en ese momento un avance tangible para el cliente, sin embargo, como hemos visto, los problemas que puede traer consigo iniciar no adecuadamente desde etapas tempranas hace importante dedicarle el suficiente análisis a estas etapas. Existe poca formalización y herramientas para ayudar a agilizar este proceso, afortunadamente con este trabajo de tesis esperamos apoyar en esta parte para así poder generar

software de calidad desde etapas tempranas sin obstruir el flujo rápido con el que se desea producir un sistema.

Gracias a esta investigación fue posible desarrollar una tabla en donde se intenta formalizar una relación entre los potenciales problemas que surgen en el desarrollo de software y los errores que presentan los requerimientos en el proyecto, la siguiente tabla muestra dichos resultados.

Tabla 7.2. Potenciales problemas derivados de los requerimientos

Problema	Causa
Casos de prueba no atómicos	No Atómico
Dificultades para estimar el costo del proyecto	No Atómico
Dificultad (imprecisión) para calendarizar los tiempos de desarrollo	No Atómico
Dificultad para establecer prioridades a las actividades	No Atómico
Retrasos por el tiempo de búsqueda de información faltante	Incompleto
Delegación de responsabilidades que pertenecen a otros roles	Incompleto
Se puede asumir información basada en un conocimiento incorrecto	Incompleto
La estimación de costos no concuerda con los esfuerzos requeridos	Incompleto
Esfuerzos pueden ser desperdiciados desarrollando cosas innecesarias	Ambiguo
Retrabajo es requerido para reparar las	Ambiguo

especificaciones mal interpretadas	
Retrasos para completar el proyecto debido a una mala interpretación de objetivos	Ambiguo
El trabajo resultante podría no ser lo inicialmente esperado (no correspondencia)	Ambiguo

En base a los resultados obtenidos se puede observar que el atributo de calidad que más impacto a los requisitos del proyecto fue el de Ambigüedad, y si evaluamos esto con la Tabla 7.2, en donde se hace una referencia a los potenciales problemas a surgir debido a requerimientos con este problema, tenemos:

- Esfuerzos pueden ser desperdiciados desarrollando cosas innecesarias.
- Retrabajo es requerido para reparar las especificaciones mal interpretadas.
- Retrasos para completar el proyecto debido a una mala interpretación de objetivos.
- El trabajo resultante podría no ser lo inicialmente esperado (no correspondencia).

Durante mi participación en el proyecto pude observar los problemas presentados, también potencialmente originados por el atributo de calidad de completitud que fue también de los que más afectó a los requerimientos:

- Delegación de responsabilidades que pertenecen a otros roles.
- Se puede asumir información basada en un conocimiento incorrecto.
- Retrabajo es requerido para reparar las especificaciones mal interpretadas.

- Retrasos para completar el proyecto debido a una mala interpretación de objetivos.

Se aplicaron encuestas para evaluación del proyecto y determinar los principales problemas que se presentaron según el punto de vista de los integrantes directos del proyecto. La encuesta aplicada se compone de 12 preguntas las cuales se derivaron de los potenciales problemas a ocurrir mostrados en la Tabla 7.2.

La encuesta realizada se puede consultar en el Apéndice 10.5

Los resultados de la encuesta aplicada se muestra en la siguiente tabla indicando:

Problema: Defecto asociado en los requerimientos
No. Preg.: Pregunta en cuestión según la lista anterior
E1 a E5: Participantes del proyecto
Freq: Cantidad de integrantes que afirmaron se presento el problema en cuestión
% Si: Porcentaje de integrantes que identificaron el problema (derivado columna Freq)
Severidad: Representa el valor matemático de con que tanta frecuencia se presento el problema, estos valores se determinan en Poco (0.01 a 0.25), Regularmente (0.26 a 0.50), Mucho (0.51 a 0.75) y Siempre (0.76 a 1.00).

Tabla 7.3. Resultados de la encuesta Industria

Problema	No. Preg	E1	E2	E3	E4	E5	Freq	% Si	Severidad
Incompletitud	1	1	1	0	0	0	2	40	0.25
No Atómico	2	1	1	1	1	1	5	100	0.25
No Atómico	3	2	3	2	3	2	5	100	0.60
Incompletitud	4	1	1	0	3	1	4	80	0.38
Incompletitud	5	1	1	1	1	0	4	80	0.25
No Atómico	6	1	0	1	0	1	3	60	0.25
No Atómico	7	1	2	1	0	0	3	60	0.33
Incompletitud	8	0	0	0	0	0	0	0	0.00
Ambiguo	9	0	2	1	1	1	4	80	0.31
Ambiguo	10	1	2	1	1	0	4	80	0.31
Ambiguo	11	2	1	2	3	2	5	100	0.50
Ambiguo	12	1	0	0	2	2	3	60	0.42

Como se puede observar en la Tabla 7.3, los dos problemas más identificados en los requerimientos fueron Ambigüedad y No atomicidad, y de acuerdo a la tabla anterior se nota que los problemas con mayor porcentaje de aparición entre los encuestados son precisamente estos con un 100% de presencia para la pregunta 2, la cual se deriva del atributo de calidad “No Atómico” y la pregunta 11 de igual forma con un 100% de aparición pero esta vez derivado del atributo de Ambigüedad.

Es importante también hacer énfasis que para los problemas de Ambigüedad y falta de Atomicidad, no solo se presentó el problema, sino como se puede ver en la columna de Severidad, este se presentó regularmente a lo largo del proyecto.

7.2 Pruebas Educación

Durante un periodo de un año se estuvieron realizando pruebas con alumnos de la carrera de Ingeniería en Computación de la Universidad Autónoma de Baja California en su campus Tijuana.

Dentro de la materia "*Programación Orientada a Objetos Avanzada*" que es parte del plan de estudio de la carrera Ingeniero en Computacion, se desarrolla un proyecto final para completar la evaluación del semestre, en dichos proyectos se hace una etapa de análisis previa para posteriormente iniciar una etapa de requerimientos antes del desarrollo del proyecto, aquí es donde se inician la experimentación para determinar que tan bien realizados fueron los requerimientos y detectar problemas potenciales antes de iniciar el desarrollo de los proyectos.

Durante los dos semestres empleados en la experimentación se incluyeron 2 grupos completos de más de 15 alumnos cada uno, para un total de 4 grupos (2 grupos x 2 semestres) y un total de más de 60 alumnos

7.2.1 Condiciones del Experimento

En este experimento, el alumno no tenía acceso a la herramienta de evaluación NLARE, lo que permitió tener una primera versión de requerimientos con metodologías conocidas para el alumno. Posteriormente con acceso a la herramienta se pudo ver la mejoría de este documento de requerimientos.

7.2.2 Fases del Experimento

En resumen el experimento se llevó a cabo siguiendo las etapas descritas a continuación:

1) Se definieron lineamentos para proyectos en clase de programación orientada a objetos avanzada.
2) Cada alumno trabajo en definir los requerimientos para su proyecto, utilizando técnicas principalmente vistas en la materia de Ingeniería de Requerimientos o bien basándose en ejemplos en clase.
3) Se revisaron los requerimientos con NLARE para determinar cuántos estaban correctos siguiendo las metodologías del alumno.
4) Se le proporciono al alumno la herramienta NLARE para ayudarle en la redacción de requerimientos
5) Se realizaron evaluaciones generales para determinar problemas recurrentes

7.2.3 Materiales y Herramientas

Para este experimento se utilizaron las siguientes herramientas:

NLARE: Herramienta de análisis automático para requerimientos

LibreOffice Calc: Generación de graficas

7.2.4 Objetivos del experimento

Los principales objetivos en este experimento fueron:

1) Determinar las principales problemáticas que ocurren en el análisis de requerimientos con los estudiantes de licenciatura.
2) Determinar en qué medida puede NLARE ayudar a los estudiantes a mejorar los requerimientos

7.2.5 Requerimientos de Proyecto Educación

Los requerimientos de los proyectos evaluados no se incluyen en esta tesis por ser una numerosa cantidad de requerimientos de diferentes proyectos siendo este total superior a los 1000 requerimientos, como muestra del trabajo realizado se incluirán los requerimientos de un proyecto realizado en el periodo 2012-1 por el alumno más destacado del curso.

Requerimientos originales previos a la revisión automática con NLARE.

1. The players must first choose their avatar before beginning the game.
2. The players must save their points at the end of each game.
3. The players can pause the game by pressing the pause button.
4. The players can keep playing by pressing the continue button.
5. The players can reset the game by pressing the reset button.
6. The players can modify the volume of game by pressing the sound button at main menu.

7. The players can configure their controls by pressing the controls buttons at the main menu.
8. The players can select the limit time of the game of: 5, 10, 15 or 20 minutes in the configuration section.

Para una lista complete de los requerimientos véase sección 10.3

En esta primera etapa de requerimientos se indico al alumno utilizar un mínimo patrón de estructura que contenga los elementos previamente descritos en esta tesis como el mínimo esperado en un requerimiento funciona.

Los requerimientos expresados por el alumno fueron pasados por la herramienta NLARE para determinar su estado, y comparar lo que en la opinión del alumno era como un trabajo de requerimientos correcto y terminado contra la opinión automática de NLARE y su evaluación, los resultados se muestran a continuación:

Total Runtime = 3.72sec (2.29 loading)@ 55.2req/sec
Total Good requirements = 11 = 13.75 %
Total Wrong requirements = 69 = 86.25 %
-not atomic requirements = 11 = 13.75 %
-ambiguous requirements = 53 = 66.25 %
-incomplete requirements = 51 = 63.75 %
List of Good requirement:
FR3, FR4, FR5, FR11, FR15, FR30, FR31, FR43, FR51, FR65, FR67,

List of Not atomic requirements:

FR8, FR9, FR12, FR18, FR24, FR28, FR34, FR39, FR42, FR44, FR68,

List of Ambiguous requirements:

FR1, FR6, FR7, FR8, FR10, FR12, FR16, FR18, FR19, FR20, FR21, FR22, FR23, FR24, FR27, FR28, FR29, FR32, FR33, FR34, FR36, FR37, FR38, FR39, FR41, FR42, FR44, FR46, FR48, FR50, FR52, FR53, FR54, FR55, FR56, FR57, FR60, FR61, FR63, FR64, FR68, FR69, FR70, FR71, FR72, FR73, FR74, FR75, FR76, FR77, FR78, FR79, FR80,

List of Incomplete requirement:

FR1, FR2, FR7, FR8, FR9, FR10, FR12, FR13, FR14, FR16, FR17, FR19, FR20, FR21, FR22, FR23, FR24, FR25, FR26, FR27, FR32, FR33, FR34, FR35, FR37, FR40, FR41, FR42, FR45, FR47, FR48, FR49, FR52, FR53, FR54, FR58, FR59, FR60, FR61, FR62, FR63, FR66, FR68, FR69, FR72, FR73, FR74, FR75, FR78, FR79, FR80

La siguiente figura nos permitirá tener una visión más clara de los resultados

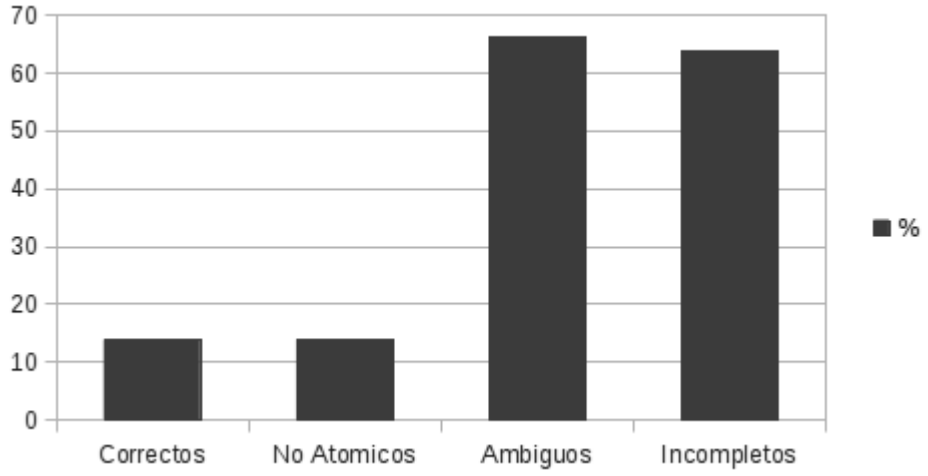

Figura 7.4. Resultados evaluación proyecto UABC 2012-1

Analizando la Figura 7.4. Podemos identificar como ya hemos estado viendo a lo largo de esta investigación que la ambigüedad y la incompletitud son los problemas más comunes, en este paso particular tenemos ambigüedad e incompletitud por encima del 60%, es prudente recordar que la sumatoria de resultados no es de 100% ya que los defectos no son mutuamente excluyentes.

Como una segunda etapa de esta prueba se procedió a proveer al estudiante con una copia de NLARE para pasar los requerimientos por una evaluación y realizar algunas correcciones de acuerdo a los resultados obtenidos por la herramienta, mismos resultados que independientemente mostramos previamente.

Una vez con la herramienta, el alumno presentó una segunda versión de los requerimientos, misma que se presenta a continuación en donde se puede ver una mayor organización de los mismo, y una estructura mas definida que facilita el entendimiento de los mismo.

A continuación mostramos la segunda edición de requerimientos:

1. The system must required that player 1 select his avatar to start the game.
2. The system must required that player 2 select his

avatar to start the game.
3. The system must required that player 1 save his points to end each game.
4. The system must required that player 2 save his points to end each game.
5. The players can pause the game by pressing the pause button.
6. The players can keep playing by pressing the continue button.
7. The players can reset the game by pressing the reset button.
8. The players can modify the volume of game by pressing the sound button at starting menu.
9. The players can configure the controls of game by pressing the control button at starting menu.

Para una versión complete de los requerimientos véase sección 10.4

Una importante nota aquí es el hecho de que la segunda versión mostrada se compone de 84 requerimientos, la razón de esto y por tanto de que no todos los requerimientos coincidan 1:1 con la primera versión, es que uno de los problemas que es la conjunción, parte de su corrección involucra la separación de requerimientos y por tanto el incremento de los mismos.

El resultado de la evaluación NLARE para la segunda muestra de requerimientos fue:
Total Runtime = 4.16 sec
Total Loading Time = 2.5 sec

Total Requirements processed = 84 @ 50.0req/sec

Total Good requirements = 84 = 100.0 %

Total Wrong requirements = 0 = 0.0 %

-not atomic requirements = 0 = 0.0 %

-ambiguous requirements = 0 = 0.0 %

-incomplete requirements = 0 = 0.0 %

List of Good requirement:

FR1, FR2, FR3, FR4, FR5, FR6, FR7, FR8, FR9, FR10, FR11, FR12, FR13, FR14, FR15, FR16, FR17, FR18, FR19, FR20, FR21, FR22, FR23, FR24, FR25, FR26, FR27, FR28, FR29, FR30, FR31, FR32, FR33, FR34, FR35, FR36, FR37, FR38, FR39, FR40, FR41, FR42, FR43, FR44, FR45, FR46, FR47, FR48, FR49, FR50, FR51, FR52, FR53, FR54, FR55, FR56, FR57, FR58, FR59, FR60, FR61, FR62, FR63, FR64, FR65, FR66, FR67, FR68, FR69, FR70, FR71, FR72, FR73, FR74, FR75, FR76, FR77, FR78, FR79, FR80, FR81, FR82, FR83, FR84,

List of Not atomic requirements:

List of Ambiguous requirements:

List of Incomplete requirement:

A continuación mostramos la gráfica resultante tas proveer al alumno con la herramienta NLARE.

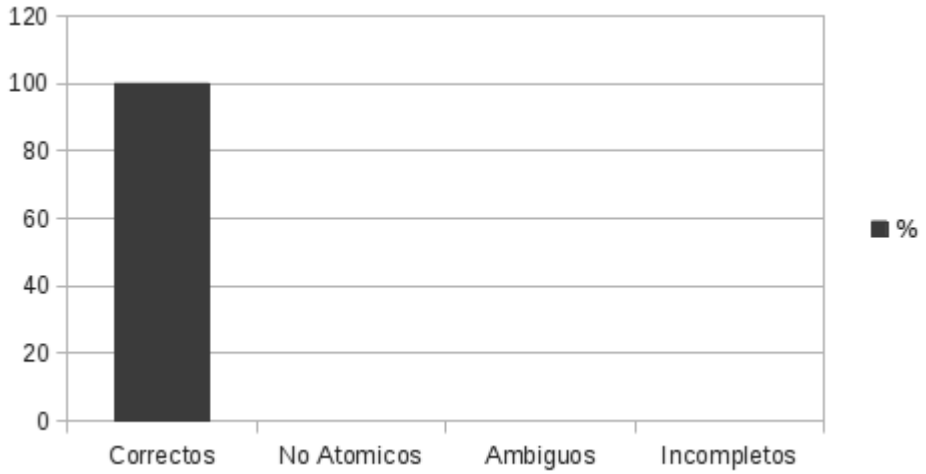

Figura 7.5. Resultados segunda versión de requerimientos con NLARE

7.2.6 Conclusiones de experimento educación.

Gracias a la herramienta el alumno fue capaz de entregar una versión que cumpliera con los estándares definidos en NLARE y entregar un conjunto de requerimientos más completo y correcto que su primera instancia.

Es importante comentar que el alumno tuvo acceso a la herramienta por lo que tuvo la oportunidad de realizar múltiples iteraciones de revisión y no significa que únicamente con una evaluación fue capaz de reducir a cero su porcentaje de error, sin embargo la capacidad del estudiante para la corrección de los requerimientos esta fuera del alcance de esta investigación.

CAPÍTULO 8

CONCLUSIONES Y TRABAJO FUTURO

8 Conclusiones y Trabajo Futuro

El tema de los requerimientos es un problema que se ha dejado un poco olvidado y curiosamente existen más avances en otras etapas del desarrollo siendo que aún falta mucho por resolver.

En mi experiencia profesional como trabajador de la Industria de software en México, he notado que la etapa de requerimientos es de las etapas más olvidadas, en numerosas ocasiones ni siquiera existe una etapa como tal, una de las razones que rodean este rechazo a los requerimientos es la idea de que se una etapa que consume numerosos recursos de tiempo y dinero y que difícilmente son justificables para el cliente final.

Sin embargo, como se comento a lo largo de esta investigación, la inversión en los requerimientos reditúa en la siguientes etapas haciendo que estas sean más compactas sin tantas iteraciones debido a errores o desacuerdos con el cliente final, es importante destacar que en ocasiones errores surgidos desde los requerimientos han tenido resultados irreversibles en proyectos a gran escala por lo que organizaciones ya maduras en el desarrollo están poniendo un pero importante en esta etapa.

La versión presentada de NLARE ha cumplido las expectativas planteadas, se ha logrado un avance en el proceso de la formalización que tanta falta le hace al desarrollo de software en general y sobre todo a la etapa de requerimientos.

La estructura modular de NLARE permite extender fácilmente sus funcionalidades por lo que es factible poder incluir sin mayor dificultad más elementos de revisión y extender a otros idiomas además del inglés.

El evaluador de requerimientos ha demostrado tener una eficiencia alta para poder considerar practico su uso, en la computadora personal de pruebas como se indico en la etapa de resultados, se logro en promedio velocidades de 50 requerimientos por segundo lo que implica que proyectos grandes pueden darle una oportunidad a NLARE sin preocuparse por el tiempo invertido ya que se pueden revisar hasta 3000 requerimientos en tan solo 1 minuto.

El evaluador ha presentado un nivel de concordancia aceptable para poder ser utilizado como una herramienta de soporte al trabajo de los ingenieros de requerimientos, y aunque consideramos bueno el trabajo que desarrolla aún no está lista para poder basar una evaluación completamente en el resultado de NLARE, sino como hemos estado manejando, un soporte a la evaluación de los ingenieros o bien paga guiar el trabajo a donde se requiere y así poder lograr una buena base en el desarrollo de software.

Sin embargo, no solo la calidad de un software es importante hoy en día, sino el tiempo requerido para lograr dicha calidad. En base a los resultados de esta investigación, se pudieron detectar problemas derivados de los requerimientos que afectan directamente el tiempo de desarrollo.

Entre los principales problemas encontrados, tenemos un déficit de información que no permite calendarizar correctamente las actividades, esto

derivado de requerimientos conjugados que si bien no afectan directamente a la calidad, si afectan a la administración de los tiempos de desarrollo.

Otro problema relacionado con el tiempo de desarrollo se liga al mal entendimiento de algunos objetivos, esto según los resultados de la investigación se puede ligar a requerimientos ambiguos, lo que crea una condición de retrabajo para realizar correctamente una actividad. Es importante señalar, que aunque el software eventualmente realice las funciones correctas, el tiempo perdido en volver a hacer una actividad ya no es recuperable lo que implica problemas graves en ciclo de vida del proyecto.

Aun quedan incógnitas por responder y problemas por atacar, sin embargo en la siguiente sección se comentara sobre necesidades ya identificadas para continuar y mejorar esta investigación.

8.1 Trabajo a futuro

En esta sección se describe el trabajo a realizar en un futuro cercano, el cual permite continuar la investigación tomando como base los logros obtenidos hasta el momento.

8.1.1 Diferentes atributos de calidad

Como se menciona a lo largo de esta investigación, existen numerosos problemas que afectan a los requerimientos, en esta caso fueron atacados ambigüedad, atomicidad e incompletitud.

Sin embargo, existen otras problemáticas que aún quedan pendientes,

tal vez el caso más peculiar es el de la ambigüedad, ya que esta se presenta en diferentes maneras. Se planea poder atacar otro tipo de ambigüedades como son la Referencial y la de Alcance. Para más información sobre este tipo de ambigüedades se puede consultar la sección 2.4

Entre otros problemas que se pretenden atacar a futuro están los déficit en los atributos de Unitario y Consistencia, ya que actualmente no se realiza ninguna validación para estas problemáticas en los requerimientos.

8.1.2 Corrección automática

Actualmente el flujo de revisión implica que el usuario deba manualmente corregir los requerimientos, aun y cuando el sistema sea capaz de detectar los problemas.

Como una oportunidad de mejora se ve la posibilidad de desarrollar un sistema experto capaz de utilizar los elementos detectados actualmente y mostrar sugerencias sobre como reparar los requerimientos.

Representaría un buen avance en como los requerimientos son completados si se pudiera tener un sistema en tiempo real capaz de ir analizando y corrigiendo de manera automática estos requerimientos, sobre todo si estos están cambiando constantemente.

8.2 Publicaciones

Durante el desarrollo de este trabajo de tesis, el cual abarco un periodo de dos años y medio de maestría, se realizaron dos presentaciones en eventos en los cuales se expuso el trabajo realizado.

Los eventos fueron:

1. Carlos Huertas, Reyes Juárez-Ramírez (2011). “A Formal Approach for Measuring the Lexical Ambiguity Degree in Natural Language Requirement Specification”. International Conference on Uncertainty Reasoning and Knowledge Engineering URKE 2011. Bali, Indonessia.
2. Carlos Huertas, Reyes Juárez-Ramírez (2012). “A Proposal to Generate Quality Computable Natural Language Requirements”. Congreso Internacional de Investigación e Innovación en Ingeniería de Software 2012. Guadalajara, Jalisco, Mexico.
3. Carlos Huertas, Reyes Juárez-Ramírez (2012). “NLARE, A Natural Language Processing Tool for Automatic Requirements Evaluation”. CUBE 2012 International Information Tech. Conference & Exhibition. Pune, India.

CAPÍTULO 9

REFERENCIAS

9 Referencias

Ambriola V. and V. Gervasi, The CIRCE approach to the systematic analysis of NL requirements, Technical Report: TR-03-05, 2003, Università Di Pisa

Barr V., Identifikation von Spezifikationsmustern im Echtzeitentwurf anhand der Fallstudie Antiblockiersystem, 1999, Diplomarbeit der Universitaet Oldenburg, Fachbereich Informatik

Barr. Michael "Embedded Systems Glossary". Neutrino Technical Library. Retrieved 2007-04-21.

Bell, T.E., Thayer, T. A: Software Requirements: are they really a problem?. Seccond International Conference on Software Engineering, (1976) 61-68

Berry, D., Kamsties, E. and Krieger, M. “From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity. A Handbook”, Technical Report, University of Waterloo, Ontario, Canada, 2003, http://se.uwaterloo.ca/~dberry/handbook/ambiguityHandbook.pdf

Berry D. and E. Kamsties, "The sintactically dangerous all and plural in specification". IEEE Software, vol. 22, pp. 55-57, 2005

Brooks, F., “No Silver Bullet: Essence and Accidents of Software Engineering”, IEEE Computer, April 1987, pp. 10-19.

Buzan, T. Gebruik je verstand (10th ed.) (C. Mouwen Trans.). Baarn: De Kern, De Fontein. 2008

Capital Community College Foundation. Capital Community College Foundation. Retrieved 20 March 2012.

Chomsky, N. Syntactic structures. Berlin/New York: Mouton de Gruyter. 2002

Davis, Alan M. Software Requirements: Objects, Functions, and States, Second Edition. Prentice Hall. ISBN 0-13-805763-X. 1993

Denger C., D. Jorg, E. Kamsties, QUASAR A Survey on Approaches for Writing Precise Natural Language Requirements, 2001, Fraunhofer IESE

Denger C. High quality requirements specifications for embedded systems through authoring rules and language patterns. Technical Report M. Sc. Thesis, 2002, Fachbereich Informatik, Universit ¨at Kaiserslautern

Denger C., D.M. Berry, E. Kamsties, Higher Quality Requirements Specifications through Natural Language Patterns, 2003, Proceedings of the IEEE International Conference on Software – Science, Technology & Engineering (SwSTE'03)

Ebenau R.G and S.H. Strauss, Software Inspection Process, 1994, McGraw Hill, New York, NY

ESI European Software Institute, "European User Survey Analysis", Report USV_EUR 2.1, ESPITI Project, January 1996

Fabbrini F., M. Fusani, G. Gnesi and G. Lami, Quality Evolution of Software Requirements Specifications, 2000, Proceedings Software and Internet Quality Week 2000 Conference, San Fransisco, CA

Field, J. Psycholinguistics. A resource book for students. London: Routledge. 2003

Fillmore C.J., The Case for Case, 1968, in Universals in Linguistic Theory

Firesmith D., Specifying Good Requirements, 2003, Journal of Object Technology, 2(4):53–64, July-August 2003

Fuchs N.E and R. Schwitter, Controlled English for Requirements Specification, 1996, IEEE Computer Special Issue on Interactive Natural Language Processing

Gause D.C and D. M. Weinberg, Exploring Requirements: Quality Before Design, 1989, Dorset House Publishing Company Inc.

Gilb T. and D. Graham, Software Inspection, 1993, Addison Wesley Wokingham, UK

Götz R. and C. Rupp, Regelwerk Natürlichsprachliche Methode, 1999, Sophist, Nürnberg, Germany

Grady, Robert. Practical Software Metrics for Project Management and Process Improvement. Prentice Hall. 1992

Grenat, M., & Taher, M. On the translation of structural ambiguity. Al- Satil (2008), 4, 9-20

Hooks I., Writing Good Requirements, Vol. 2, pp. 197-203, 1994, Proceedings of the Fourth International Symposium of the NCOSE 2, San Jose, CA

Huddleston & Pullum (2002), chapter 7

Huddleston & Pullum (2002) ("CGEL") 2002 p 602.

IEEE STANDARD 830-1998 - IEEE Recommended Practice for Software Requirements Specifications

Juristo, N., Moreno, A. M., & López, M. (2000). How to use linguistics instruments for ObjectOriented Analysis. IEEE Software, (May/June): 80–89.

Klapaftis, I., & Manandhar, S. (2005). Google & WordNet based Word Sense Disambiguation. Proceedings of the workshop on learning and extending ontologies by using Machine Learning methods, 241-248.

Kuhlman Dave. "A Python Book: Beginning Python, Advanced Python, and Python Exercises". "Python is a high-level general purpose programming language"

Laplante Philip A. CRC Press 2007 What Every Engineer Should Know about Software Engineering. 2007

Lamsweerde A.V., Building Formal Requirements Models for Reliable Software, 2000, In The Future of Software Engineering, A. Finkelstein (ed.), ACM Press

Lauesen Soren & Otto Vinte, "Preventing Requirements Defects", Sixth International Workshop on Requirements, Stockholm, 2000.

Lamsweerde Axel Van and Emmanuel Letier, Integrating Obstacles in Goal-Driven Requirements Engineering. 1998

Levelt, W.J.M. Speaking: From intention to articulation. Cambridge: MIT Press. 1989

Levinson, S.: Pragmatics. Cambridge, Eng.: Cambridge University Press. 1983

Lewin, K. Action research and minority problems. En K. Lewin (201 – 216): Resolving Social Coflicts: Selected Papers on Group Dynamics (ed. G. Lewin). London: Souvenir Press. 1973

Loos, Eugene E., et al. 2003. Glossary of linguistic terms: What is a noun?

Martinez A., Pastor O., Estrada H., A pattern language to join early and late requirements, 2004, pp 51-64 Anais do WER04 - Workshop em Engenharia de Requisitos, Tandil, Argentina

McConnell, Code Complete (2nd ed.). Microsoft Press (2004). p. 29

Miller G. A., R. Beckwith, C. D. Fellbaum, D. Gross, K. Miller. 1990. WordNet: An online lexical database. Int. J. Lexicograph. 3, 4, pp. 235–244.

Nikora Alen P., Experiments in Automated Identification of Ambiguous Natural Language Requirements, 2011

Nuseibeh B., S. Easterbrook and Russo A., Making Inconsistency Respectable in Software Development, Journal of Systems and Software, 58(2):171-180,September 2001, Elsevier Science Publishers.

Ohnishi A., Customizable Software Requirements Languages, 1994, Proceedings of the Eighth International Computer Software and Application Conference (COMPSAC), IEEE Computer Society, Los Alamitos, CA

Paul, Anne; Adams, Michael (2009), How English Works: A Linguistic Introduction (2nd ed.), New York: Pearson Longman, p. 152

Pierre Bourque and Robert Dupuis, ed. (2004). Guide to the Software Engineering Body of Knowledge - 2004 Version. IEEE Computer Society. pp. 2–1. ISBN 0-7695-2330-7.

Postal, Paul, Dinneen, Francis P., ed., "On So-Called "Pronouns" in English", Report of the Seventeenth Annual Round Table Meeting on Linguistics and Language Studies (Washington, D.C.: Georgetown University Press). 1966.

Python web, 2012 "About Python". Python Software Foundation. Retrieved 24 April 2012., second section "Fans of Python use the phrase "batteries included" to describe the standard library, which covers everything from asynchronous processing to zip files."

Python Docs http://docs.python.org/library/time.html

Python Docs 2 http://docs.python.org/library/re.html

Quiroga-Clare, C. Language ambiguity: A curse and a blessing. Translation Journal Volume 7(1) (2003). Retrieved from http://accurapid.com/journal/23ambiguity.htm

Reyes Juárez-Ramírez, "Towards improving user interfaces: a proposal for integrating functionality and usability since early phases", IEEE, Indonesia, 2011.

Robertson S. and J. Robertson, Mastering the Requirements Process,1999, Addison-Wesley Professional,

Rolland C. and Proix C., A Natural Language Approach for Requirements Engineering, 1992, pp. 257-277 in Proceedings of Conference on Advanced Informations Systems Engineering, CAiSE, Manchester UK

Sawyer P. and G. Kotonya, Chapter 2 Software Requirements, Guide to the Software Engineering Body of Knowledge Trial Version SWEBOK, A Project of the Software Engineering Coordinating Committee, 2001, © IEEE

Sommerville I., Software Engineering Sixth Edition, 2001 © Addison-Wesley

Song, Y., Yi, E., Kim, E, & Geunbae Lee, G. (2004). POSBIOTM-NER: A machine learning approach for bio-named entity recognition. Proceedings of the EMBO Workshop on critical assessment of text mining methods in molecular biology.

Standish, The Standish Group Report - CHAOS Report 2005. http://www.projectsmart.co.uk/docs/chaos-report.pdf

Steve Heath (2003). Embedded systems design. EDN series for design engineers (2 ed.). Newnes. p. 2. ISBN 9780750655460. "An embedded system is a microprocessor based system that is built to control a function or a range of functions."

Stubbs, M. (2001). Words and phrases: Corpus studies of lexical semantics. Oxford: Blackwell Publishers.

Swinney, DAVID A. 1979. Lexical access during sentence comprehension (re)consideration of context effects. Journal of Verbal Learning and Verbal Behavior 18.645–659.

Tinholt, H.W., & Nijholt, A. (2007). Computational humour: utilizing cross-reference ambiguity for conversational jokes. 7th International Workshop on Fuzzy Logic and Applications (WILF 2007): Lecture Notes in Artificial Intelligence, 477–483. Berlin: Springer Verlag.

Westhoff, G. (2002). Wat gebeurt er tijdens het lezen en hoe kan je dat aanleren? Vonk. Retrieved from http://users.skynet.be/taalonderwijs/Weetjes-lezen.htm.

Wiegers, Karl E. (2003). Software Requirements 2: Practical techniques for gathering and managing requirements throughout the product development cycle, 2nd ed., Redmond: Microsoft Press.

Wilson W, Rosenberg L, Hyatt L. Automated quality analysis of natural language requirement specifications, in Proceedings of Fourteenth Annual Pacific Northwest Software Quality Conference. 1996.

Zowghi D., REAUTS – Requirements Engineering Activities, Requirements Engineering AUTS Home Page, last accessed August 2007 http://research.it.uts.edu.au/re/re_uts_activities.html

CAPITULO 10

10 Apendices

10.1 Requerimientos Proyecto Industria Originales

1. El sistema de Coordenada Móvil debe contar con la capacidad de leer los datos que se encuentran en la memoria del módulo, en caso de que esta sea insertada en una computadora personal
2. El sistema Coordenada Móvil podrá generar consultas y reportes de históricos, tiempos muertos y excesos de velocidad con la información contenida en la memoria Servidor
3. El servidor deberá contar con una nueva consola de recepción de ubicaciones especializada en tramas tipo Wi-Fi
4. La consola debe ser capaz de recibir una trama de datos del módulo
5. La consola debe ser capaz de interpretar y manipular cada una de las tramas enviadas por el módulo
6. La consola trabajará de manera asíncrona en el proceso de interpretación de cada trama, con el uso de hilos, para asegurar que toda la información que se recibe es procesada
7. La consola deberá almacenar las ubicaciones en bases de datos
8. El servidor deberá de contar con una nueva tabla para la configuración y almacenamiento de configuración de redes inalámbricas de todos los módulos
9. El servidor deberá de poder enviar configuraciones de redes inalámbricas a los módulos
10. La aplicación cliente debe poder realizar consultas de históricos, tiempos muertos excesos de velocidad y generación de reporte de las ubicaciones enviadas por el módulo
11. La aplicación cliente debe poder configurar las redes inalámbricas de un módulo
12. La aplicación cliente debe poder modificar las redes inalámbricas de un

módulo

13. La aplicación cliente debe poder eliminar una configuración de red inalámbrica de un módulo

14. La aplicación cliente debe poder leer los datos de la memoria del módulo, y presentarlos como si fuera una consulta al servidor

15. La consola en el servidor debe ser capaz de recibir las diferentes tramas de datos enviadas por la interfaz electrónica

16. La consola en el servidor debe poder interpretar y manipular cada una de las diferentes tramas enviadas por la interfaz electrónica: temperatura, presión, nivel y configuración

17. La consola al recibir la trama de monitoreo TPV debe guardar la información en las bases de datos de acuerdo a la trama recibida.

18. La consola debe enviar la nueva notificación de parámetro TPV al buzón/repositorio de notificaciones del cliente para que la aplicación cliente pueda consultarla. Esta notificación depende del tipo de reporte recibido.

19. Las notificaciones de parámetros TPV deben permanecer 4 horas en el buzón del cliente para que se puedan consultar.

20. Al recibir la trama de petición de configuración de la interfaz electrónica, la consola debe verificar qué configuración tiene esa unidad en la base de datos para enviarle la nueva trama con la información de los servicios TPV que tenga activos.

21. La consola debe ser capaz de enviar trama de configuración de los parámetros TPV hacia la interfaz inteligente

22. El usuario responsable de monitoreo de parámetros (más adelante llamado solo el usuario) debe tener la opción de visualizar la información de monitoreo de los parámetros TPV

23. El usuario debe tener la oportunidad de filtrar las lecturas de los diferentes sensores en la tabla del tablero de control, mediante la selección

del tipo de sensor o sensores que desee visualizar.

24. La aplicación debe tener un apartado especial para todas las operaciones de consulta y configuración de Sensado TPV.

25. La aplicación debe notificar al usuario cuando se registre que un parámetro salió de rango, ya sea menor o mayor.

26. La aplicación debe notificar al usuario cuando se registre que un parámetro regresó al rango permitido.

27. El usuario debe ser capaz de configurar si desea visualizar las notificaciones de rango y/o si desea que se reproduzca el sonido exclusivo de dichas notificaciones.

28. El usuario debe poder consultar histórico de las mediciones registradas de parámetros TPV.

29. El usuario debe contar con la opción de consultar histórico de las mediciones registradas de parámetros TPV de más de un vehículo (Multi-Histórico).

30. El usuario debe contar con la opción de consultar histórico de las mediciones Fuera de Rango en un periodo de tiempo establecido (fecha y hora).

31. La consulta de Fuera de rango podrá ser de mediciones de temperatura, presión o las 2 al mismo tiempo.

32. El usuario debe tener la opción de realizar Rastreo Continuo del monitoreo de parámetros TPV.

33. El usuario debe tener la opción de generar una gráfica para visualizar la información de las consultas de Histórico TPV y Fuera de Rango.

34. La gráfica se genera a partir de la consulta realizada por el usuario, ya sea Histórico TPV o Fuera de Rango.

35. La gráfica debe tener con la opción de ser copiada al portapapeles para poder grabarla en archivo de imagen.

36. Los registros de monitoreo TPV (Ubicaciones TPV) deben mostrarse en el mapa de acuerdo al reporte registrado.

37. El usuario debe tener la opción de poder configurar los rangos permitidos de los parámetros de temperatura y presión. Para temperatura y presión se pueden configurar el rango mayor y rango menor de cada uno de los sensores.

38. El usuario debe tener la opción de configurar los rangos de varios vehículos al mismo tiempo (cambios múltiples).

39. El usuario debe contar con la opción de generar reporte del monitoreo de sensado de temperatura, presión y nivel.

40. El usuario debe tener la opción de poder configurar los rangos permitidos de los parámetros de temperatura y presión. Para temperatura y presión se pueden configurar el rango mayor y rango menor de cada uno de los sensores.

10.2 Requerimientos Proyecto Industria Traducidos

1. The mobile coordinate system must have the ability to read the data in the memory module if this is inserted into a personal computer
2. The mobile coordinate system can generate queries and historic, downtime and speeding reports with the information contained in the server
3. The server must have a new console for location specializing receipts in Wi-Fi frames
4. The console must be able to receive a data frame from the module
5. The console must be able to interpret and manipulate each of the frames sent by the module
6. The console will work asynchronously in the process of interpretation of each plot with the use of threads to ensure that all information received is processed
7. The console will store the locations in databases
8. The server must to have a new table for storing configuration and wireless network configuration of all modules
9. The server must be able to send wireless network settings to the modules
10. The client application must be able to configure a module wireless network
11. The client application must be able to modify a module wireless network
12. The client application must be able to delete a module wireless network configuration
13. The client application must be able to read data from memory module and presenting them as if it were a query to the server
14. The mobile coordinate system must have the option of monitoring and adjustment temperature, pressure and level parameters.
15. The server console must be able to receive different data frames sent by

the electronic interface.

16. The server console must be able to interpret and manipulate each of the different frames sent by electronic interface like temperature, pressure, level and configuration.

17. The console once TPV monitoring is received must save the information in the database according to the frame received.

18. The console should send the new notice to the mailbox parameter TPV and customer notifications repository for the client application to view it. This notice depends on the type of report received.

19. The TPV parameters notifications must remain 4 hours in client mailbox so they can consult.

20. Upon receiving the configuration request frame electronic interface the console must verify what configuration the unit has in the database to send the new frame with the TPV service information that be active

21. The console must be able to send TPV parameter settings to the smart interface.

22. The user should have the option to view the TPV monitoring data in control panel.

23. The user must have the opportunity to filter the readings of different sensors on the control board table by selecting the type of sensor or sensors to be displayed.

24. The application must have a special section for all query operations and configuration of TPV Sensing.

25. The application must notify the user when parameter out of range either minor or major is detected

26. The application must notify the user when a parameter return to the permitted range.

27. The user should be able to set if range notifications and or if play of

unique sound of these reports are wanted

28. The user should be able to check the TPV parameters measurements recorded historic.

29. The user must have the option to view historical measurements of TPV parameters registered by more than one vehicle

30. The user must have the option to view historical Out of Range measurements over a period of time

31. The query may be out of range of measurements of temperature, pressure or both at the same time.

32. The user must have the option of Continuous Tracking TPV parameter monitoring

33. The user must have the option to generate a graph to display the information from the TPV and Historic queries Out of Range.

34. The graph is generated from the inquiry made by the user either historical or Out of Range TPV.

35. The graph should have the option to be copied to the clipboard in order to burn it to a image file.

36. The monitoring TPV locations records should be shown on the map according to the registered report.

37. The user must have the option to configure the allowed ranges of temperature and pressure parameters. For Temperature and pressure range it is possible to set higher and lower range of each sensor.

38. The user must have the option to set the ranges of several vehicles at the same time

39. The user must have the option to generate sensing monitoring report temperature, pressure and level. It generates a single report and the choice of which sensor or sensors you want the report is with the filters on the board of control.

40. The user must have the option to generate TPV sensing monitoring report combined with position of the vehicle locations and whether to include graphics in the report or not.

10.3 Requerimientos educación versión original

1. The players must first choose their avatar before beginning the game.
2. The players must save their points at the end of each game.
3. The players can pause the game by pressing the pause button.
4. The players can keep playing by pressing the continue button.
5. The players can reset the game by pressing the reset button.
6. The players can modify the volume of game by pressing the sound button at main menu.
7. The players can configure their controls by pressing the controls buttons at the main menu.
8. The players can select the limit time of the game of: 5, 10, 15 or 20 minutes in the configuration section.
9. The players can use the mouse and keyboard to access the menus and buttons.
10. The system must place 6 rows of tile in each playfield at starting game.
11. The system must add a tiles row at the bottom of the stack every two seconds.
12. The system must remove the group of tiles that have been matched three or more tiles horizontally or vertically.
13. The system must send a block of one row to the opponent every time the player matches four tiles
14. The system must send a block of two rows to the opponent every time the player matches five tiles.
15. The system must send a block of tree rows to the opponent every time the player matched six tiles.
16. The system must swap the first row of block into tiles when the

player matches three tiles under the block.

17. The system must swap two rows of blocks into tiles when the player matches four tiles under the block.
18. The system must swap all rows of block into tiles when the player matched five or more tiles under the block.
19. The system must show an alert to player when the stack is three rows under the top.
20. The system must finish the round when a player's stack reaches to top of playfield.
21. The system must increase 100 points to a player when he/she matches 3 tiles
22. The system must increase 200 points to a player when he/she matches 4 tiles.
23. The system must increase 500 points to a player when he/she matches 5 tiles.
24. The system must increase 1000 points to a player when he/she matches 6 or more tiles simultaneously.
25. The system must play a melody when begins the game.
26. The system must increase the velocity of rows every thirty seconds.
27. The system must show the main menu at the beginning of the video game.
28. The system must delay two seconds to send a block to opponent and in this time accumulate other blocks if there are send
29. The system must place the cursor in the center on each playfield.
30. The system must sort the tiles at random in all game.
31. The system must finish the round in fifteen minutes after the game started.

32. The system must show the records when the players press the records button at the main menu.
33. The system must show the elapsed time after the game started in the game's panel.
34. The system must move all tiles down that don't have a support up by some other tile or block.
35. The system must count tree seconds before starting the game.
36. The system must go back to main menu when the players press the back button at the game's panel.
37. The system must show the controls of players when they press the control button at main menu.
38. The system must show the instructions of game when the players press the instructions button at main menu.
39. The system must show the name of game, his version and name of the author when the players press the about button at main menu.
40. The system must show who the winner is at end of game.
41. The system must ask to players if they want to play again at end of game.
42. The system must ask to players if they want to exit the game or back to main menu if they have selected won't play again.
43. The system must sort the records in ascending order of points at end of each game.
44. The system must show a grid playfield for each player with 12 rows and 6 columns.
45. The system must show a message to players when has occurred a error.
46. The system must show a confirm message if the players press the main menu button at game's panel.

47. The system must encrypt the records sheet of the players.
48. The system must have a button to go back to main menu at the controls section.
49. The system must have a button to save the configuration of the controls at controls section.
50. The system must have a button to go back to main menu at the sound section.
51. The system must have a button to save the volume at sound section.
52. The system must have a button to go back to main menu at the credits section.
53. The system must have a button to go back to main menu at the instructions section.
54. The system must have a button to go back to main menu at the records section.
55. The system must have a button to go back to main menu at the avatar's selection section.
56. The system must have a button for player one to save his/her avatar's selection.
57. The system must have a button for player two to save his/her avatar's selection.
58. The system must have a button to continue at the points section to end the game.
59. The system must have a button to save the records of players at the records section to end the game.
60. The system must start the game if the players don't select an avatar in 15 seconds.
61. The system must have a button to exit at main menu.

62. The system must have a button to exit when the program show an error message.
63. The system must show the e-mail of the author in the error message.
64. The system must have a button to configure the options of game at main menu.
65. The system must have the option of limit time at the configuration section.
66. The system must finish the game when some player wins two of three rounds.
67. The system must show number of rounds won of each player during the game.
68. The system must delay one second the increasing of rows when a player matched four tiles or more.
69. Player one can increase the velocity of the rows in the playfield by pressing a key.
70. Player one can move the cursor to the right by pressing a key.
71. Player one can move the cursor to the left by pressing a key.
72. Player one can move the cursor down by pressing a key.
73. Player one can move the cursor up by pressing a key.
74. Player one can swap the tiles horizontally by pressing a key.
75. Player two can increase the velocity of the rows in the playfield by pressing a key.
76. Player two can move the cursor to the right by pressing a key.
77. Player two can move the cursor to the left by pressing a key.
78. Player two can move the cursor down by pressing a key.
79. Player two can move the cursor up by pressing a key.
80. Player two can swap tiles horizontally by press a key.

10.4 Requerimientos educación versión corregida

1. The system must required that player 1 select his avatar to start the game.
2. The system must required that player 2 select his avatar to start the game.
3. The system must required that player 1 save his points to end each game.
4. The system must required that player 2 save his points to end each game.
5. The players can pause the game by pressing the pause button.
6. The players can keep playing by pressing the continue button.
7. The players can reset the game by pressing the reset button.
8. The players can modify the volume of game by pressing the sound button at starting menu.
9. The players can configure the controls of game by pressing the control button at starting menu.
10. The players can modify the limit of time to finish the game by selecting the time to a list at configuration section.
11. The players can use the mouse to access the menus during all game.
12. The players can use the keyboard to access the menus during all game.
13. The system must set 6 rows of tiles in random order on each playfield to start the game.
14. The system must add a tiles row at the bottom of the stack every two seconds.
15. The system must remove the group of tiles that have been

matched more to two tiles in the row.

16. The system must remove the group of tiles that have been matched more to two tiles in the column.
17. The system must send a row of block to the opponent when the player matches four tiles during all game.
18. The system must send a block of 2 rows to the opponent when the player matches five tiles during all game.
19. The system must send a block of tree rows to the opponent when the player matched six tiles.
20. The system must swap a row of block to tiles when the player matches three tiles under the block.
21. The system must swap rows of blocks to tiles when the player matches four tiles under the block.
22. The system must swap rows of block to tiles when the player matches up to four tiles under the block.
23. The system must show an alert to player when the stack is three rows under the limit during the round.
24. The system must finish the round when a stack of player reaches the limit of playfield.
25. The system must increase the points to a player in 100 points when he matches 3 tiles during all game.
26. The system must increase the points to a player in 200 points when he matches 4 tiles during all game.
27. The system must increase the points to a player in 500 points when he matches 5 tiles during all game.
28. The system must increase the points to a player in 1000 points when he matches up to 5 tiles during all game.
29. The system must play a melody when begins the game up to

starting of round.

30. The system must increase the velocity of rows every thirty seconds during all game.
31. The system must show the starting menu at the beginning of the video game.
32. The system must delay two seconds to send a block to opponent in this time accumulate more blocks if there are send during all game.
33. The system must place the cursor in the center on each playfield.
34. The system must sort the tiles at random in all game.
35. The system must finish the round in fifteen minutes after the game started.
36. The system must show the records of players in descending order by pressing the records button at staring menu.
37. The system must show the time elapsed after the game started in the panel of game.
38. The system must move all tiles down that no have a support up by something.
39. The system must count tree seconds before starting the game in the panel of game.
40. The system must go to main menu when the players press the back button at the panel of game.
41. The system must show the controls of players to play in a list by pressing the control button at starting menu.
42. The system must show the instructions of game when the players press the instructions button at starting menu.
43. The system must show the details of game when the players press the about button at starting menu.
44. The system must show the winner is at end of game.

45. The system must ask to players if they want to continue playing a game at end of game.
46. The system must back to starting menu if the players selected no continue playing at end of game.
47. The system must sort the records in ascending order of points at end of each game.
48. The system must show a grid playfield for each player with 12 rows with 6 columns.
49. The system must show a message to players when has occurred a error during the game.
50. The system must show a confirm message if the players press the starting menu button at panel of game.
51. The system must encrypt the records sheet of the players in the file of records.
52. The system must have a button to go to starting menu at the controls section when a player pressed that button.
53. The system must have a button to save the configuration of the controls at controls section when a player pressed that button.
54. The system must have a button to go to starting menu at the sound section when a player pressed that button.
55. The system must have a button to save the volume at sound section when a player pressed that button.
56. The system must have a button to go to starting menu at the credits section when a player pressed that button.
57. The system must have a button to go to starting menu at the instructions section when a player pressed that button.
58. The system must have a button to go to starting menu at the records section when a player pressed that button.

59. The system must have a button to go to starting menu at the selection of avatar.
60. The system must have a button for player one to save his avatar when a player pressed that button.
61. The system must have a button for player two to save his avatar when a player pressed that button.
62. The system must have a button to continue at the points section to end the game when a player pressed that button.
63. The system must have a button to save the records of players at the records section to end the game when a player pressed that button.
64. The system must start the game with the selected avatar if the players do no select an avatar in 15 seconds at the selection avatar section.
65. The system must have a button to exit at starting menu when a player pressed that button.
66. The system must have a button to exit when the program show an error message when a player pressed that button.
67. The system must show the email of the author in the error message.
68. The system must have a button to configure the options of game at starting menu.
69. The system must have the option of limit time at the configuration section.
70. The system must finish the game when some player wins two of three rounds in the game.
71. The system must show number of rounds won of each player during the game.
72. The system must delay a time to increasing of rows when a

player matched up to four tiles during all game.

73. Player one can increase the velocity of the rows in the playfield by pressing the button of velocity during all game.
74. Player one can move the cursor to the right by pressing a button during all game.
75. Player one can move the cursor to the left by pressing a button during all game.
76. Player one can move the cursor down by pressing a button during all game.
77. Player one can move the cursor up by pressing a button during all game.
78. Player one can swap the tiles in a by pressing a button during all game.
79. Player two can increase the velocity of the rows in the playfield by pressing a button during all game.
80. Player two can move the cursor to the right by pressing a button during all game.
81. Player two can move the cursor to the left by pressing a button during all game.
82. Player two can move the cursor down by pressing a button during all game.
83. Player two can move the cursor up by pressing a button during all game.
84. Player two can swap tiles in a row by press a button during all game.

10.5 Encuesta Aplicada Proyecto Industria

1) Delegación de actividades no acorde al perfil de la persona

( )Nunca ( )Poco ( )Regularmente ( )Mucho ( )Siempre

2) Dificultad para elegir la más eficiente prioridad de las actividades

( )Nunca ( )Poco ( )Regularmente ( )Mucho ( )Siempre

3) Problemas para cumplir con los tiempos calendarizados para las actividades

( )Nunca ( )Poco ( )Regularmente ( )Mucho ( )Siempre

4) Necesidad de investigación de información no incluida en la especificación de requerimientos

( )Nunca ( )Poco ( )Regularmente ( )Mucho ( )Siempre

5) Necesidad de asumir información por falta de una concreta especificación

( )Nunca ( )Poco ( )Regularmente ( )Mucho ( )Siempre

6) Dificultad para generar casos de prueba

( )Nunca ( )Poco ( )Regularmente ( )Mucho ( )Siempre

7) Problemas al intentar determinar el costo real total del proyecto

( )Nunca ( )Poco ( )Regularmente ( )Mucho ( )Siempre

8) Necesidad de ajustar recursos de dinero para completar el proyecto

( )Nunca ( )Poco ( )Regularmente ( )Mucho ( )Siempre

9) Desarrollo de actividad o funcionalidades que no quedaron finalmente en el producto terminado

( )Nunca ( )Poco ( )Regularmente ( )Mucho ( )Siempre

10) Retrabajo debido a errores por un entendimiento incorrecto de la actividad

( )Nunca ( )Poco ( )Regularmente ( )Mucho ( )Siempre

11) Necesidad de ajustar recursos de tiempo para completar el proyecto

( )Nunca ( )Poco ( )Regularmente ( )Mucho ( )Siempre

12) Necesidad de ajustar el diseño para poder cuadrar con el resultado esperado

( )Nunca ( )Poco ( )Regularmente ( )Mucho ( )Siempre