UNIVERSIDAD DE COSTA RICA
SISTEMA DE ESTUDIOS DE POSGRADO

AUTOMATIZACIÓN PARCIAL DE LA REVISIÓN DE ASPECTOS DE PRECISIÓN, NO-AMBIGÜEDAD Y VERIFICABILIDAD EN REQUERIMIENTOS DE SOFTWARE ESCRITOS EN LENGUAJE NATURAL

Tesis sometida a la consideración de la Comisión
del Programa de Estudios de Posgrado en
Computación e Informática para optar por el grado de
Magister Scientiae en Computación e Informática

ALLAN FRANCISCO BERROCAL ROJAS

Ciudad Universitaria Rodrigo Facio, Costa Rica

2009

Dedicatoria

Dedicada a nuestra Madre Naturaleza,

fuente y dueña de mi vida,

eterna emprendedora,

de quien todo tomamos

y a quien poco ofrecemos a cambio.

A mi amada Madre

María Emilce Rojas Garita

como agradecimiento por su impecable bondad hacia el mundo.

A mi amado Padre

Francisco Berrocal Monge

como admiración por su valentía y espíritu de libertad.

Agradecimientos

Extiendo un sincero agradecimiento al sistema de becas de la Universidad de Costa Rica. Gracias a su apoyo he logrado realizar muchos de mis logros académicos, y en este caso particular, mis estudios de maestría.

El apoyo que siempre me brindó el comité de tesis, no solo fue sumamente valioso, sino que también fue determinante para lograr la culminación exitosa de este proyecto. Estoy profundamente agradecido con las y los miembros del comité por brindarme la motivación y la guía adecuada para realizar esta investigación.

A mi directora de tesis, Dra. Elena Gabriela Barrantes, extiendo un agradecimiento especial por permitirme el privilegio de trabajar a su lado. Su constante dedicación y sus acertadas observaciones siempre fueron un ingrediente fundamental para conducir la investigación por la vía adecuada. Sin lugar a ninguna duda, puedo afirmar que debo a Gabriela gran parte del éxito en este paso profesional.

A mi asesora de tesis, M.Sc. Marta E. Calderón, agradezco profundamente sus varias y valiosas contribuciones durante la investigación. Con mucha satisfacción y humildad, debo a Marta gran parte del privilegio de entregar un trabajo que esperamos sea agradable al lector.

A mi asesor de tesis, Dr. Marcelo Jenkins, agradezco muchísimo la confianza que me brindó durante todo el proceso. El valor de sus objetivas y oportunas contribuciones fue crucial para esta investigación y por ello agradezco enormemente a Marcelo.

A mis colegas en la empresa Avionyx S.A. también extiendo un enorme agradecimiento por su apoyo inagotable. Su calor humano y profesionalismo mantuvieron en mi encendida la llama de la motivación para realizar este trabajo. En particular agradezco de corazón a los colegas que contribuyeron directamente con la investigación, dedicando valioso tiempo de sus agendas a esta labor: Ing. Eduardo Trejos, Ing. Esteban Sánchez, Ing. Daniel Pérez, Ing. Leonardo Jiménez, Ing. Roger Fernández e Ing. Fernando Bogarín.

A mi jefe M.Sc. Gustavo Cubas, agradezco su confianza y comprensión mostrando siempre una actitud de apoyo hacia este proyecto. Para Gustavo un caluroso gesto de agradecimiento.

Finalmente, doy un sincero agradecimiento a mis compañeros del grupo de tesiarios de la maestría: Adrián, Roberto, Federico, Eduard, Edgar. Gracias a sus observaciones logré mejorar este trabajo de forma extraordinaria.

"Esta tesis fue aceptada por la Comisión del Programa de Estudios de Posgrado en Computación e Informática de la Universidad de Costa Rica, como requisito parcial para optar al grado de Magister Scientiae en Computación e Informática."

Dra. Gabriela Marín Raventós
Decana del Sistema de Estudios de Posgrado

Dra. Elena Gabriela Barrantes Sliesarieva
Directora de Tesis

M.Sc. Marta Eunice Calderón Campos
Asesora de Tesis

Dr. Marcelo Jenkins Coronas
Asesor de Tesis

Magister Lidia Arévalo Bravo
Directora del Programa de Posgrado en Computación e Informática

Allan Francisco Berrocal Rojas
Candidato

Tabla de Contenidos

Dedicatoria **iii**

Agradecimientos **v**

Hoja de Aprobación **vii**

Resumen **xv**

Abstract **xvii**

Índice de Tablas **xix**

Índice de Figuras **xxi**

1 Introducción **1**

1.1 La Calidad en el Software 2

1.2 Verificación de Requerimientos de Software 4

1.3 El Trabajo Realizado 6

2 Antecedentes y Trabajo Relacionado **9**

3 Objetivos **15**

3.1 Objetivo Principal 15

3.2 Objetivos Específicos 16

4 Marco Teórico **17**

4.1 Ciclo de Vida del Software 18

4.2 Estándar DO-178B 20

4.3 Ambigüedad, Precisión y Verificabilidad 23

4.3.1 Ambigüedad 24

4.3.2 Precisión 27

4.3.3 Verificabilidad 31

4.4 Procesamiento de Lenguaje Natural 32

4.4.1 Parsing 33

4.4.2 Tagging 36

4.5 Recursos Lingüísticos 37

4.5.1 WordNet 38

4.5.2 VerbNet 40

5 Alcances y Limitaciones **43**

5.1 Alcances 44

5.2 Delimitación 46

6 Metodología General **51**

7 Selección de Elementos **53**

7.1 Nomenclatura del Trabajo 53

7.1.1 Metodología de Creación de Nomenclatura 53

7.1.2 Nomenclatura Propuesta 54

7.2 Selección de Elementos 58

7.2.1 Metodología 59

7.2.2 Resultados y Análisis 59

7.3 Validación de Selección 64
7.3.1 Metodología 64
7.3.2 Resultados y Análisis 66
7.4 Asignación de Puntajes 69
7.4.1 Metodología 70
7.4.2 Resultados y Análisis 71

8 Diseño y Construcción del Prototipo **75**
8.1 Normalización de Requerimientos 75
8.1.1 Metodología 77
8.1.2 Implementación 78
8.1.3 Validación 81
8.2 Análisis de Elementos 82
8.2.1 Metodología 82
8.2.2 Implementación del Prototipo 84
8.2.2.1 Técnicas y Recursos Utilizados 84
8.2.2.2 Arquitectura de Software 86
8.2.2.3 Presentación del Prototipo 87
8.2.2.4 Salidas del Prototipo 88
8.2.3 Validación del Prototipo 92

9 Comparación del Prototipo con los Profesionales **95**
9.1 Selección de Datos de Prueba 95
9.1.1 Metodología 96
9.1.2 Resultados y Análisis 97
9.2 Evaluación con Profesionales 100
9.2.1 Metodología 100

9.2.2 Resultados y Análisis 101
9.3 Aplicación de la Evaluación 102
9.3.1 Metodología 102
9.3.2 Resultados y Análisis 103

10 Análisis General de Resultados **109**

11 Conclusiones y Trabajo Futuro **115**
11.1 Conclusiones 115
11.2 Trabajo Futuro 118

Apéndices **121**

Apéndice A Etiquetas en Penn Treebank **123**

Apéndice B Detalle de Elementos **125**

Apéndice C Instrumento de Evaluación de Elementos **135**

Apéndice D Resultados para la Evaluación de Elementos **143**

Apéndice E Instrumento de Evaluación de Puntajes **145**

Apéndice F Resultados para la Evaluación de Puntajes **149**

Apéndice G Ejemplos de Reportes del Prototipo **151**

Apéndice H Instrumento de Evaluación de Requerimientos **153**

Apéndice I Participantes en los Experimentos **159**

Apéndice J Resultados del Proceso de Evaluación de Requerimientos **173**

Bibliografía **175**

Resumen

Los requerimientos de software son parte fundamental en el ciclo de vida del software. Una especificación de requerimientos que se realice de forma apropiada aumenta las posibilidades de producir software que satisfaga las necesidades de los ususarios.

En la industria de aviónica en particular, muchos de los sistemas empotrados que se producen se consideran sistemas de misión crítica. Por consiguiente, el proceso de desarrollo de software es muy riguroso en cada una de las etapas del ciclo de vida. Para garantizar una buena definición de requerimientos, el estándar de desarrollo de software más utilizado en aviónica, DO-178B, establece algunas propiedades que los requerimientos de software deben satisfacer.

En esta investigación se explora un enfoque sencillo para automatizar parcialmente la tarea de verificar que requerimientos de software escritos en lenguaje natural cumplan con tres de las propiedades requeridas por el estándar: *precisión*, *no-ambigüedad* y *verificabilidad*.

Se propone una definición acotada para cada uno de dichos conceptos, tomando en cuenta las principales características que los requerimientos de software deben tener para cumplir con las propiedades establecidas. Se implementa un prototipo de software que aplica técnicas de procesamiento de lenguaje natural, análisis de texto, y diccionarios especializados para detectar, en qué medida, requerimientos de software escritos en inglés cumplen o no con las propiedades deseadas.

Los resultados obtenidos muestran, por un lado, que es factible crear herramientas de software que faciliten la tarea de revisión de requerimientos de forma automática. Además, dichas herramientas producen resultados satisfactorios. Por otro lado, mediante los experimentos se notaron algunas deficiencias en cuanto a la forma en que se realizan las evaluaciones cuando no se tiene el apoyo de una herramienta. Estos hallazgos se presentan con detalle y se ofrecen posibles acciones preventivas y correctivas para cada caso.

Abstract

Software requirements are an essential part of the software life cycle. A properly definied software requirements specification increases the likelihood of producing software that satisfies user's needs.

Particularly in the avionics industry, most embedded systems are considered mission critical systems. As a result, the software development process is particularly rigorous in every phase of the software life cycle. In order to guarantee a well defined requirements specification, DO-178B, a widely used standard for software development in the avionics industry, describes a number of properties that software requirements must exhibit to satisfy specific quality objectives.

This research explores a simple approach towards partial automation for the task of verifying that software requirements written in natural language satisfy three of the properties included in DO-178B: non-ambiguity, precision and verifiability.

This work provides a bounded definition for each of the three concepts, considering the main characteristics that requirements must exhibit in order to satisfy those quality objectives. A software prototype was created that combines natural language processing techniques and specialized dictionaries to examine software requirements written in English with the goal of identifying whether or not they satisfy the desired properties.

Experimental results show that it is possible to create tools that support the process of reviewing software requirements automatically, producing satisfactory results. Furthermore, some of the experiments revealed specific deficiencies in the process of reviewing requirements manually; that is to say, without using any tool as an aid. Findings are described in detail, along with possible preventive and corrective recommendations when applicable.

Índice de Tablas

4.1 Objetivos del estándar DO-178B para la fase de diseño de software. 21

4.2 Niveles de certificación según el estándar DO-178B. 22

4.3 Aspectos a observar para determinar precisión en un requerimiento. 30

4.4 Concepto de matriz léxica utilizado en WordNet. 39

4.5 Ejemplos de clases verbales en VerbNet. 41

5.1 Sistemas utilizados para realizar la investigación. 43

5.2 Información de salida del prototipo. 45

5.3 Objetivos en DO-178B considerados en la investigación. 45

5.4 Lineamientos en DO-178B considerados en el estudio. 46

5.5 Vínculo entre conceptos estudiados y DO-178B. 47

7.1 Elementos de ejemplo para acotar el concepto de *verificabilidad*. 56

7.2 Lista general de elementos para determinar *Imprecisión*. 60

7.3 Lista general de elementos para determinar *Ambigüedad*. 61

7.4 Lista general de elementos para determinar *No-verificabilidad*. 62

7.5 Elementos utilizados en el prototipo. 63

7.6 Coincidencias entre el autor y los profesionales para la evaluación de elementos. 66

7.7 Coincidencias en la clasificación de elementos. 67

7.8 Clasificación de elementos propuesta por el autor. 71

7.9 Resultados de la clasificación de elementos. 72

7.10 Elementos y puntajes en $\{\Lambda, \Gamma, \Upsilon\}$. 74

8.1 Ejemplos de expresiones regulares para capturar requerimientos. 80
8.2 Escenarios de prueba para extracción de requerimientos. 81
8.3 Descripción de la máquina de desarrollo y pruebas. 94

9.1 Sistemas fuentes para obtención de datos de prueba. 96
9.2 Distribución de tipos de requerimientos por grupo. 100
9.3 Resumen de coincidencias entre los evaluadores y la herramienta. 104
9.4 Resumen de coincidencias entre profesionales. 105
9.5 Comparación de concidencias profesionales-herramienta de forma individual. . 106

A.1 Elementos sintácticos utilizados en *Penn Treebank*. 123

Índice de Figuras

2.1 Ejemplo de requerimiento de software y su formulación en lógica de primer órden. 10

4.1 Un ejemplo de ambigüedad referencial. 25
4.2 Partes básicas de un requerimiento de software. 27
4.3 Árbol de derivación sintáctica. 34
4.4 Ejemplo de POS-tagging aplicado a una oración. 36
4.5 Ejemplo alternativo de POS-tagging. 36

7.1 Diagrama del instrumento de evaluación de elementos. 65

8.1 Diagrama de flujo general en el prototipo. 76
8.2 Diagrama del proceso de normalización de requerimientos. 78
8.3 Esquema de representación de requerimientos. 79
8.4 Diagrama de flujo del análisis de elementos en el prototipo. 83
8.5 Diagrama de clases para el prototipo. 86
8.6 Pantalla de ayuda principal en el prototipo. 88
8.7 *Executive report (percentages)*. 89
8.8 *Executive report (scores)*. 90
8.9 *Detailed report with scores*. 90
8.10 *Unsatisfied elements (summary)*. 91
8.11 *Unsatisfied elements (descriptions)*. 91
8.12 *Unsatisfied elements (w/ requirements)*. 92

1

Introducción

La etapa de levantamiento de requerimientos es fundamental en el ciclo de vida del software. Existen diversas razones que justifican esta afirmación, dentro de las que se pueden citar las siguientes:

- En general se ha observado que los requerimientos de software escritos de manera vaga o pobre resultan en aplicaciones de software que no funcionan correctamente [32].
- A menudo es durante la etapa de levantamiento de requerimientos de software cuando se logra dimensionar verdaderamente el sistema que se desea construir [21]. Por eso, dedicar suficiente esfuerzo en esta etapa ayuda a concebir y planificar adecuadamente el software que se desea crear.
- La práctica ha demostrado que el costo de corregir un error en un requerimiento de software crece exponencialmente según el momento en que se detecte dicho error durante el ciclo de vida del software [18, 10]. Es decir, corregir un error en un requerimiento durante la etapa de levantamiento de requerimientos es mucho menos costoso que corregir el mismo error durante la etapa de pruebas.

Sumado a lo anterior, no es suficiente escribir muchos requerimientos sino que también es necesario establecer técnicas efectivas que garanticen que dichos requerimientos sean claros y comprensibles. Así, es indispensable validar dichos requerimientos antes de pasar a la siguiente etapa del ciclo de vida del software y, en general, realizar evaluaciones periódicas para identificar posibles conflictos que se detecten en los requerimientos durante las siguientes etapas del ciclo de vida.

En el caso particular de los sistemas empotrados en aviónica, las restricciones que deben cumplir los requerimientos de software son particularmente rigurosas debido al alto nivel de

seguridad y disponibilidad que deben demostrar las aplicaciones durante las operaciones de vuelo.

La certificación de software para sistemas empotrados en aviónica es regulada por entes especializados [1] que utilizan estándares de desarrollo y verificación como el DO-178B (ver sección 4.2 [30]). Dicho estándar es uno de los más estrictos que existen en cuanto a calidad respecta y entre sus lineamientos se describen ciertas propiedades que los requerimientos de software deben cumplir para satisfacer los objetivos de diseño que el estándar establece.

Con la intención de cumplir con los objetivos del DO-178B, una de las estrategias es diseñar herramientas que ayuden a mejorar los procesos de revisión de requerimientos. Contar con elementos de apoyo que ayuden a automatizar estos procesos es muy útil, pues potencialmente ayudan a reducir los costos y el tiempo invertido en estas tareas durante un proyecto. La oferta de herramientas de este tipo en el mercado es aun nula, pues todavía se realiza investigación al respecto. Sin embargo, cuando sus efectos positivos se den a conocer con más fuerza en el corto o mediano plazo, es posible que la oferta sea más favorable.

Como se explicará en detalle más adelante, en esta investigación se trata de automatizar parcialmente la tarea de verificar que requerimientos de software para aviónica escritos en lenguaje natural cumplan con ciertas propiedades que el DO-178B establece. Estas propiedades representan criterios de *calidad* que los requerimientos deben cumplir para satisfacer los objetivos que dicho estándar plantea específicamente para la fase de diseño de software.

1.1 La Calidad en el Software

Cuando se habla del tema de *calidad*, es importante definir lo que se entiende por dicho término dentro del marco en el cual se aplica. Se dice usualmente que un producto, servicio o proceso posee (o carece de) un cierto grado de calidad. Sin embargo, así descrito, el concepto de calidad queda en el plano subjetivo, permitiendo múltiples interpretaciones -quizá igualmente válidas- sobre las cualidades reales del producto, servicio o proceso en cuestión.

[1] Por ejemplo la *Federal Aviation Administration* (FAA) en E.U., las *Joint Aviation Authorities* (JAA) en Europa y la *Transport Canada Aviation* (TCA) en Canadá.

El área de desarrollo de software no es la excepción pues existe mucha investigación, desde diversas ópticas, tratando de responder a la pregunta *¿qué es calidad del software?* Sin embargo, Milicic [7] sugiere que dicha investigación suele converger hacia un punto tal que *calidad del software* se puede entender principalmente de dos maneras:

1. Conformidad con una especificación: calidad en función de que las características (medibles) de un producto o servicio estén en conformidad con una especificación dada.

2. Satisfacción de necesidades de usuario: calidad definida como la capacidad del producto o servicio para satisfacer las necesidades del usuario.

Con la intención de reconocer la importancia de la creación de requerimientos y tratando de que dicho proceso sea efectivo, se tratará de aplicar los principios anteriormente descritos a los requerimientos de software. Es decir, si se quiere minimizar la cantidad de errores en los requerimientos de software en una etapa temprana del ciclo de vida, se puede modelar el problema como un problema de búsqueda de calidad.

Nótese que es posible aplicar las definiciones de Milicic a los requerimientos de software ya que, la especificación contra la cual se valida la conformidad de los requerimientos corresponde a un estándar de desarrollo de requerimientos que describe las propiedades que dichos requerimientos deben cumplir para lograr una descripción clara y correcta del software.

Adicionalmente, es posible suponer que al validar que los requerimientos de software cumplen con una especificación, la fase de codificación del software tendrá mayores posibilidades de éxito para construir un producto que satisfaga las necesidades del usuario.

Considerando esta perspectiva sobre el tema de *calidad del software*, se puede resumir el tema de esta investigación como la automatización parcial de la primera definición de Milicic (conformidad con una especificación) aplicado a requerimientos de software escritos en lenguaje natural para sistemas desarrollados bajo el estándar DO-178B [2][30].

[2]En la sección 4.2 se describe brevemente dicho estándar y la sección 5.2 explica sobre los lineamientos aplicables a requerimientos de software cuyo proceso de validación se pretende automatizar.

1.2 Verificación de Requerimientos de Software

En el ámbito de desarrollo de software, hay dos tareas importantes que realizar: *verificación* y *validación* de software. *Validación* se refiere a la tarea de revisar y probar que el software satisface las necesidades del usuario. Es decir, "validar el software" es probar que este realiza las funciones que se esperaban. Por otro lado, *verificación* se refiere a la tarea de revisar y probar que el software cumple con la especificación establecida. Es decir, "verificar el software" es probar que este hace únicamente aquello para lo cual fue creado y, además, comprobar que lo hace correctamente.

Ahora, en el caso de los requerimientos de software, "validar" los requerimientos significa probar que el conjunto de requerimientos que describen un software es completo. Es decir, no faltan ni sobran requerimientos. Por otro lado, "verificar" los requerimientos significa probar que estos son correctos. Es decir, que estos cumplen con algunas restricciones de calidad como las que se estudian en esta investigación. Por lo tanto, en esta investigación se propone una herramienta que apoya el proceso de verificación de requerimientos y no el proceso de validación.

Antes de profundizar en la idea de automatizar el proceso de verificación de requerimientos, se describen algunos aspectos sobre el proceso de creación y revisión de requerimientos de software escritos en lenguaje natural. De acuerdo con la experiencia profesional del autor, esta labor se realiza generalmente de forma manual, es decir, sin ayuda de ninguna herramienta de software que realice parte de dicho proceso de revisión[3].

Verificar que un conjunto de requerimientos de software cumpla con los lineamientos que establece el estándar DO-178B es una tarea difícil cuando se realiza sin ayuda de herramienta alguna. Entre las razones que dificultan dicha tarea se encuentran:

- El proceso requiere un dominio importante de aspectos lingüísticos (tales como gramática

[3]En general esta tendencia es de esperarse cuando se usa el lenguaje natural para describir los requerimientos de software. Una de las razones es que, a diferencia de los lenguajes formales (estructurados y no-ambiguos), donde se pueden construir herramientas de bajo costo para automatizar el proceso de revisión, se necesita una inversión significativa en investigación y desarrollo para crear herramientas sólidas que puedan reemplazar a una persona en el análisis de requerimientos de software en lenguaje natural. Por ende, sobra mencionar que para muchos proyectos en desarrollo de software comercial, tal inversión sería un verdadero lujo.

y semántica) y técnicos al mismo tiempo, lo que sugiere la participación de personal calificado en ambas áreas para detectar con eficacia las posibles fallas en los requerimientos.

- No existe certeza de que dos o más expertos humanos produzcan el mismo resultado tras un análisis del mismo conjunto de requerimientos de software ya que, en general, hay espacio para diversas interpretaciones debido a la naturaleza informal del lenguaje natural.

- El proceso es propenso a errores humanos debido a factores naturales como fatiga, distracción y aburrimiento.

- El proceso consume una cantidad de tiempo significativa, lo que implica un costo elevado.

A menudo se subestima el tiempo y esfuerzo que se debe invertir en la fase de verificación de requerimientos. Además, en proyectos con restricciones de tiempo severas, se debe bajar la rigurosidad de los análisis, cubriendo principalmente las deficiencias de mayor prioridad. En consecuencia, las deficiencias que no fueron corregidas en los requerimientos pueden fluir peligrosamente hacia otras etapas del ciclo de vida del software.

Una manera de resolver el problema, de modo parcial[4], es aplicar técnicas computacionales de análisis de texto para detectar la presencia de aspectos que produzcan disconformidad entre un requerimiento de software y algunos de los lineamientos establecidos en DO-178B (ver sección 5.1). Se cree que mediante las técnicas actuales de procesamiento de lenguaje natural no es posible resolver el problema completamente; sin embargo, un enfoque parcial podría producir beneficios significativos tales como:

- El conocimiento lingüístico y técnico se traslada al sistema una única vez y tiene caracter acumulativo[5]; es decir, se puede agregar más conocimiento. Lo anterior reduce la dependencia de personal altamente calificado en dichas áreas.

[4]En la sección 4.4.1 se justifica el hecho de que no se pretenda resolver el problema de forma completa en esta investigación.

[5]Este asunto de trasladar conocimiento a un sistema tiene la desventaja de que se puede caer en el error de considerar un problema desde un único punto de vista, brindando una única solución que puede dejar por fuera otras soluciones igualmente válidas.

- Se pueden reproducir los resultados para un mismo conjunto de requerimientos de software de entrada, lo que reduce el riesgo de inconsistencias y agrega confiabilidad al resultado.

- Se podría reducir el tiempo de análisis significativamente.

La siguiente sección describe aspectos generales sobre la naturaleza del trabajo realizado en esta investigación.

1.3 El Trabajo Realizado

En esta investigación se aplican técnicas de procesamiento de lenguaje natural y manejo de texto para detectar la presencia de *imprecisión*, *ambigüedad* y *no-verificabilidad* en un conjunto de requerimientos de software dada una definición previa de cada concepto. El enfoque explorado en esta investigación es parcial, puesto que se tratan los conceptos de manera restringida, sin pretender resolver el problema para cualquier interpretación de dichos conceptos.

No se tratará el tema general de análisis de requerimientos de software escritos en lenguaje natural para todo tipo de sistemas. Se discutirá únicamente sobre detección automática de *imprecisión*, *ambigüedad* y *no-verificabilidad* (dada una definición específica de cada uno de dichos conceptos) en requerimientos de software para sistemas empotrados en el área de aviónica.

Los requerimientos de software utilizados en esta investigación están escritos en inglés, por lo que las técnicas utilizadas corresponden a aspectos específicos de este idioma. Sin embargo, se espera que las técnicas a aplicar puedan ser reutilizadas (con un esfuerzo moderado) en futuras investigaciones particularmente con el idioma español.

El cuerpo del documento se estructura de la siguiente manera. El capítulo 2 describe algunos de los trabajos conocidos en el área. Seguidamente, el capítulo 3 presenta el objetivo principal y los objetivos específicos de la investigación. El marco teórico se presenta en el capítulo 4. Debido a la naturaleza de los temas descritos allí, la sección 4 cubre los temas de manera general esperando que el lector encuentre información más rica en la literatura ampliamente disponible

para cada tema. Luego, el capítulo 5 se ocupa de describir el alcance y las limitaciones de la investigación.

En los capítulos del 6 al 8 se explica el enfoque metodológico seguido para cumplir con los objetivos planteados. El capítulo 9 describe aspectos de diseño y características de los experimentos realizados como mecanismo de evaluación. Este mismo capítulo describe también los resultados de los experimentos, e incluye una discusión sobre dichos resultados. El capítulo 10 incluye una discusión general sobre la investigación y finalmente, el capítulo 11 lista las conclusiones de la investigación así como algunas ideas para trabajo futuro.

2

Antecedentes y Trabajo Relacionado

Al examinar la literatura disponible en torno al tema de análisis de requerimientos, es posible distinguir al menos dos líneas de estudio complementarias. La primera estudia y propone metodologías y prácticas que apoyan los procesos manuales de creación y revisión de requerimientos de software. Así, esta línea apoya el proceso mediante la capacitación de los actores humanos involucrados, ya sea en la redacción o la revisión técnica de requerimientos. Utilizaremos el término "análisis manual" para referirnos a los trabajos en esta primera línea. Por otra parte, la segunda línea se enfoca en la implementación de métodos y herramientas computacionales que reemplacen —en cierta medida— la tarea de los actores humanos. En este sentido, la segunda línea se basa en la aplicación de técnicas de procesamiento de lenguaje natural para automatizar ciertas tareas en el proceso de requerimientos. Utilizaremos el término "análisis automatizado" para referirnos a los trabajos en esta segunda línea.

A continuación se describen los trabajos más notables en el área de análisis manual de requerimientos de software de los que se tiene conocimiento al momento.

Fuchs y Schwitter [9] utilizan *Attempto Controlled English* (ACE) para escribir especificaciones de software en un subconjunto reducido del idioma inglés. Los requerimientos escritos pueden ser interpretados por un computador de manera precisa y eficiente. Al mismo tiempo, ACE es suficientemente expresivo para facilitar su aplicación por parte de usuarios no especialistas en creación de requerimientos de software. Una especificación escrita en ACE se traduce a una representación en lógica de primer orden (LPO) y a un programa de Prolog (ver [26]) mediante el formalismo de *Definite Clause Grammar* (DCG). De forma resumida, ACE posee una gramática restringida y un léxico ajustable según el dominio de aplicación. Un texto escrito en ACE posee oraciones declarativas, de la forma *sujeto-verbo-predicado*, oraciones condicio-

nales de la forma *if-then*, y oraciones interrogativas de la forma *yes/no queries* o *wh-queries*. Además, ACE permite el uso de anáforas, oraciones subordinadas, comparativas, frases elípticas compuestas (listas unidas por las conjunciones *and* u *or*) y negación. Considere el ejemplo de la Figura 2.1.

Requerimiento en Lenguaje Natural

Every matrix shall have a NVM space which stores default init data for that matrix.

Formulación en Lógica de Primer Órden

```
all(A,matrix(A) =>exists(B,nvmspace(B) & exists(C,defaultdata(C)
& stores(B,C) & for(C,A))))
```

Figura 2.1: Ejemplo de requerimiento de software y su formulación en lógica de primer órden. La parte superior de la figura muestra un ejemplo de un requerimiento de software expresado en lenguaje natural. En la parte inferior de la figura se muestra la formulación en LPO para el mismo requerimiento.

El principal aporte de Fuchs y Schwitter es que facilita la tarea de producir especificaciones formales de requerimientos para usuarios no especializados. La ventaja más notoria es que la especificación formal puede ser evaluada con un método matemático para detectar inconsistencias o conflictos en los requerimientos sin que medie la subjetividad e indeterminismo del lenguage natural. Nótese como el enfoque propuesto por Fuchs y Schwitter es muy apropiado para comenzar a escribir los requerimientos de software desde el inicio. Sin embargo, en esta investigación no se ha utilizado esta técnica debido a que los requerimientos de software ya fueron escritos previamente en lenguaje natural.

Firesmith [8] sugiere que primero se debe comprender ciertos principios elementales sobre cómo expresar requerimientos de software en lenguaje natural antes de ser capaz de escribir requerimientos de forma clara y correcta. Para ello propone una lista de preguntas que, aunque no es exhaustiva, considera aspectos claves que los usuarios deben aplicar al escribir requerimientos. El objetivo es que el documento que contiene la especificación de requerimientos cumpla con propiedades como cohesividad, consistencia, completitud, viabilidad, no ambigüedad, verificabilidad y otras. Hay dos diferencias principales entre el trabajo de Firesmith y el realizado

en esta investigación. Primero, en la presente investigación se tratan únicamente tres conceptos: *precisión*, *ambigüedad* y *verificabilidad*. Segundo, el trabajo de Firesmith se ubica dentro de la línea de estudio de análisis manual, mientras que en esta investigación se explora un enfoque de análisis automatizado.

En su trabajo, Hooks [12] asume una posición similar a Firesmith, alegando que a menudo los errores en los requerimientos surgen como resultado de la falta de experiencia y conocimiento de los actores humanos. Su contribución describe algunas características que —él sugiere— deben tener los requerimientos; además, utilizando ejemplos, describe algunos de los problemas más comunes en requerimientos de software así como posibles maneras de evitarlos. La principal diferencia entre el trabajo de Hooks y el realizado en esta investigación es que Hooks no utiliza ninguna herramienta para analizar requerimientos. Por el contrario, su trabajo trata de mejorar el conocimiento de las personas que escriben requerimientos para que estas lo hagan de forma adecuada.

Berry *et al.* [2] exponen con amplio nivel de detalle el tema de ambigüedad en relación con tres áreas específicas: lingüística, ingeniería de software y derecho. En este trabajo se hace énfasis en problemas derivados del uso constante de vocablos que aparentan ser sencillos y que en realidad merecen un trato complejo y cuidadoso. Algunos de los vocablos que se discuten son: *all*, *each*, *every*, *only*, *also*, *even* y algunos otros más. El objetivo es que los actores adquieran mayor destreza tanto para escribir requerimientos como para realizar inspecciones técnicas certeras. A diferencia de la presente investigación donde se tratan tres conceptos (*precisión*, *ambigüedad* y *verificabilidad*), el trabajo de Berry *et al.* trata únicamente el tema de ambigüedad. Sin embargo, en esta investigación se ha utilizado el trabajo de Berry *et al.* como una fuente de información importante para el tema de ambigüedad en requerimientos de software.

Luego de introducir algunos trabajos relacionados con el análisis manual de requerimientos de software, se describen a continuación los trabajos de los cuales se tiene conocimiento en el área de análisis automatizado de requerimientos de software.

Gervasi y Nuseibeh [11] reportan los resultados de un estudio que utiliza métodos no formales para validación de especificaciones de requerimientos de software. Básicamente su enfoque

busca validar propiedades en modelos que se obtienen mediante técnicas de *shallow parsing*[1] y uso de herramientas, como Circe [11], que facilitan el proceso de análisis de los requerimientos de software. La presente investigación se diferencia del trabajo de Gervasi y Nuseibeh significativamente ya que, en nuestro caso, se realiza un análisis de los requerimientos de software como tal, mientras que en su caso se realiza un análisis sobre un modelo del sistema que se construye a partir de los requerimientos de software.

Wilson *et al.* [32] también estudian el tema de calidad en análisis de requerimientos de software en lenguaje natural. En su trabajo desarrollan una herramienta llamada ARM que implementa un modelo compuesto por atributos e indicadores de calidad. Los atributos son propiedades con que debe cumplir la especificación de requerimientos y se evalúan mediante indicadores o elementos sintácticos específicos que ayudan a medir el grado de cumplimiento de cada requerimiento con los atributos deseados. De este modo, la herramienta es capaz de encontrar defectos y producir estadísticas sobre un documento. La principal diferencia entre ARM y la herramienta que se construyó en esta investigación es que la primera (ARM), aparte del texto de los requerimientos, también considera otros aspectos de formato y diseño que debe poseer el documento de especificación de requerimientos. La herramienta que se construyó en esta investigación utiliza únicamente el texto de cada requerimiento de forma aislada sin considerar aspectos propios del documento de especificación de requerimientos.

Finalmente, Lami *et al.* [17] aseguran que la disponibilidad de herramientas automatizadas para el análisis de calidad en requerimientos de software en lenguaje natural es un factor clave para lograr software de calidad. En su trabajo se expone una metodología para analizar requerimientos de manera sistemática encontrando errores potenciales debidos a ambigüedad, inconsistencia o no completitud. La herramienta llamada QuARS (*Quality Analyzer of Requirement Specifications*) implementa la metodología propuesta y figura como una contribución importante en el área. La diferencia principal entre esta investigación y el trabajo de Lami *et al.* es que en esta investigación se restringe el dominio a requerimientos de software para sistemas empo-

[1] *shallow parsing* se refiere a métodos no formales de análisis de texto que logran resultados potencialmente parciales sobre la estructura sintáctica de un texto que se pretende parsear (ver sección 4.4.1). Generalmente se implementa tratando de asociar segmentos de una oración con patrones sintácticos ya definidos. De este modo, patrones sintácticos mal formados o poco comunes no evitan que se logre un resultado parcial durante el proceso de parsing.

trados en aviónica. La segunda diferencia es que nuestra investigación se ocupa únicamente de tres propiedades: *precisión*, *ambigüedad* y *verificabilidad*.

El trabajo de Lami *et al.* sugiere que los estudios relacionados con el análisis de requerimientos (tanto manual como apoyado por herramientas) se pueden clasificar en tres grupos según el tipo de objetivos que buscan: (a) restrictivos, (b) inductivos y (c) analíticos.

Los estudios de tipo restrictivo se basan en la definición de reglas o técnicas que limitan el grado de libertad al escribir requerimientos de software en lenguaje natural, *p. ej.,* [9]. Por otro lado, los de tipo inductivo buscan los problemas que comúnmente aparecen en los requerimientos para proponer acciones correctivas o estilos de redacción, *p. ej.,* [8, 12, 2]. Finalmente, los de tipo analítico toman los requerimientos tal y como estos han sido creados y, mediante técnicas de procesamiento de lenguaje natural, tratan de identificar y corregir algunos de los defectos, *p. ej.,* [17, 11, 32].

Al momento de desarrollar esta investigación, no se tiene conocimiento de ningún esfuerzo paralelo específicamente en relación con el análisis de requerimientos de software para sistemas empotrados en aviónica que se desarrollan siguiendo el estándar DO-178B.

3

Objetivos

En la presente sección se describen los objetivos de la investigación. El apartado 3.1 describe de manera general el objetivo principal de la investigación y la sección 3.2 describe, de manera más puntualizada, cada uno de los objetivos específicos que se han planteado como parte de la investigación.

3.1 Objetivo Principal

El objetivo principal de esta investigación es diseñar e implementar un prototipo de software capaz de evaluar requerimientos de software escritos en lenguaje natural en inglés para sistemas empotrados en el área de aviónica, en términos de *precisión, no-ambigüedad,* y *verificabilidad.*

Se escogieron esos tres criterios principalmente por tres razones:

1. Están incluídos en los lineamientos que el estándar DO-178B [30] establece para los requerimientos de software.

2. Es posible automatizar parcialmente el proceso de verificar que los requerimientos de software cumplan con estos tres criterios utilizando recursos disponibles.

3. Son críticos y afectan directamente la calidad de los requerimientos de software.

Es importante mencionar que no se pretende reemplazar la opinión de los profesionales en el área de análisis de requerimientos de software con este prototipo. Por el contrario, se pretende que el prototipo sea visto como un sistema de apoyo a la toma de decisiones que necesita de la supervisión y monitoreo de un experto humano para validar los resultados que produce.

3.2 Objetivos Específicos

Los objetivos específicos de la investigación se listan a continuación:

1. Acotar los conceptos de *precisión, ambigüedad* y *verificabilidad* para ajustarlos al contexto de análisis de requerimientos de software escritos en lenguaje natural.

2. Describir al menos cuatro elementos lingüísticos que serán utilizados para descubrir indicadores de (no) cumplimiento de los requerimientos de software respecto de las propiedades definidas en el objetivo específico 1.

3. Diseñar e implementar un prototipo de software que tome una lista de requerimientos de software como entrada, examine los elementos descritos en el objetivo 2 y determine si estos requerimientos cumplen con las propiedades definidas en el objetivo específico 1.

4. Evaluar la efectividad del prototipo con base en los resultados obtenidos a partir de experimentos comparando los resultados del prototipo con los resultados producidos por profesionales en el área de análisis de requerimientos de software.

4

Marco Teórico

En este capítulo se repasan de forma muy breve algunos de los principales conceptos, recursos o tecnologías que serán utilizados en el cuerpo de la investigación. No se pretende describir los conceptos o técnicas de manera detallada en este capítulo; sin embargo, se describe cada parte tratando de brindar al lector una perspectiva general sobre las implicaciones de cada concepto en el marco del trabajo que se ha realizado.

Inicialmente se explica a modo introductorio lo que es el ciclo de vida del software y en particular se describe el rol que juegan los requerimientos de software dentro de ese ciclo de vida (ver sección 4.1). Seguidamente se presenta una breve introducción al estándar de desarrollo de software DO-178B en la sección 4.2, ya que dicho estándar es de particular interés en esta investigación. Luego, la sección 4.3 describe algunos aspectos esenciales sobre los conceptos de *precisión*, *ambigüedad* y *verificabilidad* ya que estos son conceptos claves en esta investigación. El apartado 4.4 explica el tema de procesamiento de lenguaje natural (PLN) dado que, en forma general, el trabajo gira en torno a esa área de estudio aunque de manera muy limitada. Sin embargo, debido a que este es un tema muy amplio, se pretende acotar el concepto para indicar con precisión cuáles aspectos de dicha rama de estudio son útiles en la presente investigación. Para llevar el tema de PLN a un nivel más práctico, se describen tres conceptos o técnicas importantes que serán usadas en esta investigación: *parsing* (sección 4.4.1) y *tagging* (sección 4.4.2). Finalmente, la sección 4.5 describe dos de los recursos lingüísticos disponibles en Internet y que se incorporan a la investigación: WordNet y VerbNet.

4.1 Ciclo de Vida del Software

En el ámbito de la ingeniería de software, se conoce como “ciclo de vida del software“ a los procesos que transcurren durante el desarrollo y utilización de un software. El ciclo comienza cuando surge una necesidad y se desarrolla un software, y se extiende por un lapso de tiempo durante el cual se pueden hacer cambios al software e inclusive llegar a un estado de obsolecencia. Las características de los procesos que se realizan durante el ciclo de vida dependen del modelo de desarrollo que se utilice.

Un modelo de desarrollo de software provee un grupo de conceptos y metodologías bien coordinadas que se necesitan para desarrollar software [10]. Existen en la literatura y en la práctica diversos modelos de desarrollo de software que se deben considerar dependiendo de las características particulares del proyecto que se plantea.

El modelo de desarrollo más general se conoce como *modelo del ciclo de vida de desarrollo de software* [10], también como *ciclo de vida clásico* e inclusive como el *modelo de cascada* [27]. Este modelo describe una secuencia lineal para las actividades del ciclo de desarrollo y es la base sobre la cual surgen otros modelos de desarrollo también conocidos en la industria tales como: el modelo basado en prototipos, el modelo tipo espiral, el modelo orientado a objetos [10], y el llamado *Rapid Application Development* (RAD) [27]. En esta investigación se hacen referencias al *modelo de cascada* ya que este es el modelo de desarrollo que más se ajusta al estándar DO-178B.

El modelo del ciclo de vida del software comprende siete etapas que Galin [10] describe de la siguiente manera:

1. Definición de Requerimientos: Se refiere a la descripción detallada de las características o funcionalidades que debe reunir el software que se desea construir. El cliente es quien define los requerimientos, pues a menudo el producto a crear es parte de un sistema más grande cuyas características ayudan a delimitar mejor el software que se desea crear.

2. Análisis: Se refiere al estudio general sobre las implicaciones que tienen los requerimientos para luego describir el modelo inicial del sistema.

3. Diseño: Consiste en definir las entradas, salidas, procedimientos, estructuras de datos, arquitectura de software y demás detalles sobre el software a construir.

4. Codificación: Se refiere a la transformación de la especificación de diseño hacia un programa en código fuente. Esta etapa debe involucrar la participación de los encargados de aseguramiento de la calidad.

5. Pruebas de Sistema: Consiste en realizar las pruebas pertinentes con el propósito de descubrir la mayor cantidad de errores de manera que se puedan corregir antes de pasar a la siguiente etapa.

6. Instalación: Es la etapa en que el software se comienza a utilizar en su respectivo ambiente de producción.

7. Uso y Mantenimiento: Se refiere al mantenimiento regular del software durante su vida útil, que puede ser de tres tipos: (a) correcciones - de errores identificados por los usuarios durante su uso; (b) adaptaciones - pequeños cambios orientados a satisfacer nuevos requerimientos; y (c) mejoras - agregando nuevas funcionalidades al sistema.

De acuerdo con Galin [10], la primera etapa en el *modelo de cascada* es la definición de requerimientos. Esta etapa es de particular interés en esta investigación ya que se construyó una herramienta que ayuda a verificar que dichos requerimientos satisfacen ciertos atributos de calidad. En esta investigación se adopta una definición propuesta por el *Institute of Electrical and Electronics Engineers* (IEEE) [3], según el cual, un *requerimiento de software* es:

(1) Una condición o habilidad requerida por un usuario para resolver un problema o cumplir un objetivo.

(2) Una condición o característica que debe cumplir un sistema para satisfacer un contrato, un estándar, una especificación o cualquier otro documento regulatorio.

(3) Una representación documentada de una condición como en (1) o (2).

Como describe Galin [10], algunos de los modelos de desarrollo basados en el modelo clásico son variaciones que presentan ciertas ventajas dependiendo del tamaño y la naturaleza del proyecto en que se utilicen. Explicar aquí los detalles de cada uno de dichos modelos escapa de los objetivos de la investigación; sin embargo, cabe destacar que de una u otra manera todos ellos incluyen una etapa en la que se crean los requerimientos de software. Los requerimientos son importantes ya que establecen el punto de partida para las condiciones que el software debe satisfacer y el comportamiento que debe mostrar cuando sea utilizado por los usuarios.

En el caso particular de esta investigación, la relación entre el modelo del ciclo de vida del software y el estándar DO-178B (ver sección 4.2) es muy cercana. Sin embargo, al estudiar el estándar DO-178B a profundidad, es notorio como el nivel de restricciones y formalismos impuestos en cada etapa es mucho más riguroso en el DO-178B que en otros modelos de desarrollo. Esta rigurosidad que el DO-178B agrega al modelo de desarrollo clásico se debe, en general, al nivel de criticidad tan importante que refleja el software para sistemas en aviónica. Algunos detalles sobre este modelo se explican en el siguiente apartado.

4.2 Estándar DO-178B

DO-178B, cuyo nombre completo es *Software Considerations in Airborne Systems and Equipment Certification*, es un estándar para desarrollo y certificación de software en la industria de aviónica creado por *Radio Technical Commission for Aeronautics, Inc.* (RTCA) y *European Organization for Civil Aviation Equipment* (EUROCAE) [29].

La importancia de DO-178B en la investigación se debe a que es precisamente este estándar el que se utiliza para certificar el tipo de software para el cual se pretende evaluar la calidad de los requerimientos.

DO-178B define cinco niveles de criticidad para el software a certificar. Cuanto más alto es el nivel, más riguroso es el proceso de certificación, lo cual se refleja en el número de objetivos de calidad que se deben satisfacer y las actividades que se deben llevar a cabo en el proceso. La tabla 4.1 muestra seis de los objetivos que contempla la Tabla A-4 propiamente en el anexo A del estándar [30]. Además, la tabla 4.1 no muestra ningún objetivo para software de nivel D.

Tabla 4.1: Objetivos del estándar DO-178B para la fase de diseño de software. La 1^{ra} columna muestra el número del objetivo. La 2^{da} columna describe el objetivo de calidad propiamente. Las columnas entre la 3^{ra} y la 6^{ta} indican si el objetivo se debe cumplir para el software según su nivel de criticidad. Una marca como **X** (a diferencia de **x**) indica que el objetivo se debe satisfacer con independencia, esto quiere decir que la asignación de miembros del equipo a los diversos objetivos debe respetar ciertas reglas descritas en el estándar.

Obj	**Descripción**	**Niveles**			
		A	**B**	**C**	**D**
1	*Low-level requirements comply with high-level requirements.*	**X**	**X**	**x**	
2	*Low-level requirements are accurate and consistent.*	**X**	**X**	**x**	
3	*Low-level requirements are compatible with target hardware.*	**x**	**x**		
4	*Low-level requirements are verifiable.*	**x**	**x**		
5	*Low-level requirements conform to standards.*	**x**	**x**	**x**	
6	*Low-level requirements are traceable to high-level requirements.*	**x**	**x**	**x**	

La tabla 4.2 muestra un resumen de los distintos niveles de certificación establecidos por DO-178B según el nivel de criticidad del software. La primera columna indica los 5 niveles de certificación que contempla el estándar. Cada aplicación de software obtiene su nivel dependiendo del impacto negativo que un fallo en esa aplicación produciría en la aeronave o sus tripulantes y pasajeros (segunda columna). Por ejemplo, cuando un error en el software representa una potencial catástrofe, ese software se considera de nivel A. Para citar un caso específico, un software que calcule la altura de la aeronave respecto del suelo tiene asociado un nivel de criticidad alto (posiblemente A), dado que un fallo en sus cálculos podría provocar un accidente durante un aterrizaje o un descenso. Por otro lado, un fallo en el software que controla el despliege de televisión para los pasajeros (aparte de la molestia de los mismos) no representa un riesgo para la aeronave y por eso tiene asociado generalmente un nivel D.

La tercera columna en la tabla 4.2 indica el número de objetivos de calidad que se deben satisfacer para lograr la certificación del software según su nivel. La cuarta columna indica el número de objetivos que se deben satisfacer con independencia en cada caso. Independencia significa que la asignación de miembros del equipo a los diversos objetivos debe respetar ciertas reglas descritas en el estándar. Por ejemplo, no se permite que el autor de un requerimiento participe en la revisión de dicho requerimiento o en la codificación. De manera similar, otra de las reglas indica que el autor de una prueba no puede tomar parte en la ejecución de la prueba.

Tabla 4.2: Niveles de certificación según el estándar DO-178B. La 1^{ra} columna lista los cinco niveles de criticidad que establece DO-178B. La 2^{da} describe el efecto de un fallo en ese nivel. La 3^{ra} muestra el número de objetivos en ese nivel y la 4^{ta} indica el número de objetivos que se deben satisfacer con independencia.

Nivel	Impacto de un Fallo	Objetivos	Con Independencia
A	*catastrophic*	66	25
B	*hazardous*	65	14
C	*major*	57	2
D	*minor*	28	2
E	*no effect*	0	0

El estándar describe una serie de procesos que se deben ejecutar durante un proyecto, indicando los objetivos que se deben cumplir y los artefactos que se deben producir durante cada etapa. Sin embargo, no incluye detalles sobre cómo se han de realizar los procesos. En este sentido, el estándar es muy flexible, permitiendo que aquellos que lo utilizan definan los detalles para la implementación de cada proceso.

De modo general, en el estándar DO-178B se distinguen cinco categorías o procesos: *Planning Process*, *Development Process*, *Verification Process*, *Configuration Management Process*, *Quality Assurance Process* y *Certification Liaison Process*. Los usuarios del estándar deben afinar estas categorías agregando subprocesos que persiguen objetivos afines en cada etapa. Por ejemplo, una subclasificación para el *Development Process* incluye: *Software Requirements Process*, *Software Design Process*, *Software Coding Process* y *Software Integration Process*.

En DO-178B, los requerimientos de software se escriben dentro del *Development Process*, específicamente dentro de los subprocesos de *Software Requirements Process* y *Software Design Process*. Los requerimientos que se escriben durante la etapa de *Software Requirements Process* se llaman requerimientos de alto nivel, pues describen acciones generales que debe realizar el software, pero sin adentrar en detalles sobre dichas acciones. Por otra parte, los requerimientos que se escriben durante la etapa de *Software Design Process* se conocen como requerimientos de bajo nivel, ya que estos suelen incluir más detalles sobre las acciones que debe de realizar el software. Los requerimientos de bajo nivel siempre surgen de un requerimiento de alto nivel y su papel es precisamente extender el contenido de los requerimientos de alto nivel.

Cada proceso, independientemente de su nivel de granularidad, debe contener condiciones de transición bien definidas. Es decir, se debe indicar las condiciones de entrada y condiciones de salida que se deben cumplir para avanzar entre cada etapa del proceso. En cada proceso además, se debe aportar evidencia objetiva de que se han alcanzado los objetivos correspondientes para esa etapa. Para pasar de la fase de diseño de software a la fase de codificación, es preciso contar con un documento que incluya la especificación del diseño y la arquitectura del software (a un nivel de detalle que permita la codificación) y registros de calidad que muestren que los requerimientos de software y la documentación de diseño han sido sometidos a revisión por equipos calificados y que dicho material satisface los objetivos del DO-178B para esa fase, así como las restricciones establecidos en el estándar respectivo para requerimientos de software (*Software Requirements Standard*) y para diseño y arquitectura de software (*Software Design Standard*).

En esta investigación se hace referencia con más frecuencia a la etapa de *Development Process* y específicamente a los subprocesos relacionados con la construcción de requerimientos de software: *Software Requirements Process* y *Software Design Process*.

La siguiente sección describe tres conceptos fundamentales que se utilizan en esta investigación como propiedades que los requerimientos de software (tanto los de alto nivel como los de bajo nivel) deben satisfacer de acuerdo con el estándar DO-178B y, en general, para cumplir con buenas prácticas de desarrollo de software. Estas tres propiedades son *no-ambigüedad*, *precisión* y *verificabilidad*.

4.3 Ambigüedad, Precisión y Verificabilidad

En las siguientes secciones se aborda de manera preliminar cada uno de los tres conceptos utilizados en esta investigación. Dichas secciones no incluyen la definición estricta de cada concepto (ver para esto la sección 7.2); más bien ofrecen una introducción a los aspectos principales que son considerados en el trabajo. Adicionalmente, se comenta sobre los recursos o las técnicas que fueron utilizadas para detectar cuándo los requerimientos de software incumplen con estos atributos. Se ofrecen además ejemplos de requerimientos de software que cumplen y otros que no cumplen con los atributos tal y como son vistos en la investigación.

4.3.1 Ambigüedad

La Real Academia Española de la Lengua señala que el término *ambigüedad* se utiliza principalmente para el lenguaje, indicando que una palabra o frase puede entenderse de varios modos o admitir distintas interpretaciones y dar, por consiguiente, motivo a dudas, incertidumbre o confusión [28].

Berry [2] distingue seis tipos de ambigüedad en los requerimientos de software[1]: léxica, sintáctica, semántica, pragmática, ambigüedad por vaguedad y ambigüedad por errores de lenguaje.

1. Ambigüedad Léxica: Una palabra tiene dos o más significados (es el caso de homonimia y polisemia). Por ejemplo, **log** puede representar tanto [el proceso de registrar datos en bitácora] como el [archivo de bitácora] en los siguientes requerimientos.

   - `The system shall display log activity when SHOW_LOG is true.`
   - `The system shall enable a log when in debug mode.`
   - `The system shall hide log information during test mode.`

2. Ambigüedad Sintáctica: Una secuencia de palabras tiene varias estructuras gramaticales válidas que representan distintos significados. Por ejemplo,

   - `The system shall enable the flying control mode.`
     `The system shall enable the flying control mode.`
   - `The system shall save A and B and C shall be cleared.`
     `The system shall save A and B and C shall be cleared.`

   En las oraciones anteriores, la ausencia de los paréntesis causaría ambigüedad de este tipo pues ambas agrupaciones de elementos son gramaticalmente correctas.

[1] No es cierto que sólo existen estos seis, existen otros tipos de ambigüedad que Berry también señala -dada la generalidad de este término- pero para efectos prácticos los seis que Berry trata como principales serán suficientes en esta investigación

3. Ambigüedad Semántica: Una oración acepta más de una forma de lectura aún cuando no presenta ambigüedad léxica ni sintáctica. Por ejemplo, en el siguiente requerimiento no se sabe con certeza cuántos *logs* debe guardar el sistema, uno por módulo o uno para todos los módulos [2].

   - `All system modules shall save a log.`

4. Pragmática: Una oración tiene distintos significados en el contexto en que se emite (no sólo el contexto lingüístico sino también situacional). Generalmente aparece como un problema de ambigüedad referencial [3] como en el ejemplo de la figura 4.1:

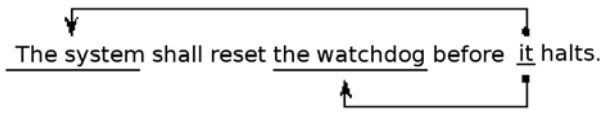

Figura 4.1: Un ejemplo de ambigüedad referencial.

   En la figura 4.1, el pronombre *it* funciona como una referencia válida tanto para el sustantivo *The system* como para el sustantivo *the watchdog*. Generalmente se resuelve este tipo de problemas mediante técnicas estadísticas que se basan en repositorios de datos cuyas fuentes de ambigüedad fueron previamente resueltas por expertos humanos.

5. Ambigüedad por Vaguedad: Una palabra es vaga cuando se tiene problemas para determinar la frontera de su significado. Generalmente suele estar presente en requerimientos no funcionales que describen *performance* y *throughput*, de forma no cuantificable, es decir, utilizando adjetivos como *slow*, *fast*, *high*, *low*, *readable*, *printable* y otros.

[2] En lógica de predicados, esta ambigüedad se debe a un problema de inclusión de cuantificadores, pues no es claro si **all** incluye a **a** o viceversa.

[3] Esto ocurre cuando una anáfora se puede servir de varios elementos válidos para tomar su referente.

6. Ambigüedad por Errores de Lenguaje: Inducida por uso incorrecto del lenguaje. Por ejemplo, es muy posible que alguien acepte las siguientes oraciones como correctas cuando no lo son:

   - Ejemplo 1 `Every module has their log file.`
   - Ejemplo 2 `The IO module shall write data when IOX equals 0x01 and MDDF equals 0xFF4 or IER0 equals 0xDD3.`

En el ejemplo 1, el error consiste en que algunos hablantes del idioma inglés tratan la estructura *every X* como plural, pero es singular. Note por ejemplo que el verbo sí se mantiene en su forma singular. Como consecuencia no se puede determinar con certeza cuántos archivos de bitácora (o *logs*) hay. El requerimiento permite pensar que hay un único *log* para todos los módulos o que hay tantos *logs* como módulos en una correspondencia uno a uno.

El ejemplo 2 es un caso muy común donde no se puede determinar con exactitud la condición que se debe cumplir para que se realice la acción princial ya que hay dos posibles interpretaciones: (a) [IOX==0x01 AND MDDF==0xFF4] OR [IER0==0xDD3], o (b) [IOX==0x01] AND [MDDF==0xFF4 OR IER0==0xDD3] y ambas son igualmente válidas según el planteamiento del requerimiento.

En esta investigación se hace mayor énfasis en los tipos de ambigüedad sintáctica, semántica y errores del lenguaje. Ambigüedad Léxica y Pragmática serán tratadas en menor grado, mientras que Vaguedad será tratada como parte del atributo de precisión.

Los recursos que se utilizarán para detectar los tipos de ambigüedad señalados incluyen los repositorios de WordNet [24] y VerbNet [13] así como el conocido Charniak parser de tipo *Probabilistic Context Free Grammar* (PCFG) puesto a disposición por Brown University [23]. También se toman algunas ideas de SPOT, un prototipo para un parser semántico puesto a disposición por University of North Texas [31]. Se utiliza también una técnica de *shallow parsing* llamada *word spotting* o *rule based multiword detection* que explica Nugues en [25].

En la siguiente sección se describe el segundo concepto que corresponde al término *precisión*.

4.3.2 Precisión

En el contexto de esta investigación, el término *precisión* se refiere al grado de concisión y exactitud del lenguaje utilizado para expresar un requerimiento de software.

En el caso de los requerimientos de software se busca precisión tanto en la forma como en el contenido.

- Forma: Indica si el requerimiento tiene sus partes básicas. Como mínimo en un requerimiento se debe distinguir entre condición y acción con suficiente detalle (figura 4.2.)

- Contenido: Indica si la condición y la acción descritas en el requerimiento tienen o no el detalle que se necesita según sea el tipo de requerimiento [4].

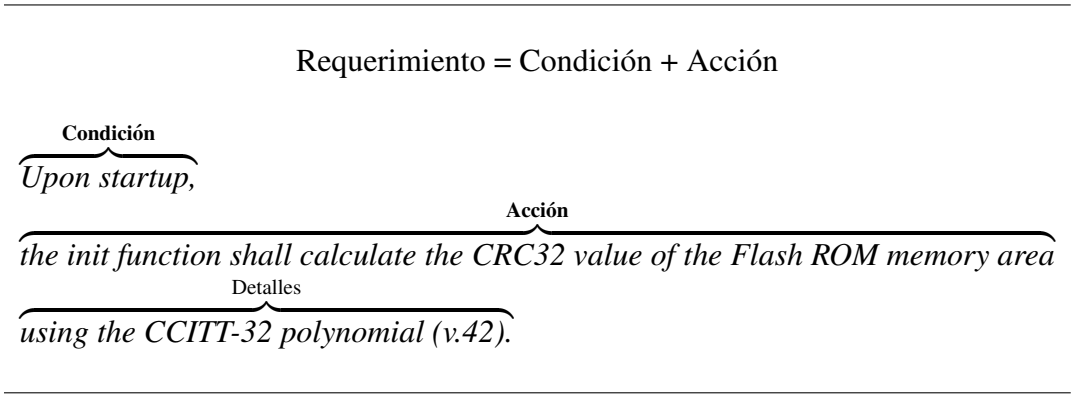

Figura 4.2: Partes básicas de un requerimiento de software. La figura muestra la separación que existe entre las partes principales de un requerimiento: condición, acción y detalles.

Cuando se considera la forma, se debe tomar en cuenta que los elementos condición y acción pueden ser oraciones mucho más complejas que las presentes en la Figura 4.2. Además, la condición puede aparecer antes, en medio o después de la acción.

[4] El nivel de detalle depende de la acción en el requerimiento, puede variar desde la inclusión de valores de tolerancia para medidas cuantificables hasta descripción de algorítmos que se necesitan para un proceso. Vale la pena notar que el nivel de detalle que se necesita para verificar un requerimiento de software es el mismo que se necesita para poder implementar dicho requerimiento en la fase de codificación.

Al ser la condición una oración subordinada, no es posible que un requerimiento de software tenga solamente la condición sin ninguna acción relacionada; sin embargo, lo contrario sí puede ser posible y se detectaría como un requerimiento de software que no es preciso. Es decir, se considera un requerimiento como impreciso cuando no se especifica una condición para la acción deseada [5].

A manera de ejemplo, se presentan a continuación algunos requerimientos de software y se discuten algunos problemas de precisión que estos presentan.

- Ejemplo 1 `The ARINC 717 Interface Component shall format ARINC 717 messages.`

El primer ejemplo presenta dos problemas fundamentales: (1) no se precisa cuándo o en qué momento debe ocurrir la acción (problema de forma), y (2) no se indica cuál es el formato que el sistema debe usar para los mensajes ARINC 717 (problema de contenido); es decir, se da por supuesto que dicho formato es conocido, lo cual no es válido (a menos que se especifique explícitamente en otro requerimiento). Ambas razones hacen que el requerimiento no sea preciso en su estado actual. La investigación propuesta se ocuparía del primer problema mas no del segundo, ya que este requiere de técnicas de otra índole para construir un modelo del sistema y realizar en este una verificación de consistencia y completitud.

- Ejemplo 2 `Event history logs shall be maintained in on board battery backed up RAM.`

Este otro ejemplo también tiene dos problemas principales (ambos de contenido): (1) no precisa cuánto tiempo se debe mantener el respaldo de datos protegido por batería, y (2), no indica qué cantidad de datos o eventos se debe poder respaldar. Con estas dos debilidades presentes tampoco es posible verificar el requerimiento rigurosamente dado que tanto el tiempo de respaldo que provee una batería como la cantidad de datos que se pueden almacenar son cantidades

[5] Excepto para procesos que por su naturaleza se saben permanentes (*e. g., watchdog process, interrupt handlers*), el hecho de no especificar una condición para una acción implica que dicha acción es de caracter permanente, es decir, que debe ocurrir en todo momento. Sin embargo, este no suele ser el caso y generalmente se necesita corregir dichos requerimientos.

finitas cuyo valor debe ser conocido (y no está presente). Nótese que la deficiencia de este requerimiento no es tan evidente y de hecho no podría ser detectada por la herramienta propuesta en esta investigación, ya que se requiere un análisis semántico para detectarla.

- Ejemplo 3 `The system shall analyze all data received from RS-232 from monitor processor.`

En el ejemplo 3, hay un problema de contenido ya que no se sabe con certeza cuál es el propósito de la acción "analizar". Esta acción puede entenderse al menos de tres maneras: (1) leer los datos que se reciben por el puerto serial verificando que al menos no sean nulos; (2) revisar la consistencia del formato de los datos contra una lista de formatos pre-establecidos; y (3) revisar el contenido de los datos a manera de inspección. Cada una de estas tres acciones genera un resultado distinto y cada uno es válido para una interpretación específica.

- Ejemplo 4 `ACE shall receive the audio data from the CMU.`

Este último ejemplo tiene un problema de forma ya que no indica cuándo se debe realizar la acción descrita. Esto crea la posibilidad de que la acción sea permanente; sin embargo, la herramienta que se construyó lo detectaría como defectuoso y generaría una señal para que un experto lo revise a fondo.

Además de estudiar los aspectos de forma, cuando se analiza el contenido de un requerimiento para sistemas empotrados, es recomendable considerar al menos los siguientes aspectos de fondo:

- Unidades: Consiste en poder determinar las unidades que correspondan cuando se tiene una magnitud física (*p. ej.,* temperatura, frecuencia, altura).
- Tolerancia: Consiste en poder identificar rangos de tolerancia cuando existen medidas para magnitudes físicas como tiempo, temperatura, presión y otras.
- Umbrales: Consiste en poder determinar fronteras o umbrales cuantificables en vez de restricciones expresadas mediante términos imprecisos tales como: alto, bajo, rápido, lento, frío, caliente, y otras.

Los aspectos que muestra la tabla 4.3 no son exhaustivos. Dicha tabla menciona algunos que comúnmente aparecen en los requerimientos de software y que se pueden expresar en términos cuantitativos. La estrategia a seguir es tratar de encontrar las apariciones de estas condiciones y revisar que las mismas estén descritas de manera clara y con valores de tolerancia cuando así corresponda.

Tabla 4.3: Aspectos a observar para determinar precisión en requerimientos de software para sistemas empotrados. La 1^{ra} columna indica el tipo de requerimiento que necesita precisión. La 2^{da} muestra algunos ejemplos en los cuales se marca con negrita el texto que hace que cada requerimiento de ejemplo sea preciso.

Categoría	**Ejemplos**
Tiempo	*- The system shall declare a* `PWR FAULT` *after the* `PWR FAIL` *signal becomes active* ***for 1s +/- 100 msec****.* *- The system shall remove turbulence alert when label 066 has not been received* ***for 10 seconds (+/- 1 second)****.*
Frecuencia	*- The system shall transmit label 271* ***at a rate of twenty Hz (+/- 5 ms)****.* *- The system shall run* `BIT` *once* ***every 5 seconds****.*
Umbrales	*- The system shall report an* `ANALOG FAULT` *if the analog reference voltage is* ***less than 2.1 volts****.* *- The system shall report a* `HEATER FAULT` *if the heater current is* ***less than 333 milliamps*** *when the heater has been on for at least 10 seconds (+/- 200 milliseconds).*

Finalmente, en cuanto a los recursos que se utilizarán para detectar imprecisión se cuenta con un parser puesto a disposición por Brown University [23]. Este parser ayudaría con la tarea de estudiar la forma de los requerimientos y con ello detectar problemas como ausencia de condiciones. Además, se utiliza la ya mencionada técnica de *word spotting* para analizar la manera en que se describen cantidades físicas y valores de tolerancia.

En la siguiente sección se describe el tercer concepto que corresponde al término *verificabilidad*.

4.3.3 Verificabilidad

Un requerimiento de software es verificable si se puede escribir y ejecutar una prueba para demostrar que el sistema se comporta tal y como se ha especificado en el requerimiento.

Se debe reconocer que en ocasiones es posible escribir casos de prueba para un requerimiento, pero no es posible implementar dichos casos de prueba. Algunas de las razones por las cuales esta situación puede ocurrir son las siguientes:

- No se cuenta con las herramientas (harware o software) adecuadas para implementar la prueba.
- Existe una restricción a nivel del software que revela un conflicto o inconsistencia entre requerimientos, y por lo tanto, impide la ejecución de la prueba.

En ambos casos, generalmente se detectan estas dificultades conociendo a fondo los requerimientos de software y considerando los aspectos técnicos del sistema y ambiente de verificación. Esta investigación no se ocupará de estos casos.

El otro escenario en el que se encuentran problemas de verificabilidad es cuando los requerimientos de software (generalmente debido a un mal uso del lenguaje) describen acciones que no se pueden atribuir a un programa de software. En este trabajo se prestará mayor atención a este tipo de casos. Por ejemplo, los siguientes requerimientos se detectarían como no verificables:

- Ejemplo 1 `The IO module shall only write its memory space.`

El ejemplo 1 es un error muy común donde se coloca el término *only* justo delante del verbo cuando la intención no es modificar o limitar el verbo. Este requerimiento sugiere que el módulo de IO únicamente debe escribir en su espacio de memoria; es decir, el módulo de IO no atiende peticiones de lectura o escritura, no hace validaciones de consistencia de datos, ni ninguna otra acción. En resumen, el módulo de IO solo sirve para escribir en su espacio de memoria. En este sentido, cualquier otro requerimiento que sugiera otra tarea para el módulo de IO estaría directamente en conflicto con este.

- Ejemplo 2 `The system shall always display cabin pressure in the upper left corner of the MFD.`
- Ejemplo 3 `The system shall never transmit signal from idle receivers into the common ARINC bus.`

Los ejemplos 2 y 3 tienen un problema de similar naturaleza entre ellos. Ambos requerimientos necesitarían de un ciclo infinito de tiempo para poder verificar las acciones *always* y *never*. Es posible que el autor del requerimiento no tenga la intención de que el software cumpla con esa funcionalidad tal y como está descrita (infinita) pero esto indudablemente crea problemas de verificabilidad en un entorno de pruebas riguroso.

Existen varios recursos que se pueden utilizar para detectar problemas de verificabiliad. Por ejempplo, para detectar problemas de mala utilización de verbos se utiliza WordNet (sección 4.5.1) y VerbNet (sección 4.5.2). Para detectar acciones no acotadas en el tiempo se utilizan técnicas de análisis léxico. También se utilizan estas técnicas para detectar casos en los que los requerimientos no indican unidades, valores de tolerancia ni umbrales cuantificables para magnitudes numéricas.

La siguiente sección introduce el tema de procesamiento de lenguaje natural así como su importancia en relación con los objetivos de este trabajo.

4.4 Procesamiento de Lenguaje Natural

El tema de procesamiento de lenguaje natural es muy amplio. En esta investigación se utiliza una de sus ramas y con la intención de acotar de manera explícita el tratamiento que se hará de este tema, se presenta en las siguientes líneas un breve resumen de los aspectos a considerar.

El procesamiento de lenguaje natural (PLN) es una rama de la lingüística computacional que busca mecanizar las facultades humanas para producción y comprensión del lenguaje [25]. En lingüística existen varias subdisciplinas que se enfocan en un objeto de estudio que varía desde el nivel de sonidos hasta el nivel de significados [25], y el procesamiento computarizado de cada nivel requiere técnicas y herramientas distintas.

Según Nugues [25], un primer nivel de análisis lingüístico es la fonética, que estudia la producción y percepción de sonidos que forman el habla. Un segundo nivel es la morfología que estudia la estructura y las formas del léxico. Las reglas morfológicas permiten que las palabras raíces se puedan modificar para dar origen a todo el vocabulario del idioma. Un tercer nivel lingüístico es la sintaxis, que estudia las categorías y función de las palabras en las oraciones. Un cuarto nivel es la semántica que estudia el significado de las palabras y oraciones. En la práctica computacional, el estudio de la semántica se orienta a determinar el sentido "correcto" de una palabra en un contexto o la representación de una oración en un formalismo lógico [25]. Finalmente, dos niveles superiores son la pragmática y el discurso que se ocupan de las variaciones semánticas de las oraciones cuando forman parte de una situación específica y generalmente dentro de un diálogo.

Esta investigación utiliza aspectos morfológicos, sintácticos y semánticos, ya que estos facilitan la tarea de revisión de requerimientos según los conceptos discutidos en 4.3.

A continuación se describen dos técnicas específicas relacionadas con el procesamiento de lenguaje natural. La primera es la técnica de *parsing* en la sección 4.4.1 y la segunda es la técnica de *tagging* en la sección 4.4.2.

4.4.1 Parsing

Se utiliza el término *parsing* para referirse al proceso de análisis sintáctico (asistido por computadora) durante el cual se examina una secuencia de palabras (tokens) para determinar si su estructura sintáctica es aceptada por una gramática dada. A su vez, una gramática es una descripción precisa de un lenguaje formal que —mediante un conjunto de reglas de derivación o reescritura— especifica cuáles secuencias de símbolos o palabras constituyen frases válidas para dicho lenguaje.

Para un computador, la manera de encontrar señales de ambigüedad en una oración es tratando de parsear dicha oración. Dado que encontrar ambigüedad en los requerimientos de software es una de las metas de la investigación, esta sección resume los conceptos básicos de la técnica de *parsing* y su relación con el término *ambigüedad* en esta investigación.

Mediante el proceso de *parsing* se obtiene la representación sintáctica del texto que ha sido analizado. A dicha estructura se le conoce como árbol de derivación sintáctica, *parse* o *parse tree* [25]. Los árboles de derivación sintáctica son importantes porque muestran el orden jerárquico de los elementos que componen una oración. Sin embargo, su importancia no yace solamente en el plano visual sino que también son importantes para determinar errores sintácticos en una oración, los cuales pueden generar múltiples interpretaciones para un mismo texto (como se explica más adelante).

La figura 4.3 muestra un árbol de derivación sintáctica para una oración simple de la forma $Noun\ Phrase{+}Verb{+}Direct\ object$, en castellano $Sujeto{+}Verbo{+}Complemento\ directo$.

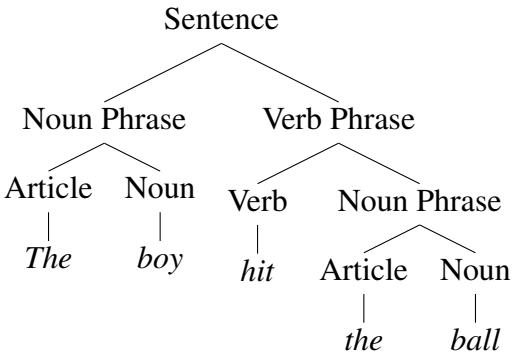

Figura 4.3: Árbol de derivación sintáctica para la oración *The boy hit the ball*. Este árbol muestra la forma de la oración luego de ser descompuesta en varios niveles. El primer nivel contiene toda la oración (*Sentence*). El segundo nivel se divide en dos partes, sujeto (*Noun Phrase*) y predicado (Verb Phrase). El tercer y cuarto nivel muestran la categoría sintáctica de cada elemento en la oración.

Los lenguajes formales pueden variar significativamente respecto del nivel de restricciones que establecen en su gramática para describir la estructura sintáctica válida para sus oraciones. Por ejemplo, en los lenguajes de programación, cada frase (o sentencia) debe tener una única interpretación, es decir, un solo *parse*. De este modo, el comportamiento del computador —al procesar dicha instrucción— es siempre predecible (*i. e.,* determinista). En general, el lenguaje natural no es tan restrictivo en su gramática[6], lo que produce el fenómeno de ambigüedad. Así,

[6]Escribir una gramática completa y correcta para el lenguaje natural (desde una perspectiva computacional) no es una tarea viable. Generalmente se trabaja con gramáticas que reconocen porciones significativas del lenguaje excluyendo estructuras complejas que son menos comunes en determinados contextos.

a menudo es posible encontrar múltiples *parses* para una misma oración en lenguaje natural, lo cual se conoce como ambigüedad sintáctica [25].

El fenómeno de ambigüedad sintáctica afecta a menudo la claridad, y por ende la calidad, de los requerimientos de software. En el sentido estricto, una oración con dos o más *parses* (*i. e.,* con dos o más árboles de derivación sintáctica) presenta necesariamente uno o más casos de ambigüedad sintáctica. Sin embargo, no se puede suponer que todos los casos representan problemas a nivel de requerimientos de software, por cuanto el significado de dicho requerimiento en el plano de lo técnico podría no verse afectado por sus variaciones sintácticas. Por ejemplo, un requerimiento que tenga dos o más condiciones todas unidas por un operador lógico del mismo tipo (AND u OR) presentará múltiples *parses* necesariamente. Sin embargo, dado que el operador lógico es el mismo para todas las condiciones, esta situación no presenta ningún error en el plano técnico. Profundizar en este supuesto es parte de las tareas realizadas en esta investigación.

Para los seres humanos identificar y resolver problemas de ambigüedad sintáctica a menudo es una tarea trivial; sin embargo, para identificar y resolver los mismos problemas a partir de un programa de computadora, se requiere aplicar diversas técnicas tanto heurísticas y estadísticas como formales.

Un repaso completo sobre las técnicas de *parsing* estadístico que brindan buenos resultados se ofrece en el libro de Manning y Schütze [20]. Por otro lado, el trabajo de Nugues [25] también describe una variedad de técnicas de *parsing* que son apropiadas en un determinado contexto. Finalmente, el trabajo de Carroll *et al.* [4] resume algunos de los reportes de investigación más notables en el tema de tecnologías de *parsing* reunidos durante las conferencias *International Workshop on Parsing Technology* IWPT2000[7] y IWPT2001[8].

En la siguiente sección se describe otra técnica de procesamiento natural conocida como *tagging*.

[7] `http://tcc.itc.it/events/iwpt2000.html`
[8] `http://hmi.ewi.utwente.nl/sigparse/iwpt2001/MIRROR/`

4.4.2 Tagging

El concepto de *tagging* se puede entender como un complemento del concepto de *parsing*. Esta técnica permite identificar la función que cumple cada palabra en una oración. Su importancia para esta investigación se debe a que ayuda a identificar problemas relacionados con los requerimientos de software en relación con los atributos descritos en el objetivo específico 1.

Se conoce como *part-of-speech tagging* (POS-tagging) o simplemente *tagging* al proceso de etiquetar cada palabra en una oración con un elemento o categoría sintáctica que le corresponda según su lugar en la oración. Por ejemplo, considere la oración en la figura 4.4 que utiliza las etiquetas (*tags*) establecidas en Brown/Penn[9], las cuales se muestran en el apéndice A.

En la oración de la figura 4.4, el término *records* es etiquetado como **NNS**, o sustantivo plural; mientras que, en la figura 4.5 ese mismo término funciona como un verbo en tiempo presente para la tercera persona singular **VBZ**. Este tipo de información sintáctica es la que se obtiene mediante técnicas de tagging.

The-AT *records*-NNS *were*-VBD *lost*-VBN .

Figura 4.4: Ejemplo de POS-tagging aplicado a una oración.

The-AT *system*-NN *records*-VBZ *all*-AT *events*-NNS.

Figura 4.5: Ejemplo alternativo de POS-tagging aplicado a la misma oración.

Dependiendo del contexto, una misma estructura puede tener dos o más formas de etiquetarla; en consecuencia, se requiere de técnicas especiales para lograr determinar la etiqueta más apropiada para una palabra en un contexto determinado y, por ello, el problema de *tagging* tampoco se resuelve de manera determinista.

Existen enfoques simbólicos y probabilísticos para resolver el problema de *tagging* [20, 25]. Ambos enfoques tienen tanto virtudes como debilidades dependiendo de las variaciones grama-

[9]Las etiquetas o *tags* de más influencia en inglés son los utilizados para etiquetar el American Brown corpus (*Brown tag set*) o su forma simplificada llamada *Penn Treebank set* que se utilizó en esta investigación.

ticales del texto que se intente etiquetar. En esta investigación se dará prioridad a los enfoques probabilísticos sobre los simbólicos, ya que hay más recursos disponibles que siguen ese enfoque y los resultados son positivos [20].

La técnica de POS-tagging será utilizada en esta investigación de manera amplia para detectar problemas tanto de ambigüedad e imprecisión como de verificabilidad en los requerimientos de software a examinar. En el caso de ambigüedad se utiliza como un complemento de la técnica de *parsing* ya que, mientras la técnica de *parsing* muestra deficiencias a nivel de la estructura de un requerimiento, la técnica de *tagging* permite ubicar elementos específicos tales como verbos y adverbios para ser examinados en más detalle. En el caso de precisión y verificabilidad se utiliza para ubicar los elementos que adquieren una categoría sintáctica específica y así poder evaluar dichos elementos. Por ejemplo, en los verbos se tratará de detectar problemas de verificabilidad y en los adverbios problemas de precisión.

En la siguiente sección se describen algunos de los recursos lingüísticos que se han utilizado en la investigación.

4.5 Recursos Lingüísticos

En general, existen diversos recursos lingüísticos disponibles en Internet que facilitan tareas como análisis léxico o semántico. La importancia de estos recursos es que proveen una base sólida sobre la cual es posible realizar investigación, en particular estos recursos han sido utilizados para cumplir los objetivos planteados en este trabajo.

Entre los recursos más relevantes para este trabajo están los repositorios de datos semejantes a diccionarios. Estos repositorios tienen la particularidad de que organizan los datos de manera accesible mediante un lenguaje de programación. A continuación se describen dos de los principales recursos utilizados en la investigación: WordNet [24] y VerbNet [13]. Cada uno de estos recursos proporciona información distinta pero complementaria sobre palabras en el idioma inglés. Esta información puede ser consultada de diversas maneras y en esta investigación en particular se hizo mediante una interfaz (o módulo) escrita en Perl.

Como se verá a continuación, WordNet es un recurso que ayuda a detectar del problema de ambigüedad, más precisamente la ambigüedad semántica. Por otro lado, VerbNet es utilizado para detectar problemas de verificabilidad en los requerimientos de software.

4.5.1 WordNet

WordNet[10] es un sistema de referencia léxica cuyo diseño se inspira en teorías psicolingüísticas modernas sobre la memoria léxica del ser humano [24]. A diferencia de los diccionarios convencionales que organizan los vocablos (información léxica) en orden alfabético, en WordNet los vocablos se organizan tomando en cuenta su significado (información semántica) en grupos denominados *synsets*[11]. Así, palabras con significado similar o relacionado estarán agrupadas entre sí independientemente de su forma [24].

Para entender cómo funciona WordNet, es bueno conocer primero la manera en que se organizan los datos en su repositorio. Una de las premisas es que generalmente se utiliza un mismo término *"palabra"* para referirse tanto a la expresión misma como al concepto que esta encierra. En lexicografía es conveniente distinguir entre *forma* para referirse a la emisión en sí de una palabra, y *significado* para referirse al concepto concreto o abstracto que dicha *forma* representa [24]. Visto así, el punto de partida en léxico-semántica consiste en crear asociaciones entre *formas* y *significados*.

En WordNet se utiliza el concepto de matriz léxica para representar las asociaciones entre *formas* y *significados*, tal y como se explica en la Tabla 4.4. En esta tabla, los encabezados de las columnas representan *formas* mientras que los encabezados de las filas representan *significados*.

Cada entrada $E_{i,j}$ en la tabla indica que la forma F_j se puede utilizar —en un determinado contexto— para expresar el significado S_i. Dos o más entradas en una misma columna j indican que la forma F_j es polisémica[12], *p. ej.,* $E_{1,2}$ y $E_{2,2}$. De manera similar, dos o más entradas en una misma fila i indican que las formas correspondientes a esas entradas son sinónimos entre sí en un determinado contexto, *p. ej.,* $E_{1,1}$ y $E_{1,2}$.

[10] WordNet es una marca registrada de *Princeton University*.
[11] *synset* se deriva del inglés *synonym set*.
[12] Dicho de una palabra o cualquier otro signo lingüístico que tiene varios significados.

Tabla 4.4: Concepto de matriz léxica utilizado en WordNet para la construcción de *synsets*. La 1^{ra} columna muestra una serie de significados en WordNet. En la derecha, bajo Formas hay una lista de formas o palabras. Las asociaciones entre formas y significados son las entradas E de la matriz. Nótese como esta forma de organizar la información en WordNet es distinta de la forma tradicional (orden alfabético) utilizada en otros diccionarios.

Significados	Formas						
	F_1	F_2	F_3	.	.	.	F_n
S_1	$E_{1,1}$	$E_{1,2}$					
S_2		$E_{2,2}$					
S_3			$E_{3,3}$				
.				.			
.					.		
.						.	
S_m							$E_{m,n}$

Un *synset* es la mínima unidad utilizada en WordNet para expresar un significado. Se identifica un *synset* o significado S_k listando las distintas formas que se pueden utilizar para expresarlo $\{F_1, F_2, ...\}$. Además, se utiliza una frase o explicación breve (denominada *gloss*) que contiene una definición del concepto y oraciones de ejemplo que ilustran su uso. Dado que WordNet organiza los conceptos según sus características semánticas y no léxicas, se dice que su repositorio consiste en una colección de *synsets* y no de palabras.

La red de relaciones semánticas en WordNet se forma a partir de asociaciones o punteros entre *synsets*. Los *synsets* contienen conceptos de cuatro categorías gramaticales: sustantivos, verbos, adjetivos y adverbios. Las unidades gramaticales funcionales[13] se excluyen bajo el supuesto de que forman parte de la sintaxis del lenguaje. Sin embargo, esto no resta valor a WordNet como recurso en la investigación ya que no se requiere hacer consultas sobre este tipo de palabras. WordNet posee muchas relaciones semánticas; sin embargo, algunas de las más relevantes para este trabajo se describen a continuación.

Synonymy Esta es una relación de similitud semántica entre palabras. Dos palabras P_1 y P_2 son sinónimos en un determinado contexto lingüístico C si la substitución de una de ellas

[13]En inglés se conocen como *function words* y corresponden a categorías como pronombres, preposiciones, conjunciones, verbos auxiliares entre otras. Estas unidades se utilizan para crear enlaces o relaciones entre unidades gramaticales de contenido, tales como sustantivos, verbos, adjetivos y adverbios.

por la otra, en C, no altera el valor verdadero del texto en ese contexto. Esta relación se da entre vocablos de la misma categoría gramatical (sustantivos, verbos, adjetivos y adverbios).

Antonymy Antonimia es una relación léxica (entre formas de palabras) y no semántica (entre significados de palabras). El antónimo de una palabra x usualmente es *no-*x pero no siempre es así. Por ejemplo, los pares [*rise* / *fall*] y [*ascend* / *descend*] son antónimos pero no se reconocen como tal sus combinaciones [*rise* / *descend*] o [*fall* / *ascend*].

Hyponymy, Hypernymy Hiponimia es una relación semántica en la cual el significado de una palabra P_1 está incluido en el de otra P_2 (su hiperónimo). Por ejemplo, *cedro* es un hipónimo de *árbol* que a su vez es un hipónimo de *planta*. Se conoce también como una relación ISA (*a* ***is a kind of*** *A*) donde $a = P_1$ *is a (kind of)* $A = P_2$.

Meronymy , Holonymy Es una relación semántica y se conoce como relación parte-todo (o HASA, *B* ***has a*** b). En WordNet, un concepto representado por el *synset* $\{x, x', \dots\}$ es un *meronym* del concepto representado por el *synset* $\{y, y', \dots\}$ si los hablantes nativos del idioma inglés aceptan oraciones de la forma: *A* y *has an* x *(as a part)* o, *An* x *is part of* y.

Estas relaciones se utilizarán en esta investigación principalmente para detectar problemas de ambigüedad semántica en los requerimientos de software.

4.5.2 VerbNet

VerbNet es la base de datos sobre verbos más grande disponible en línea para el idioma inglés y fue creada en el Departamento de lingüística de la Universidad de Colorado en Boulder [14, 13]. VerbNet posee una estructura jerárquica, independiente del dominio y compuesta por varias clases verbales que se describen utilizando información tanto sintáctica como semántica basándose en la clasificación propuesta por Levin [19].

El principal supuesto para dicha estructura es que los posibles esquemas o patrones sintácticos en los cuales puede participar un verbo, son indicadores directos de la semántica subyacente a

dicho verbo. VerbNet asocia entonces la semántica de un verbo con sus posibles esquemas sintácticos en combinación con información léxico-semántica tradicional como roles temáticos y predicados semánticos. En este sentido, VerbNet implementa la hipótesis de Levin sobre la relación cercana entre sintaxis y semántica.

Los verbos que pertenecen a una misma clase verbal en VerbNet comparten los mismos esquemas sintácticos y, por tanto, se cree que tienen el mismo comportamiento sintáctico [13]. Cada clase verbal tiene una serie de esquemas sintácticos que describen las formas posibles que pueden tomar los argumentos de los verbos en dicha clase; *p. ej.,* los complementos directos y circunstanciales en los verbos transitivos. Incluye restricciones semánticas (tales como ser vivo, humano, organización) que se aplican a los roles temáticos permitidos para los argumentos.

En esta investigación no se utiliza toda la información que VerbNet puede proveer. Sin embargo, se realizan consultas para determinar si los requerimientos utilizan verbos apropiados, es decir, verbos que representan acciones que puede realizar un computador. También se realizan consultas para determinar si un verbo posee varios significados y con ello considerar problemas de ambigüedad. La tabla 4.5 muestra cuatro grupos verbales en VerbNet junto con algunos de los verbos en cada clase.

Tabla 4.5: Ejemplos de clases verbales en VerbNet. La 1^{ra} columna muestra clases en VerbNet y la 2^{da} muestra ejemplos de verbos que existen en cada clase.

Clase	Verbos
RISK	*bet, gamble, venture*
CONFESS	*confess, admit, reveal*
GROW	*grow, develop, evolve*
REMOVE	*remove, abolish, delete*

La base de verbos en VerbNet fue actualizada recientemente por Korhonen [16] y Kipper *et al.* [15] y ofrece mapeos directos con otros recursos como WordNet y FrameNet[1].

La base de datos de VerbNet es importante pues se utiliza para identificar vicios de verificabilidad en los requerimientos de software.

Para continuar, el siguiente capítulo presenta con detalle los alcances y limitaciones del trabajo. En ese apartado se describe el producto que se espera conseguir con la investigación.

5

Alcances y Limitaciones

La investigación que aquí se plantea pertenece tanto al área de procesamiento de lenguaje natural (PLN) como al área de ingeniería de software. En el primer caso, la aplicación de PLN obedece específicamente al análisis automático de requerimientos de software. En el segundo caso, la investigación se relaciona específicamente con la rama de ingeniería de requerimientos. Se hace hincapié en que el problema a tratar corresponde únicamente a la detección parcial y automática de ciertos tipos de *imprecisión*, *ambigüedad* y *no-verificabilidad* en requerimientos de software para sistemas empotrados en el área de aviónica.

La documentación disponible para pruebas durante la investigación corresponde a tres sistemas cuyas características se describen en la tabla 5.1. Estos sistemas se componen de varios subsistemas que aportan aproximadamente 2.100 requerimientos de software tanto de alto como de bajo nivel.

Tabla 5.1: Lista de sistemas utilizados para realizar la investigación. La 1^{ra} columna indica el nombre del sistema; la 2^{da} columna indica el aporte en cuanto a número de requerimientos y la 3^{ra} columna es una descripción breve del sistema.

Sistema	**Aporte**	**Descripción**
Multi functional display	65 %	Aplicación de software que controla el comportamiento de un dispositivo de entrada de datos y salida de video conocido como MFD.
On-board health monitoring unit	20 %	Módulo de monitoreo general que constantemente revisa la integridad del sistema durante un vuelo.
On-board maintenance unit	15 %	Módulo de monitorero de errores que se utiliza para documentar ciertas fallas ocurridas durante un vuelo.

Generalmente, los requerimientos de software se agrupan en dos categorías según el nivel de detalle con que estos sean escritos. Un requerimiento de software de alto nivel describe o establece una característica funcional que debe tener el sistema a construir. Este tipo de requerimientos suele ser muy general, por lo que a menudo no es posible implementar dicha funcionalidad

en el código fuente utilizando el requerimiento como único insumo. Por el contrario, un requerimiento de software de bajo nivel también describe una característica o propiedad que debe tener el sistema; sin embargo, en este caso el requerimiento sí contiene detalles suficientes que permiten escribir el código fuente que implementa dicha funcionalidad.

Todos los requerimientos de software con que se realiza la investigación están escritos en idioma inglés, por lo cual las técnicas utilizadas corresponden a aspectos específicos de dicho idioma.

A continuación se detallan algunas de las características sobre el producto esperado y posteriormente se explican con mayor detalle algunos de los principales aspectos que no serán cubiertos en la investigación.

5.1 Alcances

Tal y como está descrito en la sección 3.1, el producto que plantea en esta investigación es un prototipo de software que tome un grupo de requerimientos de software como entrada, y produzca como salida un reporte que indique en qué grado dichos requerimientos de software se encuentran o no en conformidad con las propiedades de *precisión*, *no ambigüedad* y *verificabilidad*, tal y como se definen en la investigación (ver sección 7.2.).

La información de salida final del prototipo ayuda a determinar cuantitativamente, cuál de las propiedades está o no está presente, y en qué grado, dentro del conjunto de requerimientos de software analizado tal y como se explica a continuación.

Para comenzar, la salida muestra el número total de requerimientos analizados en una sesión. Además, se presenta un resumen que indica el número y porcentaje de requerimientos que posee algún grado de ambigüedad, imprecisión o no verificabilidad, tal y como lo muestra la tabla 5.2.

Adicionalmente, el prototipo es capaz de desplegar una lista con el identificador para los requerimientos en cada uno de los tres grupos. Finalmente, el prototipo también muestra en detalle la razón por la cual cada requerimiento fue seleccionado dentro de dicho grupo.

Tabla 5.2: Resumen de información de salida del prototipo. En este ejemplo las columnas indican la información de salida que debe producir el sistema luego de analizar un grupo de requerimientos. Por ejemplo, la 1^{ra} columna mostraría cada una de las tres propiedades que trata la investigación. La 2^{da} columna mostraría el número de requerimientos que no satisface cada propiedad, la 3^{ra} columna mostraría el porcentaje de los requerimientos que no satisface cada propiedad (respecto del total de requerimientos analizado).

Indicador	**Candidad**	**Porcentaje**
Requerimientos ambiguos	ρ_1	$\pi_1\,\%$
Requerimientos imprecisos	ρ_2	$\pi_2\,\%$
Requerimientos no verificables	ρ_3	$\pi_3\,\%$

Para facilitar el proceso de lectura e interpretación de los resultados, el sistema asigna un valor o una calificación (en escala discreta) a cada uno de los requerimientos afectados. Este valor pretende funcionar como una estimación cualitativa sobre el grado de madurez de ese requerimiento. Dado que esta calificación solamente se ofrece para los requerimientos afectados, lo que se mide es el grado de incumplimiento de un requerimiento de software con los lineamientos de DO-178B estudiados en esta investigación. Los niveles utilizados para medir el grado de incumplimiento son: *bajo*, *medio* y *alto*. Una calificación de *bajo* indica que, pese a que el requerimiento no cumple con alguno de los atributos deseados, la magnitud de la falta es leve y potencialmente fácil de corregir. La calificación de *alto* indica, por el contrario, que el daño es grave y potencialmente difícil de corregir.

Para lograr el objetivo planteado, el software utiliza técnicas de procesamiento de lenguaje natural, específicamente análisis léxico, análisis morfológico y análisis sintáctico, así como también diccionarios disponibles en Internet para realizar búsquedas especializadas sobre el significado de diversos términos.

Tabla 5.3: Objetivos en DO-178B considerados en el estudio. La 1^{ra} columna muestra un número de identificación solo para referencia. La 2^{da} columna indica la tabla del estándar DO-178B donde se encuentra este objetivo. La 3^{ra} columna indica el número del objetivo y la 4^{ta} columna tiene la descripción del objetivo.

Id	**Tabla**	**Objetivo**	**Descripción**
O_1	A-3	2	*High-level requirements are accurate and consistent.*
O_2	A-3	4	*High-level requirements are verifiable.*
O_3	A-4	2	*Low-level requirements are accurate and consistent.*
O_4	A-4	4	*Low-level requirements are verifiable.*

Los objetivos del estándar DO-178B que se incluyen[1] en esta investigación como aspectos a automatizar para la fase de *Software Requirements Process* se resumen en la tabla 5.3. Adicionalmente, para cumplir con los objetivos para la fase de *Software Requirements Process*, el estándar DO-178B propone una serie de lineamientos que sirven como guía para satisfacer los objetivos mencionados anteriormente. La tabla 5.4 resume aquellos lineamientos que fueron considerados en esta investigación[2]. Se escogieron únicamente cuatro lineamientos ya que estos hacen referencia a las propiedades que abarca la investigación: *no-ambigüedad*, *precisión* y *verificabilidad*.

Tabla 5.4: Lineamientos en DO-178B considerados en el estudio. La 1^{ra} columna muestra un número de identificación solo para referencia. La 2^{da} columna indica la sección del estándar DO-178B donde se encuentra este lineamiento. La 3^{ra} columna indica el número de lineamiento o actividad y la 4^{ta} columna es la descripción del lineamiento.

Id	Sección	Actividad	Descripción
L_1	5.1.2	a	*The system functional and interface requirements that are allocated to software should be analyzed for ambiguities, inconsistencies and undefined conditions.*
L_2	5.1.2	e	*The high-level requirements should conform to the Software Requirements Standards, and be verifiable and consistent.*
L_3	5.1.2	f	*The high-level requirements should be stated in quantitative terms with tolerances where applicable.*
L_4	5.2.2	a	*Low-level requirements and software architecture developed during the software design process should conform to the Software Design Standards and be traceable, verifiable and consistent.*

La siguiente sección demarca de forma explícita algunas de las principales limitaciones de la investigación.

5.2 Delimitación

En los capítulos anteriores se ha venido perfilando, de manera no explícita, la relación entre los conceptos de *precisión*, *no ambigüedad* y *verificabilidad* y los objetivos del estándar DO-178B para la fase de *Software Requirements Process* mostrados en la tabla 5.3, junto con

[1] Para conocer los demás objetivos refiérase al Anexo A, tablas A-3 y A-4 en [30].

[2] Para conocer los demás lineamientos refiérase a la sección *5.1.2 Software Requirements Process Activities* o la sección *5.2.2 Software Design Process Activities* en [30].

los lineamientos mostrados en la tabla 5.4. A continuación se describe con mayor claridad dicha relación respondiendo a estas preguntas:

a. ¿Cúal es el vínculo entre los conceptos de *precisión*, *no-ambigüedad* y *verificabilidad* y los objetivos y lineamientos del DO-178B en las tablas 5.3 y 5.4?

b. ¿Cuáles aspectos son tratados y cuáles no para cada uno de dichos objetivos y lineamientos[3] en esta investigación?

Considere la tabla 5.5. Esta resume las asociaciones entre los conceptos tratados en esta investigación y los objetivos y lineamientos del DO-178B seleccionados anteriormente.

Tabla 5.5: Vínculo entre conceptos estudiados y DO-178B. La 1^{ra} columna muestra el concepto del que se habla. La 2^{da} columna indica con una equis (x) los objetivos del DO-178B asociados con el concepto. La 3^{ra} columna indica también con una equis (x) los lineamientos del DO-178B asociados con el concepto.

Concepto	**Objetivos**				**Lineamientos**			
	O_1	O_2	O_3	O_4	L_1	L_2	L_3	L_4
precisión	x	-	x	-	x	-	x	-
no-ambigüedad	x	x	x	x	x	x	-	x
verificabilidad	-	x	-	x	-	x	-	x

Lo que se desea indicar con la tabla 5.5 es, por ejemplo, que el concepto *precisión* está asociado con cada elemento p del conjunto $P = \{O_1, O_3\} \bigcup \{L_1, L_3\}$ ya que:

a. El concepto está relacionado con los objetivos $\{O_1, O_3\}$ pues estos establecen (explícitamente) que los requerimientos de software deben ser *precisos* (o *accurate* en inglés).

b. Está relacionado con el lineamiento L_1 ya que este establece que se debe verificar que un requerimiento de software esté libre de condiciones no definidas (pues de otro modo dicho requerimiento no es preciso).

[3]La razón principal para no tratar completamente cada uno de los objetivos y lineamientos es que algunos de ellos imponen restricciones sobre los requerimientos (*p. ej.,* consistencia) cuya verificación sobrepasa los objetivos de esta investigación.

c. Está relacionado con el lineamiento L_3 ya que este establece que los requerimientos deben escribirse en términos cuantificables, indicando valores de tolerancia siempre que sea pertinente.

De manera similar se define la relación entre el concepto *verificabilidad* y los elementos v del conjunto $V = \{O_2, O_4\} \bigcup \{L_2, L_4\}$ ya que cada uno de estos elementos establece explícitamente que los requerimentos deben ser verificables.

En el caso del concepto *no-ambigüedad*, se define su relación con cada elemento a del conjunto $A = \{O_1, O_2, O_3, O_4\} \bigcup \{L_1, L_2, L_4\}$ ya que:

a. L_1 indica de manera explícita que se debe descartar la presencia de ambigüedades en los requerimientos.

b. La presencia de cierto tipo de ambigüedad, *p. ej.,* léxica, provoca que el requerimiento no sea preciso.

c. La presencia de cierto tipo de ambigüedad, *p. ej.,* sintáctica, provoca que el requerimiento se preste para múltiples interpretaciones. Esto lo convierte en un requerimiento no verificable pues no es válido suponer que los distintos actores del ciclo de desarrollo (*p. ej.,* autor del requerimiento, programador o verificador) tuvieran todos la misma posición respecto de dicho requerimiento.

Además, nótese que los siguientes elementos $e \in E = \{O_1, O_3\} \bigcup \{L_1\}$ sugieren que los requerimientos de software deben ser consistentes entre sí, que deben ser rastreables (del inglés *traceable*) y que deben estar en conformidad con los respectivos estándares *Software Requirements Standard* y *Software Design Standard*. Sin embargo, verificar el cumplimiento de estas últimas propiedades está fuera del alcance de esta investigación.

Finalmente, se tomará como supuesto que la calidad textual[4] de los requerimientos de software es suficientemente buena como para no inducir incoherencias en los resultados del prototipo. En esta investigación se pretende incluir mecanismos de tolerancia a errores gramaticales u

[4]Calidad textual se refiere a aspectos como gramática, ortografía, orden en el formato del texto y otros aspectos que empobrezcan la legibilidad del texto de los requerimientos.

ortográficos cuando sea posible. Sin embargo, este esfuerzo no será preponderante respecto de los objetivos de la investigación, de manera tal que conjuntos de requerimientos de software de mala calidad textual podrán ser rechazados completamente.

6

Metodología General

En este capítulo se describe la organización general de la metodología usada durante la investigación. Luego, en los capítulos 7, 8 y 9 sus componentes principales se abordan con más detalle. En la primera etapa del desarrollo de esta investigación, fue necesario establecer la nomenclatura de trabajo que permitió expresar de forma simbólica y resumida cuando un requerimiento cumple o no con las propiedades deseables. La segunda etapa consistió en acotar los conceptos de *ambigüedad*, *precisión* y *verificabilidad* para ajustarlos al contexto de trabajo en análisis de requerimientos de software. Una vez acotados los conceptos se procede con la tercera etapa: diseño y construcción del prototipo. En la última etapa se realizaron experimentos para validar la funcionalidad de la herramienta, y para esto se contó con la participación profesionales en el análisis de requerimientos de software.

Debido a que la investigación se ha separado en varias etapas, cada uno de los capítulos 7, 8 y 9 presenta una descripción de la metodología utilizada en dicha etapa, y luego presenta un resumen sobre los resultados de la investigación hasta ese punto. Esta organización "*metodología/resultados*" también se ha aplicado a nivel de secciones en los siguientes capítulos para facilitar la comprensión del texto.

En la primera tarea de acotación se preparó una lista inicial con algunos de los elementos lingüísticos que, a menudo, causan que los requerimientos no satisfagan los conceptos generales de *no-ambigüedad*, *precisión* o *verificabilidad*. Esta lista surge como resultado de una revisión de la literatura disponible y la utilización del criterio del autor con su experiencia en este campo. Dado que una lista como esta puede ser extensa, la siguiente tarea fue seleccionar un subconjunto representativo con aquellos elementos que serían evaluados por el prototipo a crear. Seguidamente, para asegurarse de que la selección del paso anterior es válida y representativa, se hizo una evaluación independiente en la cual colaboraron profesionales en el tema de análi-

sis de requerimientos de software para sistemas empotrados en aviónica. Una vez que se tenía la lista de elementos a utilizar, fue necesario asignar un puntaje a cada uno de los elementos. Para este efecto, el autor propuso un puntaje inicial para los elementos de acuerdo con su criterio profesional; luego, mediante otro instrumento, los profesionales colaboraron para validar los puntajes, ofreciendo valores alternativos cuando lo consideraron oportuno. El capítulo 7 explica ampliamente los detalles de este primer proceso.

Una vez que se tenía la lista de elementos seleccionada y con sus respectivos puntajes, se procedió a trabajar en el diseño conceptual del prototipo. El diseño incluye un esquema cuantitativo que utiliza los resultados de la selección de elementos, la nomenclatura propuesta y otras consideraciones técnicas para la creación del prototipo de software. Con el diseño listo, se continuó con la implementación del prototipo y posteriormente se aplicaron, pruebas de funcionalidad básica sobre el prototipo. Los detalles de la etapa de diseño e implementación se describen en el capítulo 8.

Para la etapa de evaluación del prototipo se diseñaron y aplicaron algunos experimentos comparativos con la ayuda de otro grupo de profesionales (distintos de los que colaboron en la fase de evaluación de elementos). Durante los experimentos, tanto la herramienta como los profesionales realizaron una evaluación de varios grupos de requerimientos escogidos de manera estratégica para medir ciertos aspectos de interés en la investigación. Por ejemplo, entre los requerimientos escogidos para evaluación se incluyen algunos que no presentan ningún error, otros que tienen un error específico (*p. ej., ambiguo*, *impreciso* o *no verificable*) y otros que tienen múltiples errores. El capítulo 9 incluye una descripción completa sobre el propósito de cada uno de los experimentos realizados en esta fase. Además, en este último capítulo se presenta un análisis de resultados basado en los datos obtenidos luego de realizar los experimentos antes mencionados. En esta etapa se discuten varios resultados producto de las comparaciones hechas durante los experimentos.

En el siguiente capítulo se detalla la primera parte del proceso, que comienza con el planteamiento de la nomenclatura y culmina con la selección de elementos a utilizar en el prototipo.

7

Selección de Elementos

En este capítulo se describe la metodología utilizada para seleccionar los elementos que determinarán cuándo un requerimiento cumple o no con los atributos de *precisión*, *no-ambigüedad* y *verificabilidad*. Cada una de las secciones en este capítulo presenta primero una descripción de la metodología utilizada en cada fase y posteriormente presenta los resultados obtenidos en dicha fase. La primera sección describe la nomenclatura que se utilizó en la investigación y las siguientes explican el proceso de acotación de conceptos. Como se verá más adelante, la fase de acotación de conceptos culmina con una lista específica de elementos que fueron luego utilizados para implementar el prototipo de software.

7.1 Nomenclatura del Trabajo

La etapa de creación de la nomenclatura se presenta en dos partes. La sección 7.1.1 describe la metodología a seguir en esta fase y la sección 7.1.2 describe con detalle los resultados obtenidos.

7.1.1 Metodología de Creación de Nomenclatura

El objetivo de establecer una nomenclatura particular es poder describir, de manera simbólica y consistente, los casos en que un requerimiento cumple o no con los tres atributos mencionados en el objetivo específico 1: *precisión*, *no-ambigüedad* y *verificabilidad*. Esta nomenclatura debe ser sencilla de entender y aplicar, de modo que no agregue complejidad innecesaria al tema de fondo, que es análisis de requerimientos de software. Para garantizar este objetivo, se pensó utilizar como base una notación algebraica conocida como teoría de conjuntos y teoría de funciones. La siguiente sección presenta los resultados de esta fase.

7.1.2 Nomenclatura Propuesta

Para establecer la nomenclatura se inicia por la representación de *elementos* para cada atributo. Para representar el atributo de *no-ambigüedad*, se define el conjunto $\Lambda = \{\lambda_1, \lambda_2, ..., \lambda_k\}$ con $k \in \mathbb{N}_{\geq 1}$ donde cada λ_i es un elemento que revela el incumplimiento del atributo de *no-ambigüedad* en un requerimiento de software dado. Por ejemplo, si λ_1 = "*Un requerimiento no debe utilizar adverbios vagos o adverbios generales para describir la acción principal*", entonces se puede decir que al aplicar λ_1 a un requerimiento R_1 = "*The system shall allow the operator to adjust volume to an acceptable level*", se revelaría que dicho requerimiento es *ambiguo*, ya que el adverbio "*acceptable*" es vago; y se diría que R_1 no satisface λ_1. Análogamente, para representar el atributo de *precisión* se define $\Gamma = \{\gamma_1, \gamma_2, ..., \gamma_r\}$ con $r \in \mathbb{N}_{\geq 1}$, donde cada γ_i es un elemento que revela el incumplimiento del atributo de *precisión* en un requerimiento de software dado. Finalmente, para el atributo de *verificabilidad* se define el conjunto $\Upsilon = \{\upsilon_1, \upsilon_2, ..., \upsilon_s\}$ con $s \in \mathbb{N}_{\geq 1}$.

Resumiendo, para cada uno de los atributos (*no-ambigüedad*, *precisión* y *verificabilidad*), se definen los conjuntos de elementos respectivos:

$$X_1 = \Lambda \quad , \quad X_2 = \Gamma \quad , \quad X_3 = \Upsilon$$

donde $X_i = \{\epsilon_1, \epsilon_2, ..., \epsilon_n\}$ y cada ϵ_i es un *elemento* o marcador que indica cuando un requerimiento de software incumple (o no satisface completamente) el atributo al que pertenece.

Para denotar cuando los elementos revelan el incumplimiento de atributos y para facilitar la lectura de los análisis, se propone utilizar la siguiente notación:

- Cuando un requerimiento de software R_e cumple con la restricción impuesta por un elemento ϵ_k, se dice que R_e *satisface* ϵ_k, y se denota $R_e \odot \epsilon_k$.
- Cuando un requerimiento de software R_e no cumple con la restricción impuesta por un elemento ϵ_k, se dice que R_e *no satisface* ϵ_k, y se denota $R_e \oslash \epsilon_k$.
- Cuando la restricción impuesta por un elemento ϵ_k no es aplicable para un requerimiento de software R_e, se dice que ϵ_k *no es aplicable* a R_e y se denota $R_e \oplus \epsilon_k$

Nótese que las expresiones $R_e \odot \epsilon_k$, $R_e \oslash \epsilon_k$ y $R_e \oplus \epsilon_k$ son en el fondo predicados lógicos. Por ejemplo, de manera alternativa se puede leer la primera expresión como $\odot(R_e, \epsilon_k)$ o bien SATISFACE(R_e, ϵ_k) .

La nomenclatura descrita puede representar el hecho de que R_e *satisfaga* o *no satisfaga* uno o más elementos para los atributos de *no-ambigüedad*, *precisión* y *verificabilidad* y por ende a nivel de elementos esta notación binaria es apropiada y se logra tener una descripción cualitativa. Sin embargo, para determinar en qué medida un requerimiento satisface un atributo, la situación es más compleja ya que el requerimiento puede *satisfacer* unos elementos y no otros. Además, algunos elementos son más críticos que otros y por ende "causan más daño" al requerimiento. Como resultado, la *satisfacción* o *no-satisfacción* a nivel de atributos se debe medir de forma numérica (y no lógica) como se describe a continuación.

Para medir cuantitativamente el cumplimiento de un requerimiento con un atributo se usa una escala numérica arbitraria entre 0 y 10, es decir, se asigna una ***calificación*** al requerimiento. Para determinar dicha *calificación*, se comienza por asignar un puntaje a cada uno de los elementos $\epsilon_1, \epsilon_2, ..., \epsilon_n$ en X_i de tal forma que los puntajes de los elementos en cada atributo sumen 10 y $puntaje(\epsilon_i) = p$, $p \in \mathbb{R}^+_{10}$ (donde $\mathbb{R}^+_{10} = [0, 10]$). Es decir:

$$\sum_{j=1}^{|X_i|} puntaje(\epsilon_j) = 10 \quad \text{con } \epsilon_j \in X_i \quad \text{y} \quad X_i \in \{\Lambda, \Gamma, \Upsilon\} \tag{7.1}$$

La distribución de puntajes para los atributos se establece tomando en cuenta la gravedad del error que cada elemento ϵ_j representa en un requerimiento. A los elementos que representan errores más críticos les corresponden valores más altos[1]. Una de las debilidades de este esquema es que hace difícil la tarea de agregar, modificar o eliminar elementos para cada atributo pues sería necesario reorganizar los puntajes asignados a cada elemento.

Por ejemplo, suponga que se define el atributo de *Verificabilidad* como $\Upsilon = \{\upsilon_1, \upsilon_2, \upsilon_3\}$ donde cada υ_i representa las condiciones descritas en la tabla 7.1

[1]En la sección 7.4 se describe con detalle el resultado de la asignación de puntajes entre los elementos de cada atributo.

Tabla 7.1: Elementos de ejemplo para acotar el concepto de *verificabilidad*. La 1^{ra} columna muestra el identificador de elemento y la 2^{da} la descripción del elemento para un escenario hipotético.

Elemento	Descripción
υ_1	*Un requerimiento no debe utilizar la estructura "shall not"*
υ_2	*Un requerimiento no debe utilizar la estructura "shall only"*
υ_3	*Un requerimiento no debe utilizar los verbos "consider" o "analyze"*

Así descritos, cada uno de esos elementos parece representar un error de similar gravedad en un requerimiento. Sin embargo, supongamos por ahora que, en la práctica, corregir un requerimiento que no satisface υ_3 es el doble de costoso que corregir un requerimiento que no satisface υ_1 o υ_2. Entonces, para asignar los puntajes en Υ podemos utilizar los siguientes valores (2,5 , 2,5 , 5) de modo que: $puntaje(\upsilon_1) = 2{,}5$, $puntaje(\upsilon_2) = 2{,}5$ y $puntaje(\upsilon_3) = 5$ y se cumple que $\sum_{i=1}^{3} puntaje(\upsilon_i) = 10$

Ahora, para determinar la *calificación* de un requerimiento R_e con respecto a un atributo representado por su conjunto $X_i \in \{\Lambda, \Gamma, \Upsilon\}$, se define una función de calificación por atributo, θ de la siguiente manera:

$$\theta(R_e, X_i) = [\ 10 - \sum_{j, R_e \oslash \epsilon_j} puntaje(\epsilon_j)\ ] = [\ \sum_{j, R_e \odot \epsilon_j} puntaje(\epsilon_j)\ ] \quad \text{donde} \quad \epsilon_j \in X_i \tag{7.2}$$

Nótese como la función θ se puede plantear utilizando tanto el predicado *no satisface* $\oslash$ como el predicado *satisface* $\odot$. En ambos casos, cuando un elemento *no es aplicable* $\oplus$ para el requerimiento que se pretende calificar, el puntaje de ese elemento no se pierde. Es decir, se asume que el requerimiento *satisface* dicho elemento (*no es aplicable* $\oplus$ $\implies$ *satisface* $\odot$).

Ahora, para leer los resultados de θ, suponga que $x = \theta(R_e, \Gamma)$ es la calificación que recibe un requerimiento de software R_e cuando es evaluado contra el atributo *Precisión*. Entonces:

- Si $x = 10$ se dice que R_e satisface Γ y se denota $R_e \odot \Gamma$. Es decir, R_e es preciso pues cumple con todas las restricciones contempladas en los elementos en Γ.
- Si $x = 0$ se dice que R_e no satisface Γ y se denota $R_e \oslash^0 \Gamma$. Es decir, R_e no es preciso pues no cumple con ninguna de las restricciones contempladas en los elementos en Γ.

- Si $0 < x < 10$ se dice que R_e no satisface Γ con un grado x y se denota $R_e \oslash^x \Gamma$. En este caso, R_e cumple con algunas de las restricciones contempladas en los elementos pero no con todas.

En los casos intermedios ($0 < x < 10$), la deficiencia del requerimiento es más leve cuando el valor de x se acerca a 10, y es más grave cuando el valor de x se acerca a 0.

Note como se sobrecarga la notación $\odot$ y $\oslash$ para indicar el cumplimiento de un atributo total, más allá de un elemento. En adelante se usará tanto la notación $\oslash$ como la notación $\oslash^x$ de manera general cuando se quiere expresar que un requerimiento no safistace un atributo y donde no es relevante el grado de incumplimiento x.

Como ejemplo del uso de puntajes para la calificación, retomemos la definición planteada en la tabla 7.1 para el atributo de *Verificabilidad*, y consideremos el siguiente requerimiento de software hipotético: $R_2 =$"*The system shall not consider commands that have timed out.*". Para este ejemplo, repasando los elementos en Υ, se puede notar que: $R_2 \oslash \upsilon_1$ y $R_2 \oslash \upsilon_3$. Por lo tanto, para conocer la calificación que tiene R_2 para el atributo de *Verificabilidad*, se sustrae de 10 la suma de los puntajes para los elementos que R_2 no satisface $\oslash$ de la siguiente manera:

$$\theta(\ R_2\ ,\ \Upsilon\ ) \quad = \quad 10 - [\ \textstyle\sum_{i, R_2 \oslash \upsilon_i} puntaje(\upsilon_i)\ ] \ = \ 10 - [2{,}5 + 5] \ = \ 2{,}5$$

Del ejemplo presentado se puede concluir que $R_2 \oslash^{2,5} \Upsilon$ (*i. e.,* R_2 no satisface el atributo de verificabilidad con un grado $2{,}5$) y por ende R_2 no es un requerimiento de software verificable (según los elementos en Υ). Además, se puede ver que el grado de incumplimiento se acerca a 0 por lo que se puede concluir que el requerimiento tiene deficiencias graves.

Finalmente, para determinar la calificación global de un requerimiento respecto de los *tres atributos*, se crea una función simple $\phi : (R_k) \rightarrow \mathbb{R}^+_{10}$ de la siguiente manera:

$$\phi(R_k) = \frac{\sum_{i=1}^{3} \theta(R_k, X_i)}{3} \quad \text{donde } X_i \in \{\Lambda, \Gamma, \Upsilon\} \tag{7.3}$$

Es decir, $\phi(R_k)$ es simplemente una media aritmética de las calificaciones que obtiene un requerimiento para cada atributo $\theta(R_k, X_i)$ en (7.2). La calificación global representa una me-

dida general sobre la calidad del requerimiento y se podría utilizar para estimar costos en un proyecto de software.

Nuevamente, para leer el resultado de ϕ suponga que $x = \phi(R_k)$ es la calificación que recibe un requerimiento de software R_k cuando es evaluado contra los tres atributos (*no-ambigüedad*, *precisión* y *verificabilidad*) utilizando los elementos en $\{\Lambda, \Gamma, \Upsilon\}$. Entonces:

- Si $x = 10$ se concluye que R_k es no-ambiguo, preciso y verificable pues se cumple que $R_k \odot X_i \quad \forall_{X_i} \in \{\Lambda, \Gamma, \Upsilon\}$.
- Si $x = 0$ se concluye que R_k es impreciso, no-verificable y ambiguo pues se cumple que $R_k \oslash X_i \quad \forall_{X_i} \in \{\Lambda, \Gamma, \Upsilon\}$,
- Si $0 < x < 10$ se dice que R_k no satisface uno o más de los elementos en $\{\Lambda, \Gamma, \Upsilon\}$. En este caso, la función θ brindará más detalle para saber cuál es la debilidad que R_k presenta.

Con la notación que se acaba de definir, se tiene un lenguaje para discutir la evaluación de atributos de manera cuantificable. El siguiente paso consiste en determinar los elementos específicos a utilizar para evaluar los atributos. Existen muchos elementos relevantes que se pueden estudiar en los requerimientos de software para identificar posibles errores. Sin embargo, en la realidad –por diversas razones– solo es práctico utilizar una fracción de esa lista cuando se pretende realizar la evaluación de manera automática.

En las siguientes secciones se explica la metodología seguida durante el proceso de selección de los elementos a considerar para el prototipo.

7.2 Selección de Elementos

Para describir el proceso de selección de elementos primero se presenta la metodología en la sección 7.2.1 y posteriormente se describen los resultados en la sección 7.2.2.

7.2.1 Metodología

Para satisfacer los objetivos específicos 1 y 2 de la investigación ("*acotar los conceptos de precisión, ambigüedad y verificabilidad*" y "*describir elementos lingüísticos a utilizar para descubrir indicadores de (no) cumplimiento con los conceptos del objetivo 1* respectivamente"), se crea una lista de elementos para medir la presencia o ausencia de un atributo (*i. e., no-ambigüedad*, *precisión* y *verificabilidad*) en un requerimiento de software dado. De esta lista, se identifica un grupo de elementos cuya aplicación se puede automatizar en un tiempo y esfuerzo moderado como parte de la investigación. Este grupo de elementos se valida con profesionales como se describe más adelante (ver las secciones 7.3 y 7.4).

Se han utilizado dos fuentes de información principales para confeccionar la lista general de elementos. En primera instancia, se ha revisado literatura influyente en esta temática dentro del área de ingeniería de software (ver trabajos descritos en el capítulo 2) y se extrajeron recomendaciones que ayudan al análisis de lenguaje natural en el contexto de los requerimientos de software. En particular, se han usado criterios del trabajo de Berry *et al.* [2] y también de Lami *et al.* [17]. Por otra parte, también se ha utilizado la experiencia profesional del autor en el área de análisis de requerimientos de software para sistemas empotrados en aviónica. En este último caso, se ha usado como base documentación existente para proyectos previos realizados bajo el estándar DO-178B, en los cuales se han detectado y corregido fallas en los requerimientos de software.

7.2.2 Resultados y Análisis

Como resultado del proceso de observación, las tablas 7.2, 7.3 y 7.4 muestran la lista inicial de elementos que se consideran importantes para determinar rasgos de *imprecisión*, *ambigüedad* y *no-verificabilidad* respectivamente en un requerimiento de software. Debido a que la información que describe cada elemento es amplia, dichas tablas muestran solamente tres aspectos: nombre del elemento, descripción y un ejemplo que no satisface el criterio descrito en ese elemento. El apéndice B describe de manera más extensa cada elemento incluyendo la justificación sobre por qué dicho elemento fue considerado, ejemplos que cumplen con el criterio y ejemplos

que no lo cumplen, así como otros comentarios pertinentes.

Tabla 7.2: Lista general de elementos para determinar *Imprecisión*. La 1^{ra} columna es el nombre corto; la 2^{da} explica la restricción o regla que representa cada elemento, y la 3^{ra} muestra un ejemplo que incumple la restricción descrita en el elemento. El texto entre paréntesis no pertenece al requerimiento original, se agregó para ilustrar la deficiencia en cada caso. También se subrayan partes del requerimiento para indicar puntos claves a observar para identificar si el requerimiento satisface o no cada elemento.

Nombre	**Elemento**	**Contraejemplo**
Argumentos del verbo	Un requerimiento debe incluir todos los argumentos para verbos transitivos que necesitan un argumento adicional al complemento directo.	*The system shall increment the fault counter [by one] when a new fault is logged.*
Tolerancia para frecuencia	Un requerimiento debe indicar un valor de tolerancia para acciones que se realizan de manera periódica con mediciones físicas.	*The system shall send ARINC label 251 every 50 ms [+/- 5ms].*
Tolerancia para tiempo	Un requerimiento debe indicar un valor de tolerancia para mediciones de tiempo.	*The system shall log the CONN_BROKEN fault if an ACK is not received within 1s [+/- 100ms] after transmitting STX.*
Tolerancia para voltaje	Un requerimiento debe indicar un valor de tolerancia para las mediciones de voltaje y corriente eléctrica.	*The system shall enter LOW_PWR mode when voltage reading equals 20v [+/- 0.5v].*
Único *shall*	Un requerimiento no debe tener más de un "*shall*", es decir, no más de una acción principal.	*The system shall display altitude every 2 seconds and shall display temperature every 3 seconds.*
Referencias externas	Un requerimiento debe evitar referencias externas para describir una acción que puede ser descrita en el requerimiento mismo.	*The system shall enter INTERACTIVE mode as defined in section 4.2.4.*
Condición explícita	Un requerimiento debe incluir una condición que indique cuándo se debe realizar la acción principal.	*The system shall clean the DMA shared space [when IER0 equals 0x1D5].*
Voz activa	Un requerimiento debe estar escrito en voz activa y no en voz pasiva.	*Validity of all commands shall be verified by the IO Module.*
No determinismo	Un requerimiento no debe incluir formas no determinísticas que funcionan como adverbios.	*The system shall periodically perform CBITE.*
Verbos generales	Un requerimiento no debe utilizar verbos que indican procesos generales que no se pueden asociar directamente con una acción atómica o específica.	*The system shall monitor responses from the slave service.*

Los elementos en las tablas 7.2, 7.3 y 7.4 representan características generales que –idealmente– deben cumplir los requerimientos de software desde un punto de vista de correctitud del lenguaje, así como consideraciones técnicas respecto de la estructura general de los requerimientos de

Tabla 7.3: Lista general de elementos para determinar *Ambigüedad*. La 1^{ra} columna es el nombre corto; la 2^{da} explica la restricción o regla que representa cada elemento, y la 3^{ra} muestra un ejemplo que incumple la restricción descrita en el elemento. El texto entre paréntesis no pertenece al requerimiento original, se agregó para ilustrar la deficiencia en cada caso. También se subrayan partes del requerimiento para indicar puntos claves a observar para identificar si el requerimiento satisface o no cada elemento.

Nombre	**Elemento**	**Contraejemplo**
Agrupaciones explícitas	Un requerimiento debe describir las agrupaciones de ANDs y ORs mediante signos de puntuación adecuados o bien utilizando paréntesis explícitos.	*The system shall enter Normal mode when SDI field on label 227 equals 2 or SSM in label 268 equals 3 and WOW is true or AIR is false.*
Verbos ambiguos	Un requerimiento no debe utilizar como verbo principal un verbo que tenga múltiples significados en el contexto en que se utiliza.	*The system shall clean the display before a write operation.*
Adverbios vagos	Un requerimiento no debe utilizar adverbios vagos o adverbios generales para describir la acción principal.	*The system shall allow the operator to adjust volume to an acceptable level.*
No determinismo	Un requerimiento no debe utilizar estructuras o formas que se saben no determinísticas.	*The system shall display altitude and/or temperature on the botton line of the screen.*
Scope ambiguity	Un requerimiento debe evitar múltiples interpretaciones debido a que el ámbito de los cuantificadores no es claro.	*The system shall keep a unique log file for all IO interfaces.*

software en aviónica. Sin embargo, solo un subconjunto de los elementos de cada atributo fue seleccionado para formar parte de los elementos que va a evaluar el prototipo, este subconjunto se muestra en la tabla 7.5.

Es importante subrayar que los elementos listados en las tablas 7.2 a 7.4 (y su versión extendida en el apéndice B) han sido descritos de manera muy puntual con la intención de que puedan ser aplicados incluso sin tener que utilizar una herramienta de apoyo. De este modo, dichas listas ya constituyen un aporte modesto en un proceso manual de revisión de requerimientos. Este aporte se clasifica dentro de la lista de trabajos que Lami *et al.* [17] distinguen como del tipo analítico.

Como un resultado parcial para esta etapa, la tabla 7.5 muestra la lista de elementos seleccionados para ser evaluados por el prototipo. Por razones prácticas, y debido a que este estudio representa fundamentalmente una prueba de concepto sobre el tema de fondo (análisis parcial de requerimientos de software) se han dejado algunos elementos por fuera del prototipo.

Tabla 7.4: Lista general de elementos para determinar *No-verificabilidad*. La 1^{ra} columna es el nombre corto; la 2^{da} explica la restricción o regla que representa cada elemento, y la 3^{ra} muestra un ejemplo que incumple la restricción descrita en el elemento. El texto entre paréntesis no pertenece al requerimiento original, se agregó para ilustrar la deficiencia en cada caso. También se subrayan partes del requerimiento para indicar puntos claves a observar para identificar si el requerimiento satisface o no cada elemento.

Nombre	Elemento	Contraejemplo
Requerimiento negativo	Un requerimiento no debe expresar la acción principal como una acción negativa.	*The system shall not enter INTERACTIVE mode while on air.*
Uso de *only*	Un requerimiento debe utilizar el término "*only*" correctamente cuando este trata de restringir únicamente la acción del verbo principal.	*The system shall only display pressure and altitude while in manual mode.*
Requerimiento infinito	Un requerimiento no debe utilizar los términos "*always*" y "*never*" para restringir la acción que describe un requerimiento.	*The system shall never perform BITE during TAKE_OFF.*
Acciones humanas	Un requerimiento no debe utilizar verbos que describan acciones que no se puedan atribuir a un computador.	*The system shall consider fault history during CBITE.*

No obstante, es preciso mencionar que, computacionalmente, es factible implementar mecanismos de análisis de texto para identificar la presencia de esos otros elementos por ahora descartados.

Los elementos que no han sido seleccionados para formar parte del prototipo en esta investigación se excluyeron principalmente por dos razones: (a) no ocurren con tanta frecuencia según la experiencia profesional del autor, y (b) para automatizar la búsqueda de ciertos elementos, se deben utilizar técnicas que escapan al alcance de esta investigación.

Es válido preguntarse si es necesario aplicar todos estos criterios para cualquier proceso de revisión de requerimientos. La respuesta es que la cantidad de criterios que se aplican para medir la calidad de los requerimientos de software puede variar de un proyecto a otro. Generalmente, cuando existe un estándar para desarrollo de requerimientos, dicho estándar lista los criterios a utilizar dependiendo de la rigurosidad con que se necesite revisar los requerimientos en ese caso. Por ejemplo, no es inusual encontrar documentos de requerimientos de software que violan los elementos 5 y 6 de la tabla 7.2 ("Único *shall*" y "Referencias externas") y ambos elementos son problemáticos. Por un lado, el primero permite que existan requerimientos compuestos;

Tabla 7.5: Elementos utilizados en el prototipo. Se muestran los elementos de cada atributo que fueron seleccionados para el prototipo de entre el total de elementos originalmente identificados. La 1^{ra} columna indica el atributo; la 2^{da} indica el número de elementos en la lista inicial; la 3^{ra} indica el número de elementos seleccionados y la 4^{ta} lista los elementos seleccionados para cada atributo con respecto a las tablas 7.2, 7.3 y 7.4.

Atributo	Disponibles	Seleccionados	Elementos
Precisión	10	6	tolerancia para frecuencia, tolerancia para tiempo, tolerancia para voltaje, condición explícita, no determinismo, verbos generales.
Ambigüedad	5	4	agrupaciones explícitas, verbos ambiguos, adverbios vagos, no determinismo
Verificabilidad	4	4	requerimiento negativo, uso de *only*, requerimiento infinito, acciones humanas

un requerimiento de software se llama “compuesto” cuando especifica múltiples acciones que debe realizar el sistema bajo una o más condiciones. El problema en estos casos es que cuando una de estas acciones falla, como resultado de una prueba, no hay forma clara de contabilizar cuál de las acciones ha fallado, pues en caso de que las otras acciones funcionen como se espera, es incorrecto suponer que todo el requerimiento falla. Por otro lado, el elemento 6 de la tabla 7.2 (“Referencias externas”) evita que que existan requerimientos que potencialmente sub-especifican una cierta funcionalidad al hacer referencias a tablas, figuras u otras secciones de un documento. *Sub-especificar* una acción es describirla de manera pobre e insuficiente, sin ofrecer todo el detalle que se requiere para lograr una descripción clara de la acción. A menudo estas referencias no son precisas, con lo que no se logra describir la acción del requerimiento de manera adecuada.

Aunque existan estudios que ofrezcan una guía o apoyo para el proceso de revisión de requerimientos, el problema de determinar el nivel de rigurosidad con que estos se revisan es fundamentalmente una cuestión de diseño y estrategia que varía para cada proyecto de software, y depende en gran medida del área o tipo de industria al que pertenece el proyecto. En el caso particular de revisión de requerimientos de software para sistemas empotrados en la industria de aviónica, sí se acostumbra aplicar procesos de revisión rigurosos ya que dicha tarea es parte de la normativa descrita en el estándar DO-178B.

En este punto, luego de que el autor seleccionó el conjunto de elementos de la tabla 7.5, es ne-

cesario determinar si otros profesionales están de acuerdo en que dicha selección es apropiada. Este proceso de validación de los elementos se describe en la siguiente sección.

7.3 Validación de Selección

El proceso de validación independiente tiene como objetivo revisar que la selección de elementos de la etapa anterior sea representativa para los tres atributos que se estudian en esta investigación. Para ello fue posible contar con el aporte de tres profesionales[2] experimentados en análisis de requerimientos de software quienes ayudaron a revisar que la selección de elementos fuera útil. Los tres profesionales que participaron en este proceso son: Ing. Eduardo Trejos, Ing. Roger Fernández e Ing. Leonardo Jiménez. El currículum vitae de estas personas se ha adjuntado en el Apéndice I.

La sección 7.3.1 describe la metodología utilizada durante el proceso de validación de la selección de elementos y luego, la sección 7.3.2 presenta los resultados de dicho proceso.

7.3.1 Metodología

Para realizar la validación, se elaboró un instrumento de evaluación para identificar si el grupo de profesionales consultado consideraba que la selección de elementos propuesta por el autor era relevante (el instrumento se encuentra en el Apéndice C).

Para la elaboración de este instrumento se tomaron en cuenta varios aspectos. Primero, se consideró importante que el mismo mostrara ejemplos reales de los tipos de errores que se pretende identificar y no solamente la descripción abstracta del error. Segundo, cuando el evaluador no tiene criterio para contestar una pregunta, el instrumento permite expresar esa condición y continuar con la evaluación. Además, para los casos en que el evaluador requiere agregar información adicional a sus respuestas, el instrumento provee espacio en todas las preguntas para recibir esta retroalimentación.

[2]Estas tres personas son ingenieros de software con entrenamiento en varios temas relacionados con la labor de validación y verificación de software para sistemas empotrados en aviónica. Además, los tres tienen al menos dos años de experiencia en la realización de tareas afines a la revisión de requerimientos de software en esta industria.

Un aspecto que se consideró cuidadosamente fue la secuencia y el tipo de preguntas que se debían realizar. El objetivo principal es recolectar la mayor cantidad de información de los evaluadores evitando a la vez introducir vicios en las preguntas. Con este objetivo, se diseñaron tres preguntas para cada elemento. En cada caso, la primera pregunta tiene como objetivo que el evaluador "identifique el error" en el requerimiento. La segunda pregunta depende de la respuesta de la primera y tiene como objetivo que el evaluador "generalice" el error; y la tercera pregunta, que depende del resultado de la segunda, tiene como propósito que el evaluador logre "clasificar el error" en las categorías de *impreciso*, *ambiguo* o *no verificable*. La estructura del instrumento se ilustra en la Figura 7.1.

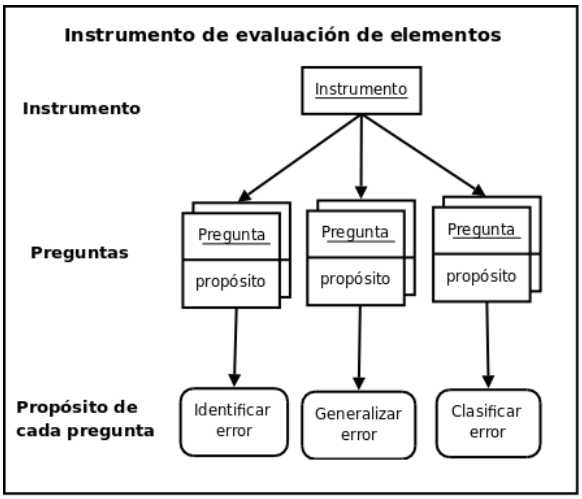

Figura 7.1: Diagrama del instrumento de evaluación de elementos. El instrumento tiene varias preguntas, cada una de las cuales tiene un propósito específico para el evaluador: *identificar un error*, *generalizar un error*, o *clasificar un error*.

En el instrumento de evaluación, las preguntas destinadas a "identificar el error" muestran uno o más ejemplos de requerimientos de software seguidos por una afirmación que se hace sobre esos ejemplos. Para ilustrar, una posible afirmación sería: "*Uno de los errores de este requerimiento de software es que carece de un valor de tolerancia para la restricción de tiempo*". Esta pregunta ofrece las siguientes posibilidades de respuesta: (a) "*De acuerdo*", (b) "*El requerimiento no tiene errores*", (c) "*No tengo criterio para decidir*" y (d) "*El requerimiento tiene errores pero el indicado no es uno de ellos*".

En el caso de las preguntas destinadas a "generalizar el error", se hace alusión al error identificado en la pregunta anterior (siempre y cuando el participante estuviese de acuerdo con que existe un error), y se le consulta al participante si está de acuerdo con que el error señalado en la consulta anterior constituye un error en general para cualquier requerimiento de software. Las opciones de respuesta en este caso son (a) "*De acuerdo*" y (b) "*En desacuerdo*".

Finalmente, para aquellas preguntas destinadas a "clasificar el error", el participante tiene la oportunidad de clasificar el error que ya se ha generalizado en la pregunta anterior (nuevamente, esto es cierto solo cuando se está de acuerdo en que sí hay un error). El participante tiene las siguientes opciones de respuesta para clasificar el error: (a) "*Precisión*", (b) "*Ambigüedad*", (c) "*Verificabilidad*", (d) "*Otro*" y (e) "*No tengo criterio para decidir*".

En todas las preguntas de la evaluación, el participante tenía la posibilidad de explicar con más detalle su respuesta. Adicionalmente, el instrumento también permitía al participante sugerir nuevos criterios que consideraba importantes y que no habían sido evaluados en el instrumento.

7.3.2 Resultados y Análisis

La tabla 7.6 muestra los resultados obtenidos para la evaluación de los elementos. En la tabla se muestra el porcentaje de coincidencias entre las estimaciones del autor y las respuestas proporcionadas por el grupo de profesionales.

Tabla 7.6: Coincidencias entre el autor y los profesionales para la evaluación de elementos. La 1^{ra} columna muestra el porcentaje de coincidencias para las preguntas cuyo propósito era identificar el error; la 2^{da} para generalización del error, y la 3^{ra} para clasificación del error. Estos valores se calcularon como un promedio de las coincidencias entre cada profesional y el autor para todas las preguntas del instrumento.

Intención de la pregunta		
Identificación del error	Generalización del error	Clasificación del error
100 %	100 %	79 %

Los datos en la tabla 7.6 indican que todos los participantes están de acuerdo en que los errores señalados en el instrumento son auténticos errores. Este dato se deriva del valor de

$100\,\%$ para las preguntas cuyo objetivo era identificar el error, es decir, las preguntas de *1* a *3*, y de *5* a *15*. Además, en su totalidad los participantes afirmaron estar de acuerdo en que los errores señalados no son específicos de la muestra y que sí se pueden generalizar. Esto se deduce del valor de $100\,\%$ para las preguntas cuyo objetivo es generalizar el error. Las preguntas en esta categoría van de la *1.a* hasta la *3.a*, la *4*, y luego de la *5.a* hasta la *15.a*.

Finalmente, para las preguntas destinadas a clasificar el error, el porcentaje de coincidencias es de $79\,\%$. Las preguntas en esta categoría van de la *1.b* hasta la *3.b* y luego de la *5.b* hasta la *15.b*. Las diferencias en la clasificación de los errores se notaron en las respuestas a las preguntas *5.b*, *6.b* y *7.b* como se muestra en la tabla 7.7. El Apéndice D muestra la tabla completa con los resultados obtenidos mediante este instrumento.

Tabla 7.7: Coincidencias en la clasificación de elementos. La 1^{ra} columna indica la categoría; en las siguientes columnas, la etiqueta (1.b, 2.b, ...) se refiere a cada pregunta en el instrumento del Apéndice C. Las celdas de color gris representan la clasificación propuesta por el autor para el error en cuestión, y los valores en cada celda indican el número de personas que clasificó el error en la categoría representada por esa fila.

Categoría	**Preguntas de Clasificación de Errores**													
	1.b	2.b	3.b	5.b	6.b	7.b	8.b	9.b	10.b	11.b	12.b	13.b	14.b	15.b
Impreciso	3	3	3										1	
No-verificable				1	1	1	1				3	3	2	3
Ambiguo				2	2	2	3	3	3	3	1			

A excepción de las preguntas *5*, *6* y *7*, la categoría para cada uno de los errores identificados en el instrumento se determinó de acuerdo con el criterio de la mayoría de los evaluadores. En este caso hay tres votos de los profesionales y un voto del autor. En los tres casos donde se presentaron diferencias grandes (preguntas *5*, *6* y *7*), el autor clasifica los errores en la categoría de *Precisión* según se justifica a continuación.

El error que se trata de subrayar en la pregunta número *5.b* (RS-5: "*The system shall perform a CRC test of the config data in memory*.") es el hecho de que los requerimientos de software que no tienen una condición explícita son potencialmente imprecisos. La excepción a esta norma son los requerimientos que describen acciones que el sistema debe realizar periódicamente[3] (en cuyo caso se especifica una frecuencia); y los requerimientos de software que describen acciones

[3] *p. ej., The system shall refresh the display at 60Hz.*

que se ejecutan una única vez[4] (en cuyo caso la condición está implícita). Sin embargo, luego de conversar con los evaluadores posterior al experimento, se concluyó que el requerimiento de software que se presentó en este ejemplo desvió la atención de los evaluadores, ya que la acción de realizar la prueba de *CRC* (*cyclical redundancy check*) también está descrita de manera insuficiente (pues se desconoce, entre otras cosas, el polinomio a utilizar en la prueba y el valor esperado contra el cual se debe comparar el resultado). Bajo estas circunstancias, los evaluadores clasificaron el error como *No-verificable* en dos casos y como *Ambiguo* en un caso. Sin embargo, se tomó la decisión de que el prototipo continúe clasificando dicho error en la categoría de *Precisión* ya que las discrepancias surgen de haber presentado un ejemplo confuso y, aunque su justificación es válida, esta es de índole semántico y escapa al alcance de esta investigación.

Por otro lado, el error que se trata de subrayar en la pregunta número *6.b* (RS-6: "*The system shall continually perform a wrap test with the display head microcontroller.*") es el hecho de que adverbios inexactos o no-determinísticos no se deben utilizar en un requerimiento de software para describir una acción. En el ejemplo se utilizó el adverbio *continually* para describir la acción que el software debe realizar. En este sentido, la forma en que este adverbio califica al verbo principal en el requerimiento representa la condición bajo la cual se debe ejecutar dicha acción. En el fondo, este requerimiento describe una acción que se debe realizar periódicamente. Sin embargo, el uso del término *continually* hace esta condición imprecisa. Es decir, no se puede determinar con certeza cuál es el período de la acción y por ende cualquier interpretación es arbiraria. Nuevamente, los evaluadores clasificaron el error como *No-verificable* en dos casos y como *Ambiguo* en un caso. Esta clasificación no es del todo incorrecta pues, al no saber con exactitud las condiciones bajo las cuales el software debe ejecutar una acción, el requerimiento se vuelve no-verificable. Luego, si alguien insiste en escribir una prueba, lo primero que debe hacer es adoptar una interpretación del término *continually*, y al hacer esto se encuentra el problema de ambigüedad. Sin embargo, al discutir con los profesionales luego del experimento, se llegó a la conclusión de que el prototipo continúe clasificando dicho error en la categoría de *Precisión* ya que este es en sí el problema de fondo.

[4] *p. ej., The system shall initialize the IOMC register to 0xFD.*

Finalmente, el error señalado en la pregunta número *7.b* (RS-7: "*The system shall process VHF signals from label 0xDB8 every second (+/- 100 ms).*" y RS-8: "*The system shall monitor the status of the VHF line when landing.*") es el hecho de que en los requerimientos de software, es incorrecto utilizar verbos generales que no se pueden asociar a una acción atómica o específica. En general, los términos *process* y *monitor* denotan acciones que, a simple vista, parecen tareas comunes para un software. Sin embargo, estos verbos no son suficientemente descriptivos, por lo que no se pueden implementar, ni tampoco verificar, sin contar con una interpretación dentro del contexto propio del sistema. En este caso, la clasificación de los evaluadores fue idéntica a los dos casos anteriores. Sin embargo, también en esta oportunidad se insiste en que los errores no hacen por sí mismos que el requerimiento sea no-verificable o ambiguo. El problema de fondo es que el verbo no es preciso, es decir, la acción no es precisa tal como se expresa; sin embargo, utilizar un verbo más específico ya sería una solución al problema. De manera similar a los casos anteriores, en una sesión con los evaluadores se discutieron los puntos de vista y se concluyó utilizar como categoría predominante la de precisión.

Analizando los resultados del proceso de evaluación de la selección, se deja en firme la selección de elementos inicial que se detalla en la tabla 7.5.

El siguiente paso en la metodología es asignar un puntaje o valor a cada uno de los elementos, los detalles de cómo se llevó a cabo este proceso se explican en la siguiente sección.

7.4 Asignación de Puntajes

El objetivo en esta etapa es asignar un puntaje a cada uno de los elementos seleccionados en el paso anterior y validar dicha asignación con un grupo de profesionales. El concepto de puntaje que se utiliza en esta sección es el que se presentó en la sección 7.1. La sección 7.4.1 describe la metodología utilizada durante el proceso de asignación de puntajes y luego, la sección 7.4.2 presenta los resultados de dicho proceso.

7.4.1 Metodología

Para realizar esta tarea se preparó otro instrumento que fue aplicado al mismo grupo de tres ingenieros que colaboraron con la selección de elementos. Este instrumento lista los elementos seleccionados en el paso anterior y los evaluadores deben clasificar cada elemento en una escala de *1* a *3*. Aunque los puntajes para los elementos se deben asignar en una escala de *1* a *10* (ver sección 7.1.2), en este caso por conveniencia y simplicidad se utilizó una escala de *1* a *3* de manera que "*1*" significa que el elemento observado aporta poca evidencia de que el requerimiento viola uno de los atributos estudiados en la investigación; "*2*" significa que el elemento observado aporta evidencia moderada de que el requerimiento viola uno de los atributos, y "*3*" significa que el elemento observado aporta mucha evidencia de que el requerimiento viola uno de los atributos. Estos valores se utilizan posteriormente para calcular el puntaje de cada elemento.

Durante la evaluación, los profesionales determinaron si el valor sugerido por el autor fue apropiado para cada elemento y sugerían nuevos valores cuando lo consideraban necesario. Los valores asignados por el autor obedecen principalmente a la experiencia de campo acumulada durante su labor profesional y académica.

La tabla 7.8 muestra los valores iniciales propuestos por el autor previo a la evaluación de los profesionales. Nótese que en la tabla solo hay cuatro elementos en la categoría de *Imprecisión*, y anteriormente se indicó que el total para esa categoría es seis. Esta diferencia se debe a que los elementos 2, 3 y 4 se han tomado como uno solo pues son de naturaleza similar (ausencia de valores de tolerancia) y se les asignará el mismo puntaje.

En este instrumento, que se adjunta en el Apéndice E, las preguntas están separadas en tres grupos (que representan las categorías de interés en el estudio). En cada pregunta de este instrumento, el evaluador tenía tres opciones de respuesta: (a) estar de acuerdo con el puntaje sugerido, (2) no estar de acuerdo con el puntaje sugerido y proporcionar un puntaje nuevo, y (c) no responder a la pregunta, si considera que no tiene suficiente criterio para tomar una determinación. Nótese que en este instrumento el término *puntaje* aún se refiere al valor en escala *1-3* y no a la definición de puntaje descrita en la sección 7.1.2.

Tabla 7.8: Clasificación de elementos propuesta por el autor. La 1^{ra} columna muestra la categoría o el tipo de error que representa cada elemento. La 2^{da} columna indica el número de la pregunta en el instrumento del apéndice E. La 3^{ra} es el nombre del elemento y la 4^{ta} es el valor que el autor asignó inicialmente a cada elemento en una escala de *1* (leve) a *3* (grave).

Categoría	Pregunta	Elemento	Valor Sugerido
Imprecisión	1	Tolerancia para frecuencia Tolerancia para tiempo Tolerancia para Voltaje	2
	2	Condición explícita	3
	3	No determinismo	1
	4	Verbos generales	1
No verificabilidad	5	Requerimiento negativo	2
	6	Uso de *only*	3
	7	Requerimiento infinito	2
	8	Acciones humanas	1
Ambigüedad	9	Agrupaciones explícitas	2
	10	Verbos ambiguos	1
	11	Adverbios vagos	2
	12	No determinismo	2

7.4.2 Resultados y Análisis

Luego de aplicar el instrumento, los resultados del proceso se resumen en la tabla 7.9.

La tabla 7.9 muestra que todos los participantes contestaron las preguntas con las opciones (a) o (b), es decir, estaban de acuerdo con el valor sugerido, o no estaban de acuerdo y procedieron a sugerir otro. De este modo, el resultado que se utilizó fue el promedio de los valores indicados por los participantes. El Apéndice F muestra la tabla completa con los resultados de la aplicación de este instrumento. La cuarta columna en la tabla 7.9 muestra, a modo de referencia, el valor inicial propuesto por el autor para cada uno de los elementos. La quinta columna muestra efectivamente el valor resultante para cada elemento. La última columna muestra un balance o diferencia entre el valor propuesto por el autor y el valor resultante al consultar a los profesionales.

Nótese como en un $58{,}3\,\%$ (7 de 12) de los casos los profesionales sugieren que el valor de los elementos se debe incrementar en relación con el valor propuesto por el autor. Los evaluadores indican que este grupo de elementos revela problemas de mayor gravedad que lo que el autor

Tabla 7.9: Resultados de la clasificación de elementos. La 1^{ra} columna es el tipo de error que representa cada elemento. La 2^{da} columna es el número de la pregunta en el instrumento del apéndice E. La 3^{ra} es el nombre del elemento. La 4^{ta} es el valor que el autor asignó a cada elemento, la 5^{ta} es el valor que resultó del instrumento y la 6^{ta} es la diferencia entre las columnas 4 y 5.

Categoría	**Pregunta**	**Elemento**	**Valor Propuesto**	**Valor Resultante**	**Diferencia**
Imprecisión (Γ)	1	Tolerancia para frecuencia Tolerancia para tiempo Tolerancia para Voltaje	2	2.33	0.33
	2	Condición explícita	3	3	0
	3	No determinismo	1	1.67	0.67
	4	Verbos generales	1	1.33	0.33
No verificabilidad (Υ)	5	Requerimiento negativo	2	2.67	0.67
	6	Uso de *only*	3	2.67	-0.33
	7	Requerimiento infinito	2	3	1
	8	Acciones humanas	1	1.67	0.67
Ambigüedad (Λ)	9	Agrupaciones explícitas	2	2	0
	10	Verbos ambiguos	1	1.67	0.67
	11	Adverbios vagos	2	2	0
	12	No determinismo	2	2	0

sugería (y por eso su valor está más cerca del tope de 3). Esto sugiere que la evaluación del autor fue ligeramente conservadora para ese grupo particular de elementos. En un $33{,}3\,\%$ (4 de 12) los profesionales estuvieron de acuerdo con la asignación inicial del autor y, finalmente, en un $8{,}3\,\%$ (solo uno) de los casos, sugieren que el valor sea menor al que el autor propuso.

Retomando la nomenclatura y los planteamientos presentados en la sección 7.1, ya se tiene lo necesario para definir los elementos que forman los conjuntos $\{\Lambda, \Gamma, \Upsilon\}$ (*Ambigüedad*, *Precisión* y *Verificabilidad*) así como el puntaje respectivo para cada elemento. Este grupo de elementos junto con su respectivo puntaje constituye el corazón de la herramienta a crear pues, una vez asociados estos elementos con sus respectivos puntajes, ya será posible que el prototipo aplique las fórmulas (7.2) y (7.3) para determinar el estado de cada requerimiento respecto del modelo planteado en esta sección.

Se definen los conjuntos $\{\Lambda, \Gamma, \Upsilon\}$ de la siguiente manera:

$\Lambda = \{\lambda_1, \lambda_2, \lambda_3, \lambda_4\} = \{$agrupaciones explícitas, verbos ambiguos, adverbios vagos, no determinismo$\}$ en la tabla 7.3.

$\Gamma = \{\gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5, \gamma_6\} = \{$tolerancia para frecuencia, tolerancia para tiempo, tolerancia para voltaje, condición explícita, no determinismo, verbos generales$\}$ en la tabla 7.2.

$\Upsilon = \{\upsilon_1, \upsilon_2, \upsilon_3, \upsilon_4\} = \{$requerimiento negativo, uso de *only*, requerimiento infinito, acciones humanas$\}$ en la tabla 7.4.

La asignación de puntajes en el conjunto de elementos se realiza utilizando los resultados en la tabla 7.9. Luego, para cumplir con la restricción de la fórmula 7.1, se debe hacer una transformación sencilla que toma los valores indicados en la tabla 7.9 (columna indicada como "Valor Resultante") que están descritos en una escala de *1-3* y los convierte en su equivalente en una escala de *0-10*.

La razón de pasar de una escala a otra es principalmente por conveniencia en la implementación. Se consideró que al momento de consultar a los profesionales la escala de *1-3* era suficientemente clara para representar valores *bajo*, *medio* y *alto*. Sin embargo, la escala de *0-10* suele ser más apropiada para representar los resultados de una evaluación como la que hace el prototipo sobre los requerimientos.

Por ejemplo, para los elementos del grupo de *Ambigüedad* (Λ) en la tabla 7.9 se puede leer que los valores resultantes para los elementos, en la escala de *1-3*, son: $\{\ 2\ ,\ 1{,}67\ ,\ 2\ ,\ 2\ \}$ y de estos se obtiene su equivalente en escala *0-10* resolviendo la ecuación simple de una incógnita $3(2x) + 1{,}67x = 10 \quad \Rightarrow \quad x = 1{,}304$
con lo que los puntajes en la escala deseada son:
$\{\ (2{\times}1{,}304)\ ,\ (1{,}67{\times}1{,}304)\ ,\ (2{\times}1{,}304)\ ,\ (2{\times}1{,}304)\ \} \quad \Rightarrow \quad \{\ 2{,}61\ ,\ 2{,}18\ ,\ 2{,}61\ ,\ 2{,}61\ \}$
es decir
$puntaje(\lambda_{\{1,3,4\}}) = 2{,}61$ y $puntaje(\lambda_2) = 2{,}18$ y se cumple (7.1) ya que:

$$Puntaje(\Lambda) = \Big\{ \quad \sum_{i=1}^{4} puntaje(\lambda_i) \quad | \ \lambda_i \in \Lambda \Big\} = 10$$

De manera análoga, utilizando los valores para los elementos del grupo de *No verificabilidad* (Υ) e *Imprecisión* (Γ), es decir:
$\{\ 2{,}67\ ,\ 2{,}67\ ,\ 3\ ,\ 1{,}67\ \}$ y $\{\ 2{,}33\ ,\ 2{,}33\ ,\ 2{,}33\ ,\ 3\ ,\ 1{,}67\ ,\ 1{,}33\}$ y las ecuaciones:
a) $3x + 2(2{,}67x) + 1{,}67x = 10 \quad \Rightarrow \quad x \simeq 1$

b) $3x + 3(2{,}33x) + 1{,}67x + 1{,}33x = 10 \quad \Rightarrow \quad x = 0{,}77$

Se obtienen los puntajes en escala *0-10* para Υ y Γ respectivamente:
$\{\ 2{,}67\ ,\ 2{,}67\ ,\ 3\ ,\ 1{,}67\ \}$ y $\{\ 1{,}79\ ,\ 1{,}79\ ,\ 1{,}79\ ,\ 2{,}31\ ,\ 1{,}29\ ,\ 1{,}02\}$
Y se puede revisar que:

$$Puntaje(\Upsilon) = \Big\{ \quad \sum_{i=1}^{4} puntaje(\upsilon_i) \quad | \ \upsilon_i \in \Upsilon \Big\} = 10 \quad \text{y}$$
$$Puntaje(\Gamma) = \Big\{ \quad \sum_{i=1}^{6} puntaje(\gamma_i) \quad | \ \gamma_i \in \Gamma \Big\} = 10$$

Resumiendo, la tabla 7.10 muestra la información tal y como será utilizada más adelante en la construcción del prototipo.

Tabla 7.10: Elementos y puntajes en $\{\Lambda, \Gamma, \Upsilon\}$. La 1^{ra} columna muestra la categoría o el tipo de error que representa cada elemento. La 2^{da} columna indica el nombre del elemento y la 3^{ra} es el puntaje (en escala *0-10*) que se utilizará en el prototipo para cada elemento.

Categoría	Elemento	Puntaje
Imprecisión (Γ)	Tolerancia para frecuencia	1.79
	Tolerancia para tiempo	1.79
	Tolerancia para Voltaje	1.79
	Condición explícita	2.31
	No determinismo	1.29
	Verbos generales	1.02
No verificabilidad (Υ)	Requerimiento negativo	2.67
	Uso de *only*	2.67
	Requerimiento infinito	3.00
	Acciones humanas	2.67
Ambigüedad (Λ)	Agrupaciones explícitas	2.61
	Verbos ambiguos	2.18
	Adverbios vagos	2.61
	No determinismo	2.61

Para cerrar este capítulo, una vez que se tienen identificados los elementos y los puntajes para dichos elementos, se procede con el diseño conceptual y la implementación del prototipo. La metodología y resultados de dicho proceso se detalla en el capítulo 8.

8

Diseño y Construcción del Prototipo

En este capítulo se describe la metodología y resultados del proceso de construcción del prototipo como parte del objetivo específico 3. El proceso de diseño y construcción del prototipo se divide en dos partes principales: normalización de requerimientos y análisis de elementos. La primera parte se refiere al procesamiento de los datos de entrada que recibe el prototipo para organizarlos de manera adecuada. La segunda parte corresponde al corazón de la herramienta que realiza el análisis de los requerimientos.

La sección 8.1 describe la metodología y resultados para la fase de normalización de requerimientos. Seguidamente, la sección 8.2 detalla la metodología y resultados para la fase de análisis de elementos. Esta última sección explica los detalles técnicos sobre el funcionamiento del prototipo de software construido.

La figura 8.1 es un diagrama de flujo que muestra las etapas principales del prototipo que se ha elaborado. El proceso comienza tomando como entrada un documento de requerimientos de software escritos en lenguaje natural y la primera tarea consiste en normalizar los requerimientos en un formato estandarizado que funciona como entrada para el proceso posterior de análisis (sección 8.1). Una vez normalizada la lista de requerimientos, se procede con el análisis de cada uno de ellos para determinar si satisfacen o no los elementos en $\{\Lambda, \Gamma, \Upsilon\}$ (sección 8.2) y finalmente se producen algunos reportes con los resultados de la evaluación.

8.1 Normalización de Requerimientos

Los requerimientos de software generalmente se escriben en documentos de texto. Tanto el formato como el tipo de información que acompaña a cada requerimiento suele variar entre un

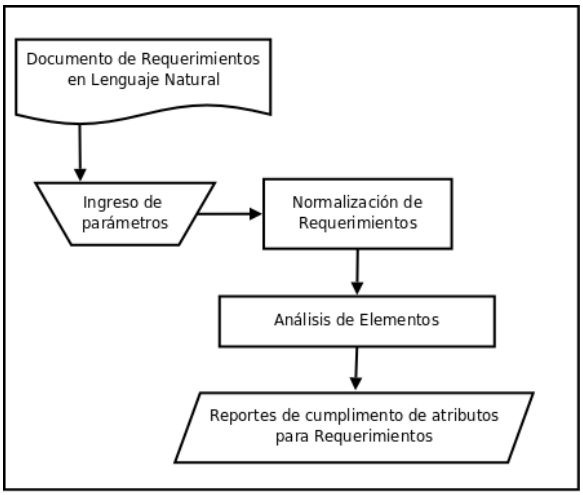

Figura 8.1: Diagrama de flujo general en el prototipo. La figura muestra el flujo general de la herramienta a un nivel macro comenzando por el documento de requerimientos de software que se debe evaluar, normalización de los requerimientos a examinar, análisis de los requerimientos con base en los elementos o criterios seleccionados y finalmente salidas del prototipo.

proyecto de software y otro. Por ejemplo, la numeración, el estilo de redacción y el uso de tablas o figuras, entre otros, son algunos de los aspectos que más a menudo cambian.

Existen herramientas en el mercado que facilitan la administración de los requerimientos en gran medida[1]. Estas herramientas proveen una amplia gama de funcionalidades que facilitan la administración (definición, revisión, mantenimiento) de los requerimientos durante el ciclo de vida del software. En particular, una de las funcionalidades comunes en dichas herramientas es exportar los requerimientos hacia un documento de texto dejándolos en un formato específico definido por el usuario. Una vez que los requerimientos se encuentran en un formato apropiado, el proceso de análisis de texto se puede realizar con menor dificultad.

De acuerdo con la experiencia del autor, en la industria de aviónica las herramientas especializadas para administración de requerimientos no son ampliamente utilizadas. Por lo general los requerimentos de software se escriben en documentos de texto como *MS Word*[2] o en hojas

[1]Algunas de las más conocidas son Telelogic DOORS (`http://www.telelogic.com/products/doors/`), IBM Rational RequisitePro (`http://www-01.ibm.com/software/awdtools/reqpro/`) y Borland CaliberRM (`http://www.borland.com/us/products/caliber/index.html`).

[2]`http://office.microsoft.com/word`

electrónicas como *MS Excel*[3]. Para una herramienta como la que se crea en esta investigación es importarte poder extraer los requerimientos del documento fuente original de manera automática evitando que una persona deba hacer el trabajo tedioso de producir una lista de requerimientos. La siguiente sección explica el trabajo que se hizo en esta investigación para lograr extraer los requerimientos de distintas fuentes y normalizarlos en un formato sencillo.

8.1.1 Metodología

El primer paso para normalizar los datos de entrada del prototipo es definir, conceptualmente, una estructura adecuada en la cual se puedan representar los requerimientos sin importar la fuente de la que estos vengan. La estructura a crear debe ser sencilla para que se adapte lo mejor posible a los diversos sabores (o formatos) en que se escriben los requerimientos. Una manera de lograr que la estructura propuesta sea suficientemente flexible y sencilla de manipular es utilizando *XML* como lenguaje de representación de datos.

El siguiente paso es examinar cada uno de los documentos fuente que contienen los requerimientos y encontrar patrones o estructuras regulares que demarquen el texto de los requerimientos, de manera que se pueda delimitar dónde comienza y dónde termina cada requerimiento. Por ejemplo, los documentos de texto en formato *MS Word* y *MS Excel* generalmente se pueden exportar a un archivo de texto en formato *XML*. En este caso, buscar un patrón es sencillo pues el requerimiento estará estructurado con etiquetas *XML* por lo que es posible demarcar el inicio y fin de cada requerimiento.

El tercer paso en este proceso es construir un programa que lea el archivo *XML*, identifique los requerimientos de software allí descritos y los extraiga, dejándolos en otro archivo de texto con la estructura definida para tal efecto.

La figura 8.2 muestra el proceso por el cual deben pasar los requerimientos de software antes de poder ser procesados por el prototipo desarrollado en esta investigación. En la siguiente sección se describen los detalles sobre la implementación de esta herramienta de normalización de requerimientos.

[3] `http://office.microsoft.com/excel`

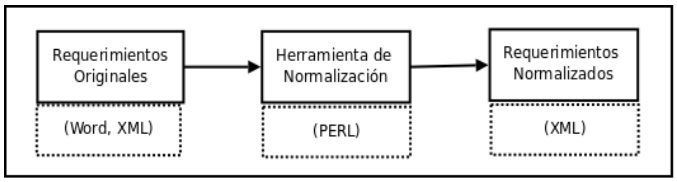

Figura 8.2: Diagrama del proceso de normalización de requerimientos. Los requerimientos se encuentran originalmente en formato de texto con alguna estructura definida. La herramienta se encarga de extraer los requerimientos y como salida produce un documento en formato *XML* con la estructura que se muesta en la figura 8.3

8.1.2 Implementación

Como se describió anteriormente, es necesario definir una estructura sencilla para listar los requerimientos de software antes de que sean evaluados por la herramienta. En un documento de requerimientos de software, cada requerimiento puede tener distintos atributos o información asociada y esta puede variar de un proyecto o de una empresa a otra. Algunos de los atributos que puede tener un requerimiento de software son: identificador único, texto, versión, tipo de requerimiento (*normal* o *derived*), nivel (*high* o *low*), comentarios y otros más. Sin embargo, para realizar el análisis que se plantea en esta investigación, el único atributo indispensable es el texto del requerimiento. El segundo atributo de importancia es un identificador único del requerimiento, ya que es útil para identificar cualquier requerimiento en los reportes que se producen posteriormente durante la evaluación. La estrutura que se utilizó para los requerimientos se muestra en la figura 8.3.

El ejemplo de la figura 8.3 es sencillo a modo de ilustración. Sin embargo, la estructura utilizada también se diseñó de manera simple y lo mínimo que se necesita para hacer el análisis es un identificador único y el texto del requerimiento. Aún cuando este identificador está ausente, la herramienta asigna uno automáticamente y continúa con la evaluación. Luego, un mismo archivo puede tener conjuntos de requerimientos de uno o más sistemas o módulos. En este caso, cada grupo se identifica por medio de la etiqueta *XML* <DOC> para indicar que estos grupos provienen de distintos documentos origen.

Un aporte en esta etapa es que el usuario no tiene que reescribir el documento de requerimientos

```
<SRSGROUP>
 <DOC name="Nombre_del_Documento"
  <REQ id="Id_del_Requerimiento"> texto del requerimiento </REQ>
  <REQ id="Id_del_Requerimiento"> texto del requerimiento </REQ>
 </DOC>
 <DOC>
   ...
 </DOC>
</SRSGROUP>
```

Figura 8.3: Esquema de representación de requerimientos. Este esquema para representación de requerimientos en su forma normalizada es sencillo pero cumple con los requisitos básicos que debe aportar cada requerimiento: texto del requerimiento e identificador único.

en el formato anterior. Para facilitar esta labor se cuenta con una herramienta que extrae los requerimientos y los escribe en el formato adecuado. Esta herramienta está escrita en Perl, es pequeña y fácil de entender, por lo que se puede personalizar para extraer requerimientos completos de cualquier fuente, siempre y cuando estén escritos en formato de texto (y en *XML* idealmente) para poder identificar los patrones que ayudan a delimitar donde comienza y termina un requerimiento.

La herramienta que se usa para extraer los requerimientos del texto original utiliza expresiones regulares para encontrar los requerimientos y luego convertirlos al formato estándar mencionado anteriormente. Estas expresiones regulares se encargan de localizar el inicio y final de cada requerimiento para poder extraerlo y agregarlo a la estructura de la figura 8.3. Tanto el inicio como el final de cada requerimiento puede estar representado de diversas formas en un documento de texto. Por ejemplo, el inicio podría aparecer al principio de una nueva línea o podría aparecer en cualquier otra posición en una línea. Además, el inicio podría estar precedido por algún caracter especial que demarque el inicio como tal o podría estar precedido por cualquier otro caracter. Similares consideraciones se deben hacer para encontrar el final del requerimiento. Afortunadamente las expresiones regulares permiten expresar todas estas situaciones a manera de reglas para lograr encontrar y extraer específicamente el texto que se necesita para los requerimientos.

Tabla 8.1: Ejemplos de expresiones regulares para capturar requerimientos. La 1^{ra} columna muestra una expresón regular, y la 2^{da} columna indica para qué se utilizó dicha expresión regular durante la captura de requerimientos.

Expresión Utilizada	Texto Buscado
^<para>([a-zA-Z0-9]*)<para>	Identificador del requerimiento
^<para>(.*shall.*)<para>	Texto del requerimiento

Por ejemplo, en uno de los documentos que se utilizó en la investigación, las expresiones regulares que se muestran en la tabla 8.1 ayudaron a extraer el texto y el identificador único de cada requerimiento. La primera expresión en la tabla 8.1 permite reconocer un identificador compuesto por una combinación de letras del alfabeto latino entre la *a* y la *z* (mayúsculas y minúsculas) y dígitos entre el *0* y el *9*. Por ejemplo, dicha expresión reconocería un identificador como *OFP1020* pero no reconocería *OFP-1020* (ya que la expresión regular no contempla el guión "-"). En este caso, un vistazo a los documentos de requerimientos origen basta para encontrar el patrón o formato que se utiliza para numerar los requerimientos y así crear una expresión regular para capturar dichos identificadores.

La segunda expresión en la tabla 8.1 permite reconocer un trozo de texto que se encuentra entre las etiquetas de XML <para> y </para> y que contiene la palabra *shall*. En estos casos se busca la palabra *shall* porque se sabe de antemano que los requerimientos utilizan este término de manera consistente. En caso de no incluir el término *shall* en la expresión regular, se cae en el error de tratar como requerimientos otras oraciones que también están bordeadas por las etiquetas <para> y </para> pero que no son requerimientos de software (generalmente son comentarios adicionales al requerimiento).

Es preciso mencionar que la herramienta de extracción de requerimientos no realiza ninguna validación en el conjunto de requerimientos procesado. Por ejemplo, si el documento de requerimientos origen contiene requerimientos duplicados, la herramienta los va a procesar de manera independiente y ambos quedarán escritos en el archivo de salida en formato normalizado. Es decir, en esta investigación se toma como supuesto que el documento de requerimientos origen es correcto.

La siguiente sección describe el proceso de validación de la herramienta.

8.1.3 Validación

Antes de continuar con el resto del prototipo fue necesario validar que la herramienta de extracción de requerimientos realizara la tarea de forma aceptable. Para ello se identificaron varios escenarios que pueden ocurrir en un ambiente real y se crearon pruebas para validar el funcionamiento de la herramienta ante estos escenarios. Algunos de los escenarios utilizados durante las pruebas se muestran en la tabla 8.2.

Tabla 8.2: Escenarios de prueba aplicados a la herramienta de extracción de requerimientos. La 1^{ra} columna indica el nombre del sistema; la 2^{da} columna indica el aporte en cuanto a número de requerimientos y la 3^{ra} columna es una descripción breve del sistema.

Escenario	**Resultado**
Un único documento de requerimientos origen que no tiene ningún requerimiento.	Satisface
Un único documento de requerimientos origen que tiene sólo un requerimiento.	Satisface
Un único documento de requerimientos origen que tiene muchos (500+) requerimientos.	Satisface
Más de un documento de requerimientos origen que no tienen ningún requerimiento.	Satisface
Más de un documento de requerimientos origen que tienen sólo un requerimiento entre todos.	Satisface
Más de un documento de requerimientos origen que tienen sólo un requerimiento cada uno.	Satisface
Más de un documento de requerimientos origen que tienen muchos (500+) requerimientos.	Satisface
Documento de requerimientos origen que incluye requerimientos sin un identificador único.	Satisface
Documento de requerimientos origen que incluye requerimientos con el mismo identificador.	Satisface

Las pruebas para estos escenarios se aplicaron de manera iterativa, corrigiendo los errores detectados durante cada iteración. Durante cada iteración los resultados se revisaron de forma manual en algunos casos y en otros casos se escribieron *scripts* en el lenguaje AWK[4] para revisar los resultados con una herramienta independiente. El nivel de confianza en la herramienta se consideró aceptable cuando las pruebas no arrojaron más errores mientras se procesaban documentos de requerimientos con las características anteriormente mencionadas.

[4] `http://www.gnu.org/software/gawk/`

Una vez que se tiene una herramienta y un esquema para normalizar los requerimientos que van a ser analizados, se continúa, en la sección 8.2, con el proceso de diseño y construcción del prototipo que realiza la evaluación de los requerimientos.

8.2 Análisis de Elementos

La etapa de análisis de requerimientos consiste en la aplicación práctica de los elementos seleccionados en el capítulo 7. Para lograr esta tarea, el prototipo que se construye toma cada uno de los requerimientos normalizados y, utilizando diferentes técnicas, evalúa el estado de los requerimientos respecto de los elementos que se deben examinar (ver tabla 7.10).

La figura 8.4 muestra cómo a la fase de normalización le siguen una serie de etapas de análisis léxico y sintáctico antes de producir los resultados finales. Durante estas etapas de análisis se utilizan distintos recursos y técnicas para examinar cada uno de los elementos. La sección 8.2.2 presentará más detalles sobre estas técnicas.

En la sección 8.2.1 se explica la metodología utilizada para la fase de diseño e implementación del prototipo de software que realiza la evaluación de los requerimientos. Luego, en la sección 8.2.2 se presentan los detalles sobre la implementación de dicho prototipo, y en la sección 8.2.3 se describe el proceso que se utilizó para validar el funcionamiento del prototipo.

8.2.1 Metodología

El primer paso en la metodología es analizar los elementos contra los que se deben evaluar los requerimientos (ver tabla 7.10) para identificar las técnicas y recursos que se necesitan para llevar a cabo la evaluación de manera efectiva y eficiente.

Una vez identificadas las técnicas y recursos con las que se pueden evaluar los requerimientos, se deben definir otros aspectos técnicos en relación con la arquitectura e implementación del prototipo. Por ejemplo, en esta etapa se definieron aspectos como la plataforma y el lenguaje de programación a utilizar de manera que estos sean compatibles con las técnicas y recursos previamente identificados.

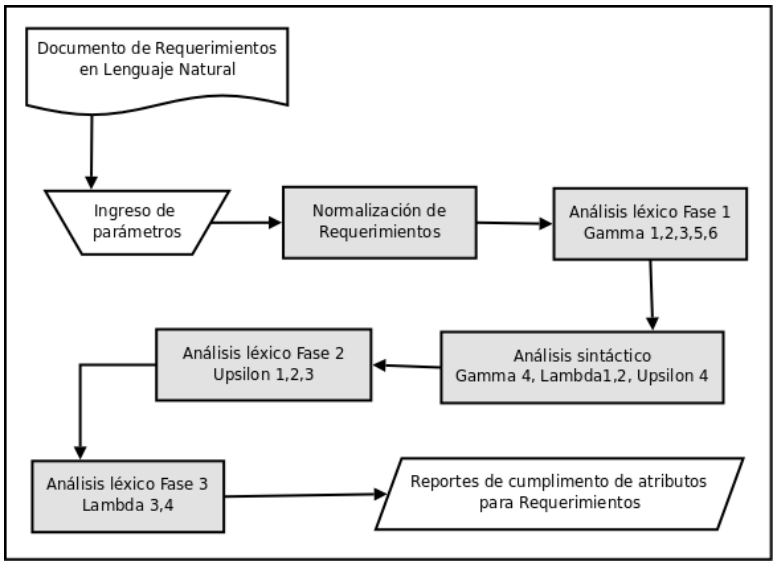

Figura 8.4: Diagrama de flujo del análisis de elementos en el prototipo. Este diagrama es una extensión de la figura 8.1 ya que muestra más detalle para las fases de análisis de los requerimientos a nivel del prototipo. Las fases de análisis aparecen sombreadas en el diagrama y los números junto a los grupos Gamma, Lambda y Upsilon corresponden a los elementos específicos que se analizan durante cada fase del proceso (ver Apéndice G).

El paso siguiente es diseñar el modelo de objetos o arquitectura de software a un nivel conceptual. En esta fase de diseño, se debe tomar en cuenta, aunque de modo general, aspectos de usabilidad y escalabilidad del prototipo. Es claro que ninguno de estos dos aspectos suele ser prioritario a nivel de un prototipo; sin embargo, todo esfuerzo que se haga en este respecto favorece el futuro traslado del prototipo a una herramienta de producción.

Cuando se han definido los principales aspectos del diseño del prototipo, el siguiente paso es definir el ambiente de pruebas funcionales así como los datos que se pueden utilizar durante las pruebas. Este punto es importante pues ayuda a determinar cuándo se puede considerar que la herramienta posee un nivel de funcionalidad aceptable como prototipo.

La última etapa en la metodología consiste en implementar el prototipo que se ha diseñado y posteriormente evaluar la funcionalidad del mismo aplicando las pruebas que se han definido. La sección 8.2.2 describe los resultados de dicha etapa.

8.2.2 Implementación del Prototipo

La implementación del prototipo se va a presentar en tres partes principales. La sección 8.2.2.1 describe las técnicas y recursos que fue necesario utilizar para la construcción del prototipo. Luego, en la sección 8.2.2.2 se presenta el diseño conceptual que se planteó para la arquitectura a nivel de software. Finalmente, las secciones 8.2.2.3 y 8.2.2.4 describen los resultados de la implementación del prototipo; se incluyen aspectos de usabilidad y se explican los reportes o salidas que se producen luego de evaluar un grupo de requerimientos.

8.2.2.1 Técnicas y Recursos Utilizados

Como se muestra en la figura 8.4, durante la evaluación de los requerimientos se aplican principalmente técnicas de análisis léxico y sintáctico. Además, se utilizan también dos recursos lingüísticos de gran valor académico: los diccionarios de WordNet[24] y VerbNet[13].

El análisis léxico consiste en buscar ciertas palabras claves o ciertas estructuras conocidas en los requerimientos y se realiza utilizando expresiones regulares. Esta técnica es de aplicación sencilla y en esta investigación resultó de mucha utilidad, pues permite analizar varios de los elementos, *p. ej.,* búsqueda de valores de tolerancia (para frecuencia, tiempo y voltaje), verbos generales, requerimientos negativos, requerimientos infinitos, uso de *only*, adverbios vagos y otros. La figura 8.4 muestra el análisis léxico dividido en tres fases. Esto es consistente con la implementación y se realizó de esa manera para poder establecer ciertas prioridades que son necesarias dependiendo de la estructura de los requerimientos que se analizan.

Para apoyar el proceso de análisis sintáctico se utilizó un parser puesto a disposición por Eugine Charniak y Brown University [23]. Este parser ha sido utilizado en diversas aplicaciones, tiene un buen desempeño y además se integra de manera directa con los demás recursos utilizados en el prototipo. Esta herramienta está escrita en C y en esta investigación se utilizó un wrapper para Perl puesto a disposición por el grupo *Computational Linguistics And Information Retrieval* (CLAIR) de University of Michigan [5].

Aparte del análisis léxico y sintáctico, durante la etapa de evaluación de elementos es ne-

cesario hacer consultas a diccionarios para determinar la categoría sintáctica de algunos de los términos que se analizan. Estas consultas se logran resolver utilizando una interfaz para el diccionario de WordNet[24]. De manera similar, algunas de estas consultas se deben hacer específicamente para verbos, en cuyo caso se requiere consultar el diccionario VerbNet[13], para el cual también hay disponible una interfaz de consultas. Todos estos recursos se encuentran disponibles en el portal de CPAN[6].

Como se ha anticipado, una decisión técnica importante fue la escogencia de Perl[5] como lenguaje de programación principal para desarrollar el prototipo, aunque algunas tareas intermedias y pruebas se implementaron en Awk. Como lenguaje, Perl tiene algunas características que permiten un avance relativamente rápido durante la elaboración de un prototipo, por ejemplo:

• Es un lenguaje de *scripting* interpretado lo que facilita el desarrollo de prototipos.

• No es un lenguaje fuertemente tipado por lo que permite trabajar con estructuras de datos sin tener que monitorear conversiones de tipos con mucha rigurosidad. Sin embargo, se debe prestar atención a este aspecto pues en ciertos contextos puede inducir a errores.

• Tiene tipos de datos primitivos muy útiles como *hash* y *array* con los cuales se pueden construir otros tipos de datos más complejos de forma sencilla.

• La gama de estructuras de control de flujo es amplia y la sintaxis del lenguaje es flexible.

• Perl permite programación orientada a objetos.

• Existen amplios repositorios de bibliotecas que se pueden utilizar sin mucho esfuerzo para concentrarse en tareas propias del problema que se investiga.

• El motor de expresiones regulares de Perl está entre los más sofisticados en su categoría.

• Es un lenguaje e intérprete de código abierto y existe mucha documentación para los usuarios.

Resumiendo, las principales técnicas que se deben utilizar para la evaluación de los requerimientos son: análisis léxico, análisis sintáctico y consultas a diccionarios. Como se describió anteriormente, es posible integrar cada una de esas técnicas en una herramienta común (en este caso escrita en Perl) para facilitar la tarea de evaluación de requerimientos.

La siguiente sección describe la arquitectura de software utilizada en el prototipo.

[5] `http://www.perl.org/`

8.2.2.2 Arquitectura de Software

Con la intención de construir un prototipo que se pueda extender con un esfuerzo moderado, se planteó un diseño modular tal y como se aprecia en el diagrama de clases de la figura 8.5. Este diagrama utiliza la nomenclatura de UML[6] y muestra el prototipo de manera general. No se pretende profundizar en detalles como atributos y miembros en cada uno de los módulos (o clases). Si el lector desea conocer estos aspectos, la información está disponible a solicitud.

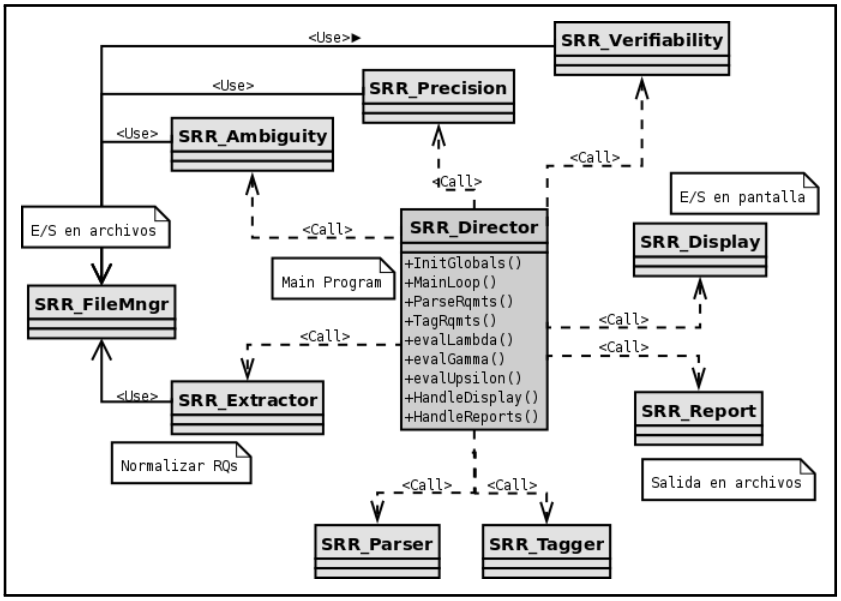

Figura 8.5: Diagrama de clases para el prototipo. Los rectángulos con fondo oscuro representan clases o módulos que tienen una funcionalidad específica. Las etiquetas con fondo claro (blanco) son comentarios que se agregan al diagrama con la intención de indicar o aclarar la funcionalidad que realiza un determinado módulo (aquel que está más cerca de la etiqueta). Algunos de los módulos están unidos mediante líneas. Estas indican que hay una relación entre cada uno de los dos módulos unidos por una línea. Las líneas sólidas, con la marca <use>, representan una relación en la que un módulo usa los servicios de otro módulo proveedor (en este caso el proveedor es SRR_FILEMNGR). Las líneas parciales, con la marca <call>, se usan para indicar que un módulo hace un llamado (*call*) a un proceso que se realiza en otro módulo. En este diagrama se puede notar como el módulo principal –SRR_DIRECTOR– hace llamados a procesos en muchos otros módulos.

El diagrama de la figura 8.5 muestra un módulo principal en el centro SRR_DIRECTOR. Este tiene el control principal del programa y utiliza las funcionalidades de otros módulos que le

[6] *Unified Modeling Language* `http://www.uml.org`

sirven de soporte. Los módulos SRR_AMBIGUITY, SRR_PRECISION y SRR_VERIFIABILITY controlan el proceso principal que revisa aspectos de ambigüedad, precisión y verificabilidad propiamente. Los módulos SRR_PARSER y SRR_TAGGER controlan los procesos de parsing y tagging. El módulo SRR_EXTRACTOR se encarga de leer los requerimientos de entrada desde un archivo XML y finalmente, los módulos restantes se encargan de otras tareas misceláneas como despliegue en pantalla y manejo de archivos.

El diseño modular presentado en la figura 8.5 favorece el mantenimiento y crecimiento del prototipo cuando se requiera pasar a una versión de producción. En la siguiente sección se describe el aspecto gráfico del prototipo.

8.2.2.3 Presentación del Prototipo

En su primera versión, la herramienta es relativamente sencilla. Por ejemplo, no es completamente interactiva aún y la interfaz para el usuario es textual y no gráfica. Sin embargo, internamente se ha construido de manera modular, lo que permite extender sus capacidades con un esfuerzo moderado. Se provee una funcionalidad básica para leer un conjunto de requerimientos, realizar los análisis pertinentes y luego le permite al usuario examinar los resultados mediante una serie de reportes que se explican en la sección 8.2.2.4. La figura 8.6 muestra una captura de pantalla para la pantalla de ayuda del prototipo.

El comando típico para invocar la aplicación es el siguiente:

```
$ perl SRR_Director.pl -v -x XML_FILE
```

El comando anterior ejecuta el programa principal en modo verboso (-v) y extrae los requerimientos del archivo (XML_FILE). Como se omiten ciertos argumentos, como el tipo de análisis a realizar y el tipo de reporte que se desea producir, el prototipo utiliza los valores por omisión (en este caso, realiza los tres tipos de análisis, genera todos los reportes y los guarda en el directorio **./reports**).

Cuando se ejecuta en modo verboso, el prototipo constantemente imprime mensajes en la consola para indicar el estado del proceso. Por el contrario, cuando se ejecuta en modo silencioso,

```
------------------------------------------------------------------------
SRR_Director V 1.0
By A. Berrocal        allan.berrocal@ecci.ucr.ac.cr
To be freely available soon.
------------------------------------------------------------------------
#               Help Page for SRR_Director
#
# Very poor for the moment, please be patient!
# Program Arguments:
# -v               : Enable verbose mode
# -h               : Shows this help page
# -x xml_Rqmts     : Input xml file with requirements
# -d PATH          : Folder to store the reports.
#                    defaults to ./reports.
# -a {all,a,v,p}   : Type of analysis to perform
#                    {all, ambiguity, verifiability, presicion}
# -r {all,r[1-5]}  : Number of report to produce
#                    r1 Executive Report (Percentages).
#                    r2 Executive Report (Scores).
#                    r3 Detailed Report with Scores.
#                    r4 Unsatisfied Elements (Summary).
#                    r5 Unsatisfied Elements (Descriptions).
#                    all produces all five reports.
# -s [NAME]        : Store raw data file for later usage
#                    defaults to SRR_DB
# -l NAME          : Load raw data file to create reports
#------------------------------------------------------------------------
```

Figura 8.6: Pantalla de ayuda principal en el prototipo. Hacia la izquierda de la figura 8.6 se listan las banderas y argumentos que recibe el programa y en la parte de la derecha se explica el propósito de cada bandera o argumento. Una bandera es un indicador para el programa de que debe comportarse de cierta manera ya definida. Por ejemplo, la bandera **-h** le indica al programa que debe desplegar la pantalla de ayuda. Por otro lado, un argumento es una especie de bandera que necesita un dato adicional por parte del usuario. Por ejemplo, el argumento **-x** requiere que el usuario indique el nombre del archivo origen con los requerimientos a evaluar.

los resultados se verán cuando finalice la evaluación de los requerimientos.

La siguiente sección describe los reportes que produce el prototipo como resultado de la evaluación de requerimientos.

8.2.2.4 Salidas del Prototipo

Las salidas que produce el prototipo se basan en las condiciones descritas en la sección 5.1. Para este efecto, el prototipo produce siete reportes en formato .csv[7] con información organizada

[7]El tipo de formato .csv (*comma separated values*) es fundamentalmente un archivo de texto con la facilidad de que se puede importar a una hoja de cálculo como *OpenOffice Spreadsheet* preservando la estructura de filas y columnas contenida en el archivo .csv.

de distintas formas de acuerdo con el tipo de usuarios que utilizará la herramienta.

Para clasificar los siete reportes se han identificado tres tipos de usuarios de la herramienta: ingenieros de calidad, ingenieros de requerimientos e ingenieros de sistemas. En general, a un ingeniero de calidad le interesan los resultados globales en términos de porcentajes de requerimientos que satisfacen o no satisfacen cada uno de los criterios evaluados. Por otra parte, a un ingeniero de requerimientos le interesa conocer con más detalle los motivos por los cuales se considera que un requerimiento no satisface los elementos considerados en la evaluación, pues con base en esto se determinan las acciones correctivas y preventivas pertinentes. Finalmente, a un ingeriero de software que realiza tareas de mantenimiento y actualización del prototipo, le interesa conocer aspectos de desempeño de la herramienta para así proponer mejoras en los algoritmos y demás técnicas utilizadas.

A continuación se describen dos reportes que produce el prototipo y que se consideran de mayor interés para **ingenieros de calidad**:

1) ***Executive report (percentages)*****:** Muestra el número de requerimientos que fue analizado, el porcentaje de requerimientos que no satisface en algún grado cada uno de los atributos (no-ambigüedad, precisión y verificabilidad) y el porcentaje total de requerimientos que no satisface al menos uno de los atributos. Ver ejemplo en la figura 8.7.

Reporte # 1

Title: Executive Report (Percentages).				
# of RQs examined	% of RQs w/errors	% of Ambiguous RQs	% of Non-verifiable RQs	% of Inaccurate RQs
1698	66.30%	42.30%	4.70%	36.90%

Figura 8.7: *Executive report (percentages)*. Muestra de forma resumida los resultados de una revisión de requerimientos. Los valores incluyen el número de requerimientos examinado, el porcentaje total de requerimientos con alguno de los tres errores, y el porcentaje de requerimientos con errores de ambigüedad, verificabilidad y precisión.

2) ***Executive report (scores)*****:** Similar al primero (en valores porcentuales) pero indica el puntaje obtenido para los requerimientos en cada uno de los atributos. También muestra el puntaje global que obtuvo el grupo de requerimientos (fundamentalmente un promedio de los puntajes por atributo). Este reporte es un complemento del reporte 1 ya que, mientras aquel indica el

porcentaje de requerimientos que son imprecisos, este indica qué tan imprecisos son los mismos. Falta agregar acá otros indicadores tales como desviación estándar para tener una evaluación más informativa. Ver ejemplo en la figura 8.8.

Reporte # 2

Title: Executive Report (Scores).					
# of RQs examined	% of RQs w/errors	Overall Score	Ambiguity Score	Verifiability Score	Accuracy Score
1698	66.25%	9.37	9.07	9.89	9.16

Figura 8.8: *Executive report (scores)*. Muestra de forma resumida los resultados de una revisión de requerimientos. Los valores incluyen el número de requerimientos examinado, el porcentaje total de requerimientos con alguno de los tres errores, la calificación general que se le da a los requerimientos en una escala (0-10), y la calificación para cada uno de los tres atributos estudiados.

Los siguientes cuatro reportes que se describen se consideran de mayor interés para los **ingenieros de requerimientos**:

1) *Detailed report with scores*: Este agrega un nivel de detalle respecto del reporte *Executive report (scores)* indicando, para cada requerimiento, el puntaje que obtuvo para cada atributo, y además, muestra el número de los elementos que el requerimiento no satisface en cada atributo. Por comodidad de espacio en los reportes, se utiliza un número para referirse a cada uno de los elementos en vez de su nombre (en el reporte *Miscellaneous* se indica el número correspondiente a cada elemento). Ver ejemplo en la figura 8.9.

Reporte # 3

Title: Detailed Report with Scores.				
Overall Score: 9.37				
RQ uid	Property	Score	Elements	Overall Score
1	Gamma	7.69	4	
1	Upsilon	7.33	1	
1	Lambda	10	0	
1				8.34
2	Gamma	7.69	4	
2	Upsilon	10	0	
2	Lambda	7.39	4	
2				8.36

Figura 8.9: *Detailed report with scores*. Se muestra la calificación en cada uno de los tres atributos y se listan los elementos que el requerimiento no satisface en cada atributo.

2) *Unsatisfied elements (summary)*: Este es mucho más específico que el reporte *Detailed report with scores* y se puede utilizar principalmente para identificar (mediante fórmulas en la hoja de cálculo) los elementos que los requerimientos no satisfacen en cada atributo. Se puede determinar cuáles y cuántos requerimientos satisfacen o no satisfacen simultaneamente más de un atributo. Ver ejemplo en la figura 8.10.

Reporte #4

Title: Unsatisfied Elements (Summary).			
RQ uid	**Accuracy**	**Verifiabilty**	**Ambiguity**
1	4	1	0
2	4	0	4
3	6 4	0	0
4	6	0	0
5	6	0	0

Figura 8.10: *Unsatisfied elements (summary)*. Este reporte lista los elementos que cada requerimiento no satisface.

3) *Unsatisfied elements (descriptions)*: Es más de caracter cualitativo que cuantitativo. Se listan todos los requerimientos junto con cada uno de los elementos que estos no satisfacen. Se presenta una descripción corta indicando cuál es la razón por la cual la herramienta determinó que el requerimiento no satisface ese elemento. La intención de este reporte es que un experto humano pueda analizar cada caso y determinar si efectivamente existe un error. Ver ejemplo en la figura 8.11.

Reporte #5

Title: Unsatisfied Elements (Descriptions).		
RQ uid	**Element**	**Description**
1	Gamma 4	Check that an explicit condition exists for the main action.
1	Upsilon 1	Reword main action of the requirement as a possitive action.
2	Gamma 4	Check that an explicit condition exists for the main action.
2	Lambda 4	Possibly using ambiguous constructs such as {<and/or> <any> <because> <not limitted to>}.
3	Gamma 6	Replace verbs that express general or non-concrete actions.
3	Gamma 4	Check that an explicit condition exists for the main action.

Figura 8.11: *Unsatisfied elements (descriptions)*. Brinda una explicación de porqué los requerimientos no cumplen con cada uno de los elementos en la lista.

4) *Unsatisfied elements (w/ requirements)*: Esencialmente una variante del reporte *Unsatisfied elements (summary)*. Sin embargo, este contiene dos datos más que el reporte *Unsatisfied elements (summary)*, los cuales facilitan el trabajo de lectura a una persona: el identificador del requerimiento y el texto en prosa. Ver ejemplo en la figura 8.12.

Reporte #6

Title: Unsatisfied Elements (w/ Rqmts).					
RQ uid	Accuracy	Verifiability	Ambiguity	RQ Tag	RQ Text
1	4	1	0	10390	Weather and Terrain shall not be displayed together (overlaid)
2	4	0	4	10400	The same function (i.e. RANGE) shall be on the same key in diferent menus for any given installation
3	6 4	0	0	ABP000010	The ABP CSCI shall interface to the four serial channels as defined in the Application Processor CCA Design Specification for the SuperSet Family.

Figura 8.12: *Unsatisfied elements (w/ requirements).*

El último reporte que produce la herramienta (*miscellaneous*) se considera de interés para los **ingenieros de software**, ya que tiene datos sobre el desempeño del prototipo en cada ejecución. Además, a manera informativa este reporte muestra la asociación entre el número que se utiliza en el prototipo para referirse a cada elemento y su nombre corto tal y como aparece en las tablas 7.3, 7.2 y 7.4.

El reporte ***Miscellaneous*** muestra datos generales como el número de requerimientos revisados, el tiempo que tardó el análisis y el tiempo promedio en cada requerimiento. Ver ejemplo en el Apéndice G.

Habiendo presentado los detalles sobre el diseño e implementación del prototipo, en la siguiente sección se describen algunas de las pruebas realizadas para verificar el comportamiento del prototipo.

8.2.3 Validación del Prototipo

Durante el desarrollo del prototipo se realizaron pruebas parciales constantemente con el objetivo de validar que la herramienta hace el análisis de cada uno de los elementos de la manera esperada según las características de cada elemento. Además, dado que cada uno de

los elementos se debe examinar de forma independiente, también se realizaron pruebas para verificar que al agregar nuevas funcionalidades a la herramienta, no se afectaban los elementos previamente contemplados.

Para garantizar la validez y efectividad de las pruebas, se utilizaron muestras de requerimientos de software reales en la mayoría de las pruebas. Aunque no se realizaron pruebas para cada uno, durante el desarrollo se contó con 1.698 requerimientos disponibles (aproximadamente 80 % de los 2.100 requerimientos que corresponden a los sistemas descritos en la tabla 5.1). En algunos casos, sin embargo, sí fue necesario crear grupos de requerimientos con errores muy específicos para poder validar el comportamiento de la herramienta ante escenarios poco comunes. Los resultados de las pruebas en esta etapa se revisaron de forma manual.

Durante las pruebas en esta etapa de validación también se decidió sobrecargar el sistema para tener un indicador de desempeño en cuanto a consumo de tiempo y recursos. Una de las pruebas se realizó con un total de 1698 requerimientos en una máquina con las características descritas en la tabla 8.3. En dicha prueba, el prototipo tardó 56,7 minutos para hacer la evaluación. Es decir, en promedio tardó 2 segundos para cada requerimiento, lo que es un resultado muy alentador si se toma en cuenta que en la práctica, a un profesional le toma aproximadamente 2 minutos[8] analizar un requerimiento detalladamente. Suponiendo que esta persona puede trabajar sin descansos, le tomaría casi 56 horas realizar la misma tarea. Se puede también suponer que el profesional encontrará patrones en los requerimientos, con lo que su tiempo de análisis se reduce. Considerando un factor de 50 % en la reducción, el profesional aún tardaría 28 horas para conseguir el resultado.

[8]Este dato se ha obtenido basado en experiencias previas durante el desarrollo de proyectos en la empresa Avionyx. S.A., en la cual colabora el autor de la investigación. El dato no se corroboró en esta investigación pues no se solicitó a los evaluadores. Como se detalla en la sección 5.1, la eficiencia en tiempo no fue uno de los parámetros a medir en los experimentos.

Tabla 8.3: Descripción de la máquina de desarrollo y pruebas. La 1^{ra} columna describe la máquina que se utilizó para desarrollo y pruebas y la 2^{da} columna muestra la versión del software utilizado para el mismo propósito.

Características de Hardware	**Características de Software**
• Lenovo SL-300 • Intel Core2 Duo T5670 @1.8GHz • 3 GB RAM	• Ubuntu Linux 8.10 / Kernel 2.6.27-11-generic • Perl V5.10.0 / VerbNet 2.3 / WordNet 3.0 • Charniak parser (08-2005) • clairlib-core 1.05 / MEAD 3.11

En términos estrictos, no se tiene información exacta en cuanto a requisitos mínimos que se deben considerar para utilizar el prototipo. Sin embargo, la tabla 8.3 describe las características del computador en el cual se desarrolló la aplicación y en el que también se realizaron las pruebas preliminares obteniendo resultados aceptables en cuanto a consumo de tiempo y recursos.

9

Comparación del Prototipo con los Profesionales

La comparación que se realiza entre el prototipo y los profesionales pretende medir el cumplimiento de los objetivos específicos 3 y 4 de la investigación: *diseñar e implementar un prototipo de software que evalúe los requerimientos de software con los elementos seleccionados en el capítulo 7* y *evaluar la efectividad del prototipo comparando sus resultados con los resultados producidos por profesionales* respectivamente.

La primera parte de la metodología de evaluación es seleccionar los requerimientos que se utilizan como datos de prueba. Para contar con una selección apropiada de datos de prueba, los requerimientos se han tomado de sistemas reales, de modo que los datos de prueba sean representativos de la realidad en el dominio estudiado. Esta primera parte se describe en la sección 9.1. La segunda parte es crear un instrumento apropiado para realizar la evaluación con un grupo de profesionales. En la sección 9.2 se explican los detalles sobre esta tarea. Finalmente, la tercera parte consiste en la aplicación de los experimentos; es decir, la evaluación en sí de los requerimientos por parte del prototipo y por parte de los profesionales que colaboraron en esta etapa. Esta tercera parte se describen en la sección 9.3.

En las siguientes secciones se describe la metodología utilizada en cada una de las tres fases. Además, cada sección describe también los resultados obtenidos para cada fase. La sección 9.1 presenta la metodología así como los resultados para la fase de selección de datos de prueba.

9.1 Selección de Datos de Prueba

Los detalles metodológicos para el proceso de selección de datos de prueba se presentan en la sección 9.1.1 y los resultados de dicho proceso se describen en la sección 9.1.2.

9.1.1 Metodología

El primer paso para diseñar los experimentos es seleccionar un conjunto de requerimientos de software de prueba. En esta investigación, los requerimientos utilizados para evaluación corresponden a requerimientos de software auténticos para los sistemas descritos en la tabla 5.1. Estos mismos sistemas se describen nuevamente en la tabla 9.1 por comodidad.

Tabla 9.1: Sistemas fuentes para obtención de datos de prueba. La 1^{ra} columna indica el nombre del sistema; la 2^{da} columna indica el aporte en cuanto a número de requerimientos y la 3^{ra} columna es una descripción breve del sistema.

Sistema	Aporte	Descripción
Multi functional display	65 %	Aplicación de software que controla el comportamiento de un dispositivo de entrada de datos y salida de video conocido como MFD.
On-board health monitoring unit	20 %	Módulo de monitoreo general que constantemente revisa la integridad del sistema durante un vuelo.
On-board maintenance unit	15 %	Módulo de monitorero de errores que se utiliza para documentar ciertas fallas ocurridas durante un vuelo.

La idea de utilizar los sistemas en la tabla 9.1 como fuente principal es garantizar que los requerimientos son reales, lo cual agrega validez a los resultados que se obtengan en los experimentos. Estos requerimientos se tomaron de versiones iniciales de cada uno de los documentos de requerimientos, por cuanto las versiones finales de dichos documentos ya tenían correcciones para las deficiencias detectadas durante el desarrollo de dichos proyectos. Para aclarar, aunque se realizaron mejoras en los requerimientos durante cada uno de esos proyectos, dichas mejoras se implementaron en versiones posteriores a las que se utilizaron en esta investigación (no se contó con la información de las nuevas versiones para esta investigación).

Debido a restricciones de propiedad intelectual, los requerimientos originales pasaron por un proceso de saneamiento en el cual se reemplazaron nombres de conceptos específicos así como descripciones de funcionalidades específicas de software propietario por nombres o descripciones ficticias. No obstante, los cambios realizados no afectan negativamente los requerimientos para hacerlos imprecisos, ambiguos o no verificables cuando no lo eran. En resumen, se reemplazaron únicamente etiquetas o nombres de conceptos propietarios que comprometen la confidencialidad de estos sistemas.

La tarea principal en esta fase es determinar cuáles son los aspectos que se desea que sean evaluados tanto por parte de la herramienta como por parte del grupo de profesionales que colaboran en los experimentos. Una vez conocidos los aspectos que se pretenden evaluar, se utiliza el conjunto de requerimientos disponible para elaborar los experimentos o pruebas específicas, de manera que los datos se ajusten a las características de cada experimento.

La sección 9.1.2 presenta los resultados para el proceso de selección de los datos que se utilizan en los experimentos.

9.1.2 Resultados y Análisis

En total se disponía de 2.100 requerimientos, de los cuales se reservaron inicialmente 402 (aproximadamente 20 % del total) para efectos de control y no se tomaron en cuenta durante las pruebas de desarrollo del prototipo. Como se explicó en la sección 8.2.3, el otro 80 % de los requerimientos estuvo disponible durante las pruebas que se realizaban para validar el funcionamiento de la herramienta. Como se detalla más adelante, debido al tipo de experimentos que se han planteado en la investigación, se utilizarán requerimientos de ambos grupos para medir distintos aspectos.

Como una decisión de diseño, se planteó el experimento en cuatro grupos de 20 requerimientos cada uno, para un total de 80 requerimientos. Uno de los cuatro grupos de 20 requerimientos se tomó de entre la muestra del 20 % que había sido reservada al principio. Los siguientes tres grupos se tomaron del 80 % restante de requerimientos. La idea de formar cuatro grupos obedece principalmente a dos razones: (a) clasificar los requerimientos cuando se sabe de antemano el tipo de error que tienen (para asegurar una muestra balanceada); y (b) reducir la carga de trabajo para los evaluadores humanos al particionar el instrumento en cuatro grupos que se pueden revisar en distintos momentos. La carga de trabajo puede parecer irrelevante; sin embargo, para los profesionales que colaboran en esta etapa, participar en los experimentos es una carga pues el trabajo que requiere analizar 80 requerimientos es considerable[1]. Para minimizar el efecto de fatiga, se diseñó el experimento en grupos pequeños que reduzcan la sensación de tamaño, pues

[1] En este experimento cada participante utilizó en promedio cuatro horas para evaluar los 80 requerimientos.

ya no es un total de 80 requerimientos a evaluar sino cuatro grupos de 20.

Se puede argumentar que solo 80 requerimientos en los experimentos son insuficientes pues el número es apenas 4 % de la cantidad disponible (2100). Sin embargo, la selección de requerimientos se hizo de manera tal que la muestra seleccionada contuviera una amplia cantidad de requerimientos con deficiencias que se debían identificar. Para lograr estas características, 60 de los 80 requerimientos fueron seleccionados tomando en cuenta la evaluación que la herramienta previamente había realizado sobre dichos requerimientos. Utilizando esta información se lograron escoger mayoritariamente requerimientos con errores (55 de 60), dejando solo un pequeño grupo de requerimientos sin errores (5 de 60) en la muestra seleccionada para los experimentos.

De forma general, los dos escenarios que se pretende cubrir en los experimentos son:
• *Muestra desconocida*: Se escoge un grupo de requerimientos aleatoriamente, luego la herramienta y los profesionales examinan los requerimientos y determinan si estos incumplen con los atributos de *no-ambigüedad*, *precisión* y *verificabilidad*. Luego se comparan los resultados de la herramienta con los resultados de los profesionales y se analizan las diferencias para determinar la efectividad del prototipo para clasificar errores de forma similar a los profesionales utilizando una muestra desconocida.
• *Muestras variables*: La herramienta examina todos los requerimientos disponibles. Luego, de acuerdo con los resultados de la evaluación, se hacen tres grupos de requerimientos. Los profesionales evalúan estos grupos y se comparan sus resultados con los resultados de la herramienta para determinar la similitud en la clasificación del prototipo y los profesionales en cada uno de los tres escenarios.

Los cuatro grupos que se formaron tienen el propósito de satisfacer los dos escenarios ya descritos (muestra desconocida y muestras variables). Así, de los 80 requerimientos a analizar, 20 se tomaron aleatoriamente para formar el primer grupo, como se explica más adelante, y se tomaron de la reserva de entre 402 requerimientos que habían sido apartados para evaluación. Los restantes 60 requerimientos (para formar tres grupos) se escogieron mediante una decisión de diseño conociendo la evaluación que el prototipo producía para esa muestra. De este modo, los tres grupos son controlados y cumplen el propósito del segundo escenario descrito (muestras variables).

Los requerimientos evaluados por el prototipo se clasificaron en las siguientes categorías: (1) *sin problemas*, (2) *ambiguo solamente*, (3) *impreciso solamente*, (4) *no-verificable solamente*, (5) *ambiguo e impreciso*, (6) *ambiguo y no-verificable*, y (7) *impreciso y no-verificable*.

Una vez con los requerimientos clasificados, se elaboraron cuatro grupos con las siguientes características:

Grupo 1:
Constitución: El 100 % de los ejemplos son desconocidos para la herramienta, es decir, no han sido evaluados previamente.
Propósito: Medir la efectividad del prototipo para clasificar errores de manera similar a como lo hacen los profesionales.

Grupo 2:
Constitución: 25 % de los requerimientos son *ambiguos solamente*, 25 % son *imprecisos solamente*, 25 % son *no-verificables solamente* y 25 % *no tienen problemas*.
Propósito: Medir la similitud en la clasificación de errores que hace el prototipo y la clasificación de los profesionales en una muestra diversificada.

Grupo 3:
Constitución: 50 % son *ambiguos e imprecisos*, 25 % son *ambiguos y no-verificables* y 25 % son *imprecisos y no-verificables*.
Propósito: Medir el efecto en el número de coincidencias en la clasificación cuando se sabe (según el prototipo) que los requerimientos tienen más de un error simultaneamente.

Grupo 4:
Constitución: 25 % son *ambiguos solamente*, 35 % son *no-verificables solamente* y 40 % son *imprecisos solamente*.
Propósito: Medir el efecto en el número de coincidencias en la clasificación cuando se sabe (según el prototipo) que todos los requerimientos tienen únicamente un error.

La tabla 9.2 muestra la distribución de requerimientos por tipo de error en los cuatro grupos.

La sección 9.2 describe la metodología y los resultados del proceso de evaluación.

Tabla 9.2: Distribución de tipos de requerimientos por grupo. La 1^{ra} columna indica los posibles errores. La 2^{da} columna indica el número de requerimientos disponibles (con el error de la columna 1). Las columnas 3 a 6 indican el número de requerimientos con errores del tipo indicado que pertenecen a cada grupo.

Tipo de error	**Total disponible**	**Grupo**			
		1	**2**	**3**	**4**
Desconocido	20	20	0	0	0
Sin problemas	5	0	5	0	0
Ambiguos solamente	10	0	5	0	5
Imprecisos solamente	13	0	5	0	8
No-verificables solamente	12	0	5	0	7
Ambiguos e imprecisos	10	0	0	10	0
Ambiguos y no-verificables	5	0	0	5	0
Imprecisos y no-verificables	5	0	0	5	0
Totales	**80**	**20**	**20**	**20**	**20**

9.2 Evaluación con Profesionales

Los detalles metodológicos para el proceso de evaluación con profesionales se presentan en la sección 9.2.1 y los resultados de dicho proceso se describen en la sección 9.2.2.

9.2.1 Metodología

Una vez seleccionados los grupos de requerimientos a evaluar, el siguiente paso fue elaborar un instrumento adecuado para realizar la evaluación. Para elaborar dicho instrumento se pretende utilizar un formato *amigable* con los evaluadores, ya que la tarea de revisar requerimientos es tediosa y se quiere evitar, dentro de lo posible, que los evaluadores cometan errores durante la evaluación debido a fatiga.

Algunos de los aspectos que se tomaron en consideración para hacer el instrumento *amigable* son:

• **Tamaño**: Un instrumento que se considere corto crea una mejor disposición en el evaluador que un instrumento sumamente extenso.

• **Instrucciones**: El instrumento debe contener instrucciones claras que le indiquen al evaluador correctamente lo que se le solicita.

• **Tipo de respuestas**: Se refiere a la facilidad que se le brinda al evaluador para responder. Por

ejemplo, es más tedioso responder preguntas de forma argumentativa que responder preguntas de selección múltiple.

• **Alternativas de respuesta**: Consiste en asegurar que el instrumento no tenga vicios que impidan al evaluador responder las preguntas con suficiente libertad. Por ejemplo, es bueno permitir una opción de respuesta donde el evaluador indique que no sabe cómo responder a la pregunta.

• **Formato**: Se refiere a la forma de aplicar el instrumento, *p. ej.,* de forma electrónica, de forma impresa, por teléfono, entre otras.

Con estos criterios en mente se elaboró un instrumento para realizar la evaluación y los detalles se muestran en la sección 9.2.2.

9.2.2 Resultados y Análisis

En el instrumento que se preparó para la evaluación con los profesionales, cada grupo de requerimientos está separado en una hoja de tipo carta (8.5 x 11 pulgadas), de manera que el tamaño del cuestionario no influya negativamente en los evaluadores. Además, la mecánica es tal que en una misma línea el evaluador marca su respuesta y agrega comentarios adicionales cuando lo considere necesario. El instrumento se le entrega a los evaluadores de forma impresa y de forma electrónica para que puedan utilizar la forma que más les convenga.

El instrumento a utilizar se muestra en el Apéndice H. Cada instrumento tiene los cuatro grupos de requerimientos claramente separados. En cada grupo se incluye la lista de 20 requerimientos y justo al lado se presentan seis opciones de respuesta que el evaluador puede marcar según los resultados de su evaluación para cada requerimiento. Las opciones son: (1) *Es impreciso*, (2) *Es ambiguo*, (3) *Es no-verificable*, (4) *Tiene otro problema*, (5) *No tiene ningún problema* y (6) *No tengo criterio para decidir*.

Como se puede notar en el Apéndice H, la tarea del evaluador consiste en leer cuidadosamente cada uno de los 20 requerimientos en cada grupo y luego marcar con una equis “x” en una o más de las opciones disponibles. Luego, debido a que en la evaluación hay un cierto grado de

subjetividad[2], es importante conocer el razonamiento que llevó a cada evaluador a seleccionar una respuesta específica. Para este efecto, se ha pedido a los profesionales explicar sus criterios cuando sea posible.

La tercera parte de la evaluación consiste en aplicar los experimentos. Estos detalles se explican en la siguiente sección.

9.3 Aplicación de la Evaluación

Los detalles metodológicos para el proceso de evaluación se presentan en la sección 9.3.1 y los resultados de dicho proceso se describen en la sección 9.3.2.

9.3.1 Metodología

El primer paso para aplicar los experimentos es seleccionar a los participantes. En este caso se contó con la ayuda de tres ingenieros de software experimentados en el área de verificación de software para sistemas desarrollados bajo el estándar DO-178B, y en particular en el tema de análisis de requerimientos. Los tres profesionales que participaron en este experimento son: Ing. Daniel Pérez, Ing. Esteban Sánchez e Ing. Fernando Bogarín. El currículum vitae de estas personas se ha adjuntado en el Apéndice I.

Ninguno de los tres participantes pertenece al equipo que colaboró con las evaluaciones descritas en los apartados 7.3 y 7.4. Sin embargo, los tres participantes seleccionados para este experimento poseen un nivel de entrenamiento y experiencia igual al de aquellos en el primer grupo. Además, ni el autor ni los colegas que han colaborado en esta fase de experimentos formaron parte del equipo encargado de la revisión de los requerimientos de software en los proyectos que sirvieron como fuente de requerimientos.

Los tres evaluadores usaron copias idénticas del instrumento y se les solicitó completarlos sin colaborar entre ellos. Esta restricción de no colaborar entre evaluadores se implementó para

[2]La subjetividad se debe a que los evaluadores pueden tener criterios ligeramente distintos para realizar su trabajo pues no utilizan un método formal o al menos compartido entre ellos.

lograr total independencia en los resultados. Sin embargo, en un proceso de análisis de requerimientos real, la colaboración entre personas o equipos es más bien una práctica deseable. El prototipo también evalúa exactamente el mismo grupo de requerimientos que evalúan los profesionales.

La sección 9.3.2 presenta los resultados de los experimentos.

9.3.2 Resultados y Análisis

Luego de aplicar el instrumento con los tres evaluadores y con la herramienta, se tabularon los datos para poder realizar una análisis de los resultados finales. La tabla completa con los datos de los evaluadores y la herramienta se encuentra en el apéndice J.

Se elaboraron dos tablas que resumen los resultados obtenidos mediante los experimentos. La tabla 9.3 muestra los porcentajes de coincidencias entre los resultados de los evaluadores y los resultados producidos por la herramienta para cada uno de los grupos.

En la tabla 9.3, el *Porcentaje de coincidencias con mayoría de evaluadores* se calcula como la suma de todos los casos donde la evaluación de la herramienta coincide con la evaluación hecha por dos o tres de los evaluadores, dividida entre el total de requerimientos evaluados en el grupo (20). De forma similar, el *Porcentaje de coincidencias con al menos un evaluador* se calcula como la suma de todos los casos donde la evaluación de la herramienta coincide con la evaluación hecha por al menos uno de los evaluadores, dividida entre el total de requerimientos evaluados en el grupo (20).

La primera línea de la tabla 9.3 (*Porcentaje de coincidencias con mayoría de evaluadores*) indica el porcentaje de veces en que los tres evaluadores coincidieron con la evaluación que hizo la herramienta. Nótese como este valor cambia de manera significativa para cada uno de los grupos, pero en general, es un porcentaje bajo y alcanza un promedio de $37{,}5\,\%$ entre los cuatro grupos evaluados con una desviación estándar de $32{,}27$. Hay que tomar en cuenta que este valor promedio y desviación estándar corresponden a una muestra pequeña de apenas cuatro grupos; por ende, lo mejor es observar principalmente el resultado en cada grupo.

Tabla 9.3: Resumen de coincidencias entre los evaluadores y la herramienta. La 1^{ra} columna describe el resultado que se presenta en cada fila. Las columnas 2 a 5 muestran los valores resultantes para cada uno de los grupos en el instrumento. La 6^{ta} columna muestra el promedio general de resultados para todos los grupos y la 7^{ma} muestra la desviación estándar. El símbolo "n/a" indica que el dato no aplica para esa entrada de la tabla y por esa razón no se muestra ningún número.

Similitud entre herramienta y profesionales	**Grupo**				**Media**	**Desviación estándar**
	1	**2**	**3**	**4**		
Porcentaje de coincidencias con mayoría de evaluadores	25 %	75 %	0 %	50 %	37,5 %	32,27
Porcentaje de coincidencias con al menos un evaluador	55 %	95 %	10 %	75 %	58,75 %	36,37
Porcentaje de coincidencias para dos criterios con mayoría de evaluadores	n/a	n/a	0 %	n/a	n/a	n/a
Porcentaje de coincidencias para al menos un criterio con mayoría de evaluadores	n/a	n/a	40 %	n/a	n/a	n/a
Porcentaje de coincidencias para al menos un criterio con al menos un evaluador	n/a	n/a	100 %	n/a	n/a	n/a

La segunda línea de la tabla 9.3 (*Porcentaje de coincidencias con al menos un evaluador*) indica el porcentaje de veces en que al menos un evaluador coincidió con la evaluación que hizo la herramienta. En este caso se puede notar una pequeña mejora en los grupos 1, 2 y 4 de $30\,\%$, $20\,\%$ y $25\,\%$ respectivamente; sin embargo, el promedio de coincidencias en esta medición aún se mantiene relativamente bajo en $58\,\%$ para los cuatro grupos, con una desviación estándar también significativa de 36.

Los resultados de las últimas tres líneas de la tabla 9.3 son válidos únicamente para el grupo número tres ya que solo en ese grupo se tenían requerimientos con al menos dos errores conocidos. Como se puede notar en la tabla, no hubo ni un solo caso en que los tres evaluadores coincidieran con la herramienta para los mismos dos defectos que la herramienta encontró en los requerimientos (*Porcentaje de coincidencias para dos criterios con mayoría de evaluadores*). Luego, en un $40\,\%$ de los casos sí hubo coincidencia entre los tres evaluadores y la herramienta pero solo para uno de los dos errores detectados (*Porcentaje de coincidencias para al menos un criterio con mayoría de evaluadores*). Finalmente, un resultado más alentador, y no del todo despreciable para el tercer grupo, es el caso donde al menos un evaluador coincide con la herramienta para al menos uno de los dos errores que la herramienta detectó; en este caso el valor es de $100\,\%$ (*Porcentaje de coincidencias para al menos un criterio con al menos un evaluador*).

Uno de los resultados que arrojan los datos para el grupo número tres en la tabla 9.3 es que los evaluadores humanos aparentemente no se empeñan tanto en examinar los requerimientos de manera exhaustiva. Es decir, parece ser el caso de que una vez que se ha identificado un error en un requerimiento se da por terminado el análisis dejando otros criterios sin examinar (o quizá realizando un análisis superficial). Nótese como en la segunda línea (*Porcentaje de coincidencias con al menos un evaluador*), el valor más bajo corresponde al tercer grupo ($10\,\%$).

Antes de continuar discutiendo los resultados de la tabla 9.3, la tabla 9.4 presenta los resultados para los porcentajes de coincidencias entre los evaluadores a la hora de identificar y clasificar errores.

Tabla 9.4: Resumen de coincidencias entre profesionales. La 1^{ra} columna describe el resultado que se presenta en cada fila. Las columnas 2 a 5 muestran los valores resultantes para cada uno de los grupos en el instrumento. La 6^{ta} columna muestra el promedio general de resultados para todos los grupos y la 7^{ma} muestra la desviación estándar.

Similitud entre profesionales	**Grupo**				**Promedio**	**Desviación estándar**
	1	**2**	**3**	**4**		
Porcentaje de coincidencias entre tres evaluadores	$15\,\%$	$20\,\%$	$5\,\%$	$15\,\%$	$13{,}75\,\%$	$6{,}29$
Porcentaje de coincidencias entre al menos dos evaluadores	$50\,\%$	$85\,\%$	$40\,\%$	$75\,\%$	$62{,}5\,\%$	$21{,}02$

Como se puede notar en la tabla 9.4, el porcentaje de coincidencias entre los tres evaluadores es muy bajo en los cuatro grupos, con una media de $13{,}75\,\%$ y una desviación estándar de $6{,}29$. El porcentaje de coincidencias entre tres evaluadores se calcula como la suma de los casos en que los tres profesionales coincidieron con su evaluación, dividia entre el total de requerimientos evaluados en un grupo (20). Nótese que el porcentaje de coincidencias entre evaluadores es independiente de la evaluación de la herramienta. Es decir, los evaluadores podrían coincidir en una evaluación distinta de la que hace la herramienta para algún requerimiento. Al calcular el porcentaje de coincidencias entre al menos dos evaluadores, el resultado que se obtiene mejora muy poco para tener una media de $62{,}5\,\%$ y una desviación estándar de $21{,}02$.

Uno de los resultados que evidencia la tabla 9.4 es que no hay uniformidad entre los profesionales en cuanto a los conceptos de *ambigüedad*, *precisión* y *verificabilidad*. Esta falta de uniformidad entre evaluadores es un dato importante de subrayar ya que puede producir resultados desfavorables para evaluaciones en producción (aún sin que exista ninguna herramienta de por

medio). Por ejemplo, puede suceder que los distintos tipos de errores (errores de *ambigüedad*, *precisión* o *verificabilidad*) tengan asociado un costo; es decir, el costo de resolver los distintos problemas es conocido previamente y además es distinto para cada tipo de error. Entonces, la incapacidad de un evaluador para distinguir correctamente los errores puede generar pérdidas económicas potencialmente importantes.

Para complementar los resultados de la tabla 9.4, se decidió evaluar si hubo al menos uno de los evaluadores que, de forma individual, sí coincidiera con los resultados de la herramienta en los experimentos realizados. Este dato es importante para saber si al menos uno de los evaluadores mostró criterios similares a la herramienta en su evaluación. La tabla 9.5 muestra los resultados de esta medición y se puede notar que tampoco hay una tendencia marcada para ninguno de los evaluadores en relación con los resultados de la herramienta. Como se nota en la tabla, el **evaluador B** es quien más coincide con la herramienta, en general, pero el porcentaje sigue siendo bajo, $46{,}25\,\%$ con una desviación estándar significativa de $29{,}26$ puntos.

Tabla 9.5: Comparación de coincidencias profesionales-herramienta de forma individual. La 1^{ra} columna muestra el nombre del evaluador. Las columnas 2 a 5 indican el porcentaje de coincidencias entre la herramienta y cada evaluador en el grupo indicado en cada columna. Las últimas dos columnas muestran el promedio y la desviación estándar para cada evaluador en su respectivo grupo.

Evaluador	Grupo 1	Grupo 2	Grupo 3	Grupo 4	Promedio	Desv. Estd.
A	50 %	65 %	0 %	55 %	42,5	29,01
B	55 %	80 %	10 %	40 %	46,25	29,26
C	20 %	40 %	0 %	30 %	22,5	17,08

Otro de los resultados importantes que se pudo notar al realizar estos experimentos es que en general las personas tienden a ser poco meticulosas para realizar evaluaciones de requerimientos de software escritos en lenguaje natural. En particular, esta limitante es muy notoria cuando se trata de requerimientos de software para sistemas de aviónica, ya que las condiciones que deben satisfacer los requerimientos para cumplir con los objetivos de calidad suelen ser más rigurosas que las condiciones que se usan para otros tipos de software (muchos de los cuales ni siquiera utilizan un estándar como guía de desarrollo). Como se puede inferir de las tablas que describen los elementos en el capítulo 7, tablas 7.2, 7.3, 7.4 y su versión extendida en el apéndice B, existen aspectos que no son intuitivos y que se deben considerar para lograr una

descripción aceptable en los requerimientos de software. Algunos aspectos incluso invitan a utilizar documentación externa como apoyo. Tal es el caso de algunos criterios para determinar *ambigüedad* y *no verificabilidad* principalmente cuando se trata de utilización de ciertos verbos que, de forma sutil, pueden producir diferentes significados.

Retomando los datos de la tabla 9.3, los porcentajes de coincidencia entre la herramienta y los evaluadores son en general muy bajos. Además, dado que no hay una marcada coincidencia entre los profesionales (como lo muestra la tabla 9.4), no es posible hacer comparaciones entre ellos y la herramienta en téminos de eficacia. Sin embargo, dado que los resultados presentados en la tabla 9.4 describen el comportamiento de los evaluadores al revisar el mismo grupo de requerimientos que revisó la herramienta, se hace un llamado especial para tomar conciencia sobre la necesidad de abrir una discusión para mejorar las técnicas que se utilizan en la industria de aviónica para revisar requerimienos de software, al menos para los criterios de *ambigüedad*, *precisión* y *verificabilidad*.

Sin embargo, es preciso notar como la herramienta construida en esta investigación es de gran utilidad para identificar deficiencias muy específicas en los requerimientos de software, dado que las personas (al menos las que han participado en estos experimentos) muestran comportamientos erráticos al tratar de identificar ese tipo de deficiencias.

10
Análisis General de Resultados

Como se mencionó en la sección 6, los capítulos entre 7 y 9 describen cada una de las etapas en que se ha desarrollado la investigación. Cada uno de dichos capítulos incluye a su vez un apartado donde se discuten los resultados parciales que se han obtenido durante cada etapa. A manera de resumen, este capítulo retoma algunos de los aspectos más importantes que se observaron durante el desarrollo de la investigación.

Un primer aporte de la investigación fue definir una notación especial que sirve para expresar las diferentes situaciones que resultan de la evaluación de los requerimientos (ver sección 7.1). Esta notación o nomenclatura es sencilla de utilizar y permite describir de forma clara y consistente ciertos resultados en torno al tema de análisis de requerimientos. Un aspecto importante es que la nomenclatura propuesta permite expresar de forma numérica el estado de salud de cada requerimiento (y también de todo el conjunto de requerimientos) lo que permite obtener una evaluación cuantitativa y no solo cualitativa, como suele ser la costumbre (calificando los requerimientos subjetivamente como buenos, regulares o malos).

El segundo aporte significativo en la investigación fue la definición de la lista de elementos que se deben tomar en cuenta para determinar tres tipos de errores en los requerimientos: *ambigüedad*, *imprecisión* y *no-verificabilidad*. La validez de esta lista de elementos se comprobó mediante consultas a profesionales como se explica en la sección 7.3. Una lista completa de los elementos sugeridos se incluye en el apéndice B. Aunque no todos los elementos se utilizaron para la construcción del prototipo en la investigación (ver sección 7.2), el aporte consiste en el hecho de que la lista de elementos se puede aplicar en un proceso de revisión de requerimientos incluso sin contar con la ayuda de una herramienta, ya que cada elemento describe de manera específica los detalles que se deben observar para determinar si un requerimiento cum-

ple o no con los criterios de calidad evaluados. La definición de la lista de elementos y la forma en que los mismos ayudan a identificar problemas de *ambigüedad*, *precisión* y *verificabilidad* satisface los objetivos específicos 1 y 2 de la investigación.

Como se describe en el capítulo 8, para satisfacer el objetivo específico 3, se construyó un prototipo de software que revisa un conjunto de requerimientos para determinar en qué medida estos cumplen con los elementos identificados en la sección 7.2. Este prototipo se puede extender de forma sencilla para mejorar la efectividad de las búsquedas o para ampliar el ámbito de búsqueda cuando se requiera revisar otros aspectos en los requerimientos. Este prototipo es una herramienta de bajo costo que utiliza recursos disponibles de forma gratuita en Internet (ver sección 8.2.2.1) así como otras técnicas de procesamiento de texto descritas en la sección 4.4.

Una de las ventajas del prototipo es que reduce significativamente el tiempo de análisis de los requerimientos en comparación con el tiempo que necesita una persona (ver sección 8.2.3). Además, como resultado de la evaluación, el prototipo genera una serie de reportes que resumen resultados de interés para distintos actores en un proyecto de software, específicamente, ingenieros de calidad, ingenieros de requerimientos e ingenieros de sistemas (ver sección 8.2.2.4). Aunque se advierte que la herramienta es aún un prototipo, los resultados que produce pueden ayudar a rectificar –tempranamente– deficiencias en un proyecto de software, en particular cuando este se encuentra en la etapa de definición de requerimientos. Como resultado, las correcciones que se realizan a los requerimientos son potenciales errores que ahora tendrán menos posiblidades de aparecer en el software final.

Durante la etapa de evaluación del prototipo (capítulo 9), se observaron varios resultados de interés general. En esta etapa se contó con la ayuda de profesionales que tienen experiencia en el análisis de requerimientos de software para la industria de aviónica utilizando el estándar DO-178B. Uno de los resultados importantes es que hay un gran déficit de profesionales con entrenamiento adecuado para realizar tareas de revisión de requerimientos de manera detallada. Uno de los errores más notorios en este tema es la creencia errónea de que, en general, una persona con un dominio aceptable del idioma[1] y con conocimientos ingenieriles (en asuntos de

[1]Este es un criterio muy subjetivo, entiéndase por *dominio aceptable* la capacidad de leer, escribir y discutir con notable solvencia sobre diversos asuntos utilizando un idioma, específicamente Inglés, y en particular cuando este no es el idioma natal de una persona.

software, electrónica, o computacionales en general), posee las destrezas suficientes para identificar deficiencias en un requerimiento de software (para sistemas empotrados). Este resultado es lamentable pero muy posiblemente válido aún en otras áreas en la industria del software.

Es evidente que la forma de resolver una deficiencia como la descrita anteriormente no es sencilla. Para comenzar, este problema no es de índole puramente económico, pues hay aspectos que tienen que ver más directamente con las estrategias sugeridas por las distintas metodologías de desarrollo. Por ejemplo, una posible mejora en la etapa de revisión de requerimientos de software (cuando estos están escritos en lenguaje natural) sería contar con la participación de personal experto en el idioma (*p. ej.,* lingüistas) quienes, posiblemente, puedan identificar ciertas deficiencias de forma más efectiva que otra persona con menos conocimiento del idioma. Sin embargo, esta persona evidentemente también debe tener conocimiento suficiente en asuntos técnicos en el área de desarrollo y verificación de software; por lo tanto, el costo de un proyecto se incrementaría al contar con profesionales de este tipo y quizá esa es una de las razones por las cuales no se cuenta con personal tan calificado para realizar esas tareas.

Uno de los resultados en los experimentos descritos en la sección 9.3.2 de esta investigación fue la escasa similitud entre las evaluaciones de tres profesionales experimentados en el análisis de requerimientos en el área de aviónica. En algunos de los casos se notaron deficiencias en el manejo de los conceptos de *ambigüedad*, *precisión* y *verificabilidad*, al punto de que los profesionales lograban identificar errores de igual naturaleza pero los clasificaban en una u otra categoría inconsistentemente. En otros casos se notó que los profesionales lograban identificar un error pero no lo asociaban con ninguna categoría de las que contemplaba el experimento y en cambio indicaban que "el error pertenecía a otra categoría". Finalmente, otro de los resultados observados fue, para aquellos requerimientos que tienen al menos dos errores relativamente evidentes, los profesionales tienden a encontrar uno de los errores (y no ambos) durante las revisiones. Los detalles de estos resultados se describen en la sección 9.3.2. La ventaja de contar con una herramienta que apoye el proceso de revisión en una etapa temprana es que, potencialmente, reduce los costos y tiempos requeridos para el desarrollo de un software.

Las inconsistencias notadas entre los profesionales en esta investigación no se pueden considerar, categóricamente, como una representación de un problema general en la industria; tampo-

co se puede generalizar esta situación a otras industrias de desarrollo de software; sin embargo, los resultados sí se deben tomar como una indicación de una posible deficiencia en los procesos de revisión de requerimientos escritos en lenguaje natural. Además, se debe notar que, pese a las deficiencias encontradas, los procesos de desarrollo y verificación de software en la industria de aviónica se llevan a cabo de manera rigurosa, ya que así lo requiere el estándar. En consecuencia, las deficiencias que una persona pasa por alto posiblemente son detectadas por otros actores en un proceso real de desarrollo, ya que la cantidad de revisiones que se realizan durante el proceso, sumado a la diversidad de personas que participan en dichas revisiones, es muy amplia, lo cual garantiza, en un alto porcentaje, que se logra encontrar la mayoría de errores en algún momento durante el desarrollo.

En general, los resultados obtenidos durante los experimentos ponen en evidencia las dificultades que se habían descrito en el capítulo 1, en relación con la tarea de validar que un conjunto de requerimientos de software cumpla con los lineamientos establecidos por el estándar DO-178B cuando no se cuenta con la ayuda de una herramienta adecuada. Una posible repercusión de estas incongruencias puede afectar negativamente el costo de un proyecto; por ejemplo, suponga que cada uno de los tres tipos de errores estudiados *ambigüedad*, *precisión* y *verificabilidad* tiene un costo asociado. Este costo representa la cantidad de esfuerzo o dinero que se debe invertir en un requerimiento para corregir ese error. Entonces, un posible escenario es que no se logren identificar los errores del todo. En este caso, existe la posibilidad de que los errores escalen hacia las fases siguientes en el ciclo de desarrollo, con el riesgo de que estos afecten el software final. Otro escenario es que se logren identificar errores en los requerimientos, pero estos errores se clasifican en categorías erróneas. El problema de este escenario es que se puede caer en el error de estimar de forma equivocada el esfuerzo o costo necesario para corregir los errores encontrados en los requerimientos. Por ejemplo, si una persona clasifica todos los requerimientos que revisó como *no-verificables*, se estima el costo de las correcciones para dichos requerimientos basado en el costo conocido para errores de *verificabilidad*. Sin embargo, puede ocurrir que en la realidad los errores no fueran de *verificabilidad* sino más bien de *ambigüedad* y entonces el costo podría ser más alto que el costo estimado inicialmente. En este punto ya se genera un conflicto que puede afectar de forma directa los planes del proyecto.

Finalmente, otra de las consecuencias de las diferencias notadas entre las evaluaciones de los profesionales es el hecho de que no se pueden hacer comparaciones significativas entre la herramienta y los profesionales en cuanto a la calidad de los resultados que se producen. Como se muestra en la tabla 9.5, tampoco fue posible encontrar un patrón de similitud representativo entre al menos uno de los profesionales y la herramienta. Pese a esta situación, el objetivo específico 4 se logra cumplir ya que es posible realizar los análisis de los requerimientos y obtener resultados válidos que incluso ayudaron, en este caso, a detectar posibles debilidades en los procesos manuales de revisión de requerimientos.

Una vez resumidos los principales aspectos de la investigación, que además se detallaron en los capítulos del 7 al 9, se continúa en el siguiente capítulo con la presentación de las conclusiones y la descripción de algunas ideas sugeridas como trabajo futuro.

11
Conclusiones y Trabajo Futuro

Este capítulo presenta las conclusiones de la investigación en la sección 11.1. Además, con base en las conclusiones y otros resultados observados, se presenta en la sección 11.2 una lista de posibles trabajos que se pueden realizar en el futuro para ampliar el alcance de la investigación.

11.1 Conclusiones

Una de las principales razones que originó la investigación es el supuesto de que contar con una especificación de software que cumpla con ciertos criterios de calidad, aumenta las posibilidades de que el software que se produce como producto final, cumpla con las espectativas de los usuarios, al tiempo que se minimiza el número de posibles errores. Los requerimientos de software son una forma muy común de producir especificaciones de software, al menos en la industria de aviónica, y como se mencionó en la sección 4.2, el estándar DO-178B indica los objetivos de calidad que deben satisfacer dichos requerimientos dentro del ciclo de desarrollo del software.

Tal y como se había descrito en la sección 4.2, el estándar de desarrollo de software DO-178B lista una serie de criterios de calidad que los requerimientos de software deben satisfacer; sin embargo, el estándar no aporta información detallada sobre una posible metodología que se pueda utilizar para verificar el cumplimiento de dichos criterios.

Aunque existe investigación orientada a producir requerimientos de software de calidad (ver capítulo 2), el hecho de tener que realizar ciertas tareas de forma manual hace de esta una tarea difícil. Parte de la dificultad se debe a que el lenguaje natural permite muchas libertades en

cuanto a las estructuras que se utilizan para transmitir una idea; en consecuencia, estas libertades pueden introducir problemas graves cuando no se utiliza el lenguaje con conocimento de causa. Adicionalmente, esta misma dificultad hace que se deba contar con personas capacitadas en el tema de análisis de requerimientos, y estas personas suelen ser escasas. Sumado a lo anterior, las listas de requerimientos son por lo general extensas, lo cual implica que se debe invertir mucho tiempo en los procesos de revisión manual de requerimientos.

Una herramienta que facilite el proceso de revisión de requerimientos es de gran utilidad durante el ciclo de desarrollo del software. Sin embargo, estas herramientas no son abundantes debido, entre otras razones, a la complejidad de producirlas. En esta investigación se planteó la posibilidad de construir un prototipo capaz de evaluar un conjunto de requerimientos para determinar en qué grado estos cumplen con algunos criterios de calidad definidos en la investigación: *no-ambigüedad*, *precisión* y *verificabilidad* (ver capítulo 3).

La primera conclusión en la investigación es que es posible establecer guías que ayuden a los actores de una revisión de requerimientos a profundizar en detalles que comúnmente se subestiman, y que pueden afectar negativamente la calidad o el significado completo de un requerimiento de software. Para facilitar su utilización y maximizar su efectividad, dichas guías deben ser breves y muy puntuales, de manera que se puedan incluir fácilmente en una lista de revisión (en inglés *check list*) que las personas puedan utilizar rápidamente.

La segunda conclusión es que es posible desarrollar herramientas que ayudan a automatizar parcialmente el proceso de revisión de requerimientos. En particular, se ha aportado evidencia para mostrar que se puede automatizar parcialmente el proceso de revisión de requerimientos para determinar si estos satisfacen los criterios de *no-ambigüedad*, *precisión* y *verificabilidad* (dada una definición particular de dichos términos).

Con la utilización de herramientas como la que se propone en esta investigación, se logran otros resultados importantes en los procesos de revisión. Por ejemplo, se reduce el tiempo requerido para revisar grandes cantidades de requerimientos (ver sección 8.2.3) en comparación con un proceso manual. Adicionalmente, dado que los resultados de la herramienta se pueden reproducir una y otra vez, se reducen las posibilidades de inconsistencias entre las personas que

participan en las revisiones mediante procesos manuales.

Para apoyar la idea de incorporar herramientas a los procesos de revisión de requerimientos de software, se hace énfasis en el hecho de que existen suficientes recursos disponibles de forma gratuita en Internet que facilitan la construcción de herramientas como la que se planteó en la investigación. Entre los recursos disponibles se subrayan diccionarios especializados en ciertos aspectos del lenguaje natural, lenguajes de programación que facilitan las tareas de manipulación de textos, módulos de software reutilizables que realizan tareas complejas como *parsing* y *tagging* que se pueden incorporar fácilmente dentro de otra herramienta, y algunos otros recursos más. Como resultado, se debe notar que, sin tener que construir absolutamente todos los componentes de una aplicación, es posible desarrollar herramientas pequeñas que aportan resultados valiosos a los procesos de revisión de requerimientos.

La tercera conclusión de la investigación es que podrían existir algunas debilidades en las metodologías de revisión de requerimientos de software aplicadas en la industria. Por ejemplo, durante los experimentos descritos en la sección 9.3.2, se notó que existen deficiencias conceptuales en la interpretación de los términos *ambigüedad*, *precisión* y *verificabilidad*. Por otra parte, se notó que las personas tienden a revisar los requerimientos de forma superficial –en algunos casos– con lo que ciertos errores sutiles no se logran detectar eficientemente. Como consecuencia de estas debilidades, se producen a menudo inconsistencias entre las evaluaciones que realizan distintas personas sobre un mismo grupo de requerimientos, lo cual afecta el nivel de confianza que se puede tener en dichos resultados.

Una consecuencia importante respecto de las debilidades en los procesos de revisión de requerimientos de software, en particular para los conceptos de *ambigüedad*, *precisión* y *verificabilidad*, es que se puede incurrir en pérdidas económicas durante el ciclo de desarrollo de un software cuando no se tiene una clara conceptualización de dichos términos (ver análisis en la sección 10).

De manera similar, en relación con los problemas de conceptualización, es evidente que, si no se cuenta con un mecanismo efectivo para revisar requerimientos de software, no es posible verificar que el software producido implementa correctamente los requerimientos. La razón

es que, si los requerimientos tienen errores de *ambigüedad*, *precisión* o *verificabilidad* que no fueron detectados (y corregidos) durante la fase de requerimientos, entonces no se puede saber con certeza cuál fue la interpretación que se hizo de dichos requerimientos durante la fase de codificación de software. Una metodología de diseño de casos de prueba rigurosa generalmente logra descubrir este tipo de errores en los requerimientos ya que es necesario analizar los requerimientos detalladamente para crear los casos de prueba apropiados.

El objetivo final en un proyecto de desarrollo de software (bajo el estándar DO-178B) es lograr que el producto final del proceso, el software, implemente correctamente la especificación de requerimientos junto con la arquitectura propuesta durante la etapa de diseño. Lograr este objetivo presupone que el software resultante satisface las necesidades y expectativas de los usuarios finales del producto. Los resultados obtenidos en esta investigación sugieren que el uso de herramientas que apoyen el proceso de revisión de requerimientos ayuda a detectar, de manera eficiente, ciertos problemas en etapas tempranas del ciclo de desarrollo. En particular, una herramienta como la que se propone en esta investigación ayuda a detectar problemas de *ambigüedad*, *precisión* y *verificabilidad* en los requerimientos de software escritos en lenguaje natural para sistemas empotrados en la industria de aviónica.

11.2 Trabajo Futuro

A continuación se listan algunas ideas que pueden ser implementadas en investigaciones futuras para tratar de mejorar la calidad de los resultados que brinda el prototipo.

La primera mejora consiste en afinar el mecanismo de búsqueda para el atributo de *ambigüedad*. Se recomienda utilizar información contextual para derminar si un verbo es o no es *ambiguo*. En esta primera implementación, el criterio de *ambigüedad* para un verbo se define a partir de consultas a los diccionarios WordNet y Verbnet. Sin embargo, hay ciertos verbos que, pese a ser considerados como *ambiguos* en VerbNet, tienen un significado generalmente claro en el ámbito del desarrollo de software, por ejemplo: *set*, *shutdown/turn off*, *send/receive*, *clear* entre otros. En una implementación futura sería útil para los usuarios configurar listas de excepciones de manera que los verbos incluidos en esas listas no se consideren ambiguos.

La segunda mejora es permitir la posibilidad de seleccionar los criterios a aplicar dependiendo del tipo de requerimientos que se desea revisar. La razón de esta mejora es que a diferencia de los requerimientos de bajo nivel, en donde es necesario contar con especificaciones completas y muy detalladas, en los requerimientos de alto nivel a menudo se permite omitir ciertos detalles. En un sistema que se desarrolla bajo el estándar DO-178B, generalmente se cuenta con requerimientos de alto nivel y de bajo nivel; además, los requerimientos de bajo nivel siempre están vinculados con su respectivo requerimiento de alto nivel. Por lo tanto, a menudo las "deficiencias" de un requerimiento de alto nivel se corrigen o se aclaran en los requerimientos de bajo nivel (nivel de diseño) respectivos. Dada esta situación, puede ser valioso que los usuarios de una herramienta escojan el nivel de rigurosidad con que desean que se realice la evaluación.

La tercera mejora es permitir al prototipo procesar requerimientos que se escriben con un formato menos usual. Por ejemplo, hay requerimientos de software que utilizan listas con varios ítems, donde cada uno de los ítems representa ya sea, una de las acciones que debe realizar el software o bien, las condiciones que se deben cumplir para que el software realice una determinada acción. Es importante poder distinguir correctamente entre ambos casos y poder procesar adecuadamente la lista de ítems pues, en su estado actual, la herramienta no cubre estos aspectos.

La cuarta mejora es muy puntual pero que podría ser un aporte en determinadas circunstancias. En la sección 7.1.2, específicamente para la ecuación 7.2, se mencionó que durante la evaluación de un requerimiento, el prototipo da por satisfechos aquellos elementos que no aplican para dicho requerimiento. Este enfoque se adoptó por conveniencia dada su claridad y sencillez. Sin embargo, se recomienda explorar otras alternativas donde la evaluación refleje, de otro modo, situaciones en las que ciertos elementos no son aplicables a un requerimiento.

Una quinta sugerencia consiste en repetir los experimentos aplicados en esta investigación, pero permitiendo que los profesionales que evalúan los requerimientos puedan trabajar en forma grupal (y no individual como se realizó en esta ocasión). El enfoque de trabajo grupal podría incidir en algunos de los resultados de los evaluadores. Uno de los aspectos a medir sería en porcentaje de coincidencias entre profesionales ya que las técnicas de trabajo en grupo podrían reducir las diferencias de percepción entre los expertos a la hora de realizar sus evaluaciones.

Finalmente, una última sugerencia es mejorar el esquema de asignación de puntajes a elementos descrito en la sección 7.4.1. La debilidad que se podría mejorar es el hecho de que los evaluadores utilizan una escala de *1* a *3* para calificar o valorar cada elemento. Sin embargo, como se describe en ese mismo apartado, el prototipo utiliza una escala de *0* a *10* para su evaluación. Dos posibles aspectos negativos son: no es fácil agregar, remover o modificar elementos bajo este esquema y, hay una dificultad para el usuario final al tener que mapear calificaciones entre una escala y otra cuando eso sea necesario. Proponer un esquema de asignación de puntajes que comparta la misma escala de valores con el esquema de evaluación de requerimientos podría generar algunas ventajas para efectos de análisis por parte de los usuarios.

Apéndices

A
Etiquetas en Penn Treebank

La siguiente tabla[1] muestra la lista de etiquetas llamada *Penn Treebank* descrita en [22]. Esta es una lista simplificada de la versión utilizada para etiquetar el corpus *American Brown*.

Tabla A.1: Elementos sintácticos utilizados en *Penn Treebank*.

Etiqueta	**Descripción**	**Ejemplos**
CC	Coordinating conjunction	here, there, now
CD	Cardinal number	3, fifteen
DT	Determiner	this, each, another
EX	Existential there	there
FW	Foreign word	
IN	Preposition or subordinating conjunction	although, when
JJ	Adjective	happy, bad
JJR	Adjective, comparative	happier, worse
JJS	Adjective, superlative	happiest, worst
LS	List item marker	A, A., First
MD	Modal	can, could, 'll
NN	Noun, singular or mass	aircraft, woman, sugar
NNS	Noun, plural	women, books
Continúa en la página siguiente		

[1]Para permitir mayor claridad en la tabla, se optó por una transcripción directa de los mismos manteniendo los nombres en el idioma original.

Tabla A.1 – continúa de la página anterior

Etiqueta	**Descripción**	**Ejemplos**
NNP	Proper noun, singular	London, Michael
NNPS	Proper noun, plural	Australians, Methodists
PDT	Predeterminer	quite, half
POS	Possessive ending	’s, ’
PRP	Personal pronoun	you, we
PRP$	Possessive pronoun	yours, mine
RB	Adverb	often, particularly
RBR	Adverb, comparative	faster
RBS	Adverb, superlative	fastest
RP	Particle	up, off ,out
SYM	Symbol	[fj]*
TO	to	
UH	Interjection	oh, yes, mmm
VB	Verb, base form	take, live
VBD	Verb, past tense	took, lived
VBG	Verb, gerund or present participle	taking, living
VBN	Verb, past participle	taked, lived
VBP	Verb, non-3rd person singular present	am, ’m
VBZ	Verb, 3rd person singular present	takes, lives
WDT	Wh-determiner	which, whaever
WP	Wh-pronoun	whom
WP$	Possessive wh-pronoun	whose

B

Detalle de Elementos

A continuación se presenta una descripción detallada de cada uno de los elementos para detectar problemas de *precisión*, *ambigüedad* y *verificabilidad* respectivamente. En cada caso se ofrece una descipción, justificación, ejemplos en positivo y negativo, fuente y otros comentarios cuando sea necesario. Las partes subrayadas en los ejemplos ilustran aspectos específicos de cada elemento.

Precisión

P1) Elemento: Argumentos del verbo
Descripción: Un RS debe incluir todos los argumentos para verbos transitivos que necesitan un argumento adicional al complemento directo.
Justificación: Al no indicar los argumentos del verbo, el lector del requerimiento debe hacer una interpretación para completar su significado. Generalmente el significado correcto es de conocimiento común pero sigue siendo un error de precisión no especificarlo con claridad.
Ej. incorrecto: "*The system shall increment the fault counter when a new fault is logged.*"
Ej. correcto: "*The system shall increment the fault counter by one when a new fault is logged.*"
Comentarios: Otros ejemplos son: "*The system shall set (a bit), to on / to off*". "*The system shall decrement (a variable) by X*"
Fuente: Propuesto en esta investigación.

Los siguientes tres elementos tratan sobre la necesidad de especificar valores de tolerancia. Cuando se utilizan unidades físicas, es importante saber que estas generalmente no presentan un problema para el programador ya que, a nivel de software, estas se manejan mediante variables discretas. Sin embargo, para efectos de verificación de software, a menudo de utilizan

instrumentos que capturan señales físicas y realizan las mediciones y es en este momento donde el verificador requiere contar con la especificación para el margen de tolerancia.

P2) Elemento: Tolerancia para frecuencia
Descripción: Un requerimiento debe indicar un valor de tolerancia para acciones que se realizan de manera periódica con mediciones físicas.
Justificación: El verificador no puede crear una prueba correcta si no conoce cuál es la tolerancia del sistema. Si utiliza exactamente los valores descritos en el requerimiento, corre el riesgo de que el instrumento muestre una medición ligeramente distinta al valor esperado y por ende debe fallar la prueba.
Ej. incorrecto: "*The system shall send ARINC label 251 every 50 ms.*"
Ej. correcto: "*The system shall send ARINC label 251 every 50 ms (+/- 5ms).*"
Comentarios: Otros indicadores de periodicidad son: "*once per (time unit) / at a rate of (x Hz) / at a (x Hz)*" .
Fuente: Propuesto en esta investigación.

P3) Elemento: Tolerancia para tiempo
Descripción: Un requerimiento debe indicar un valor de tolerancia para mediciones de tiempo.
Justificación: El verificador no puede crear una prueba correcta si no conoce cuál es la tolerancia del sistema. Si utiliza exactamente los valores descritos en el requerimiento, corre el riesgo de que el instrumento muestre una medición ligeramente distinta al valor esperado y por ende debe fallar la prueba.
Ej. incorrecto: "*The system shall log the CONN_BROKEN fault if an ACK is not received within 1s after transmitting STX.*"
Ej. correcto: "*The system shall log the CONN_BROKEN fault if an ACK is not received within 1s (+/- 100ms) after transmitting STX.*"
Comentarios: Otros indicadores de tiempo son: "*within / after / for / over / until*"
Fuente: Propuesto en esta investigación.

P4) Elemento: Tolerancia para voltaje
Descripción: Un requerimiento debe indicar un valor de tolerancia para las mediciones de voltaje y corriente eléctrica.

Justificación: El verificador no puede crear una prueba correcta si no conoce cuál es la tolerancia del sistema. Si utiliza exactamente los valores descritos en el requerimiento, corre el riesgo de que el instrumento muestre una medición ligeramente distinta al valor esperado y por ende debe fallar la prueba.
Ej. incorrecto: "*The system shall enter LOW_PWR mode when voltage reading equals 20v.*"
Ej. correcto: "*The system shall enter LOW_PWR mode when voltage reading equals 20v (+/- 0.5v).*"
Comentarios: También se consideran unidades inferiores como milivoltio (mV).
Fuente: Propuesto en esta investigación.

P5) Elemento: Único shall
Descripción: Un requerimiento no debe tener más de un "shall", es decir, no más de una acción principal.
Justificación: Cuando un requerimiento describe varias acciones, se denomina "compuesto". Las pruebas en ese caso deben verificar cada acción individualmente; entonces, cuando una prueba falla, se debe marcar el requerimiento como fallido. Generalmente hay rastreabilidad[1] entre los requerimientos y las pruebas, es decir, cada prueba está asociada a un requerimiento; por ende, cuando una prueba falla, se documenta que "el requerimiento falla". En un requerimiento compuesto se corre el riesgo de que sea solo una de las acciones la que falla.
Ej. incorrecto: "*The system shall display altitude every 2 seconds and shall display temperature every 3 seconds.*"
Ej. correcto: "*The system shall display altitude every 2 seconds (+/- 100ms)*" y por separado "*The system shall display temperature every 3 seconds (+/- 100ms).*"
Comentarios: ninguno
Fuente: Propuesto en otra literatura.

P6) Elemento: Referencias externas
Descripción: Un requerimiento debe evitar referencias externas para describir una acción que puede ser descrita en el requerimiento mismo.

[1]Este concepto, llamado en inglés *traceability*, se utiliza en ingeniería de software para denotar la capacidad de mantener registros actualizados sobre las relaciones o vínculos entre ciertos elementos importantes que forman parte del ciclo de vida del software. Por ejemplo, un mapeo entre los requerimientos de alto nivel y los requerimientos de bajo nivel permite que haya rastreabilidad entre dichos elementos. Lo mismo ocurre cuando existe un mapeo entre los requerimientos de bajo nivel y el código fuente del programa.

Justificación: El principal problema de las referencias externas es que las secciones, tablas o figuras referenciadas no siempre le permiten al lector del requerimiento identificar claramente el dato que necesita para completar el significado del requerimiento. Cuando la referencia es a una sección completa de un documento, se aumenta el ámbito potencial de búsqueda y por ende se aumenta también el margen de error para la persona que debe leer el requerimiento.
Ej. incorrecto: "*The system shall enter INTERACTIVE mode as defined in section 4.2.4.*"
Ej. correcto: "*The system shall enter INTERACTIVE mode by sending ARINC label 260 with the ENQ command.*"
Comentarios: ninguno
Fuente: Propuesto en otra literatura.

P7) Elemento: Condición explícita
Descripción: Un requerimiento debe incluir una condición que indique cuándo se debe realizar la acción principal.

Justificación: La condición es una parte fundamental de un requerimiento ya que sin esta no se puede escribir una prueba para verificar la funcionalidad del sistema.
Ej. incorrecto: "*The system shall clean the DMA shared space.*"
Ej. correcto: "*The system shall clean the DMA shared space when IER0 equals 0x1D5.*"
Comentarios: A veces la condición está implícita: (a) la acción es permanente, como un *watchdog*, (b) se puede derivar claramente del contexto por ejemplo en "*The system shall initialize register IER0 to 0xA5A5.*"
Fuente: Propuesto en otra literatura.

P8) Elemento: Voz activa
Descripción: Un requerimiento debe estar escrito en voz activa y no en voz pasiva.
Justificación: Por definición, los requerimientos de software son acciones que **el software** debe realizar bajo ciertas circunstancias (condiciones). Por lo tanto, el énfasis debe hacerse en el hecho de que **el software** realiza la acción. La voz pasiva no satisface esta condición y pone el énfasis en la acción que se debe realizar dejando en segundo grado al responsable de la acción (*i. e.,* al software).
Ej. incorrecto: "*Validity of all commands shall be verified by the IO Module.*"

Ej. correcto: "*The IO Module shall verify validity of all commands. [when ...]*"
Comentarios: ninguno
Fuente: Propuesto en otra literatura.

P9) Elemento: No determinismo
Descripción: Un requerimiento no debe incluir formas no determinísticas que funcionan como adverbios.
Justificación: El problema de los adverbios no determinísticos es que las personas pueden percibir diferentes valores o significados dada la imprecisión de estos conceptos.
Ej. incorrecto: "*The system shall periodically perform CBITE.*"
Ej. correcto: "*The system shall perform CBITE every 5 minutes (+/- 30s).*"
Comentarios: Otros adverbios no determinísticos usados comúnmente son: *continually* y *regularly*.
Fuente: Propuesto en otra literatura.

P10) Elemento: Verbos generales
Descripción: Un requerimiento no debe utilizar verbos que indican procesos generales que no se pueden asociar directamente con una acción atómica o específica.
Justificación:
Ej. incorrecto: "*The system shall monitor responses from the slave service.*"
Ej. correcto: "*The system shall log responses from the slave service to the SLAVE_ACTIVITY file.*"
Comentarios: Otros verbos que indican procesos generales son *process, compare, check, use, define, interface, support*.
Fuente: Propuesto en esta investigación.

Ambigüedad

A1) Elemento: Agrupaciones explícitas
Descripción: Un requerimiento debe describir las agrupaciones de ANDs y ORs mediante signos de puntuación adecuados o bien utilizando paréntesis explícitos.

Justificación: Las agrupaciones que usan las conjunciones “and” y “or” deben ser precisas porque estas son en el fondo las condiciones de relaciones lógicas que se deben implementar en el sistema. Cuando estas agrupaciones no son explícitas, el programador o el verificador deben “adivinar” las agrupaciones correctas a partir de su conocimiento del sistema y del contexto. Esto es evidentemente indeseable pues es muy propenso a errores.

Ej. incorrecto: “*The system shall enter Normal mode when SDI field on label 227 equals 2 or SSM in label 268 equals 3 and WOW is true or AIR is false.*”

Ej. correcto: “*The system shall enter Normal mode when SDI field on label 227 equals 2 or SSM in label 268 equals 3 and* **,** *WOW is true or AIR is false.*”

Comentarios: Otra forma de presentar el requerimiento claramente es usando varias líneas:

The system shall enter Normal mode when:
- SDI field on label 227 equals 2 or SSM in label 268 equals 3 and
- WOW is true or AIR is false.

Fuente: Propuesto en esta investigación.

A2) Elemento: Verbos ambiguos

Descripción: Un requerimiento no debe utilizar como verbo principal un verbo que tenga múltiples significados en el contexto en que se utiliza.

Justificación: El verbo es por mucho el principal componente de un requerimiento de software. Un verbo que acepte varios significados en un mismo contexto es muy probable que origine errores en algún punto del ciclo de vida del software. En un proyecto de software varias personas tienen contacto con el requerimiento en algún momento, por ejemplo, arquitectos, desarrolladores, verificadores, revisores, ingenieros de calidad, *Designated Engineering Representative* (DERs) y otros; por ende, es muy riesgoso suponer que todos estos actores van a interpretar un verbo ambiguo de la misma forma.

Ej. incorrecto: “*The system shall clean the display before a write operation.*”

Ej. correcto: “*The system shall clear all visible characters from the display before a write operation.*”

Comentarios: Definir una medida para determinar cuando un verbo se considera ambiguo en un determinado contexto no es tarea sencilla. Existen varios enfoques para realizar dicha tarea. En este caso se utilizó información conjunta de los diccionarios VerbNet y WordNet para identificar

verbos ambiguos.

Fuente: Propuesto en otra literatura.

A3) Elemento: Adverbios vagos

Descripción: Un requerimiento no debe utilizar adverbios vagos o adverbios generales para describir la acción principal.

Justificación: El problema principal de los adverbios vagos o generales es que no son medibles de manera consistente.

Ej. incorrecto: "*The system shall allow the operator to adjust volume to an acceptable level.*"

Ej. correcto: "*The system shall allow the operator to adjust volume from 1 to 50.*"

Comentarios: Otros ejemplos son: *about, accurate, adequate, applicable, appropriate, average, better, desirable, easy, efficient, excessive, immediately, insufficient, known, less, low, necessary, normal, quick, reasonable, relevant, safe, secure, similar, sufficient, variable*

Fuente: Propuesto en otra literatura.

A4) Elemento: No determinismo

Descripción: Un requerimiento no debe utilizar estructuras o formas que se saben no determinísticas.

Justificación: El problema de estas estructuras es que, pese a que pueden ser aceptadas en el lenguaje, inyectan ambigüedad al requerimiento pues no se sabe con certeza cuál es la intención real del mismo.

Ej. incorrecto: "*The system shall display altitude and/or temperature at the botton line of the screen depending on the value of SHOW.*"

Ej. correcto: "*The system shall display altitude at the botton line of the screen when SHOW equals X.*", en otro requerimiento "*The system shall display temperature at the botton line of the screen when SHOW equals Y.*" y finalmente, en otro requerimiento "*The system shall display altitude and temperature at the botton line of the screen when SHOW equals Z.*"

Comentarios: Otros ejemplos son: *any, not limited to, because, capable of.*

Fuente: Propuesto en otra literatura.

A5) Elemento: *Scope ambiguity*

Descripción: Un requerimiento debe evitar múltiples interpretaciones debido a que el ámbito

de los cuantificadores no es claro.
Justificación: Este tipo de ambigüedad se da en situaciones donde no queda claramente definida la precedencia de los cuantificadores existencial y universal.
Ej. incorrecto: *The system shall keep a unique log file for all IO interfaces.*
Ej. correcto: *The system shall keep a log file for each IO interface.*
Comentarios: ninguno
Fuente: Propuesto en otra literatura.

Verificabilidad

V1) Elemento: Requerimiento negativo
Descripción: Un requerimiento no debe expresar la acción principal como una acción negativa.
Justificación: Los requerimientos de software tienen el propósito de describir únicamente aquellas acciones que el software debe realizar. Existe una infinidad de acciones que un software no realiza; por ende, cualquier intento por listar estas acciones es inutil pues la cantidad es infinita. Sin embargo, listar aquellas cosas que el software sí debe hacer es siempre una tarea finita.
Ej. incorrecto: *The system shall not enter INTERACTIVE mode when ON_AIR is true.*
Ej. correcto: *The system shall enter INTERACTIVE mode when ON_AIR is false.*
Comentarios: Esta restricción puede parecer algo sutil a primera vista; sin embargo, como se explicó anteriormente, la justificación tiene su fundamento en la definición misma de Requerimiento de Software pues los requerimientos deben describir acciones que el sistema sí realiza.
Fuente: Propuesto en otra literatura.

V2) Elemento: Uso de *only*
Descripción: Un requerimiento debe utilizar el término "*only*" correctamente para evitar restringir erroneamente la acción del verbo principal.
Justificación: Cuando el término *only* precede al verbo en un requerimiento de software, este modifica al verbo de manera categórica y como resultado, la acción que describe ese requerimiento se convierte en la única acción que realiza el software.
Ej. incorrecto: *The system shall only display pressure and altitude while in manual mode.*

Ej. correcto: *The system shall display only pressure and altitude while in manual mode.*
Comentarios: ninguno
Fuente: Propuesto en otra literatura.

V3) Elemento: Requerimiento infinito
Descripción: Un requerimiento no debe utilizar los términos "*always*" y "*never*" para restringir una acción que debe realizar el software.
Justificación: El problema en este caso es que se necesita tiempo infinito para demostrar que el sistema realiza o no realiza una determinada acción debido a las restricciones **siempre** o **nunca**.
Ej. incorrecto: *The system shall never perform CBITE when status equals TAKE_OFF.*
Ej. correcto: *The system shall perform CBITE when status does not equal TAKE_OFF.*
Comentarios: ninguno
Fuente: Propuesto en en otra literatura

V4) Elemento: Acciones humanas
Descripción: Un requerimiento no debe utilizar verbos que describan acciones que no se puedan atribuir a un computador.
Justificación: Si se utiliza un verbo cuya acción –técnicamente– no puede realizar un software, esta siempre termina siendo **interpretada** como una acción que sí realiza un software. Para evitarlo, los verbos utilizados deben describir unicamente acciones propias de un software.
Ej. incorrecto: *The system shall consider fault history during CBITE.*
Ej. correcto: *The system shall read the fault history during CBITE.*
Comentarios: Otros ejemplos son: *determine, ignore, meet, examine, analyze*.
Fuente: Propuesto en esta investigación.

C

Instrumento de Evaluación de Elementos

Proyecto: *Automatización parcial de la revisión de aspectos de precisión, no-ambigüedad y verificabilidad de requerimientos de software escritos en lenguaje natural*
Instrumento: Cuestionario de Evaluación de Elementos
Versión: 1.0

Instrucciones Generales:

1. Lea detenidamente las siguientes afirmaciones sobre un requerimiento de software (RS).
2. Luego, respecto de cada afirmación marque la respuesta que le parezca más adecuada según las opciones que se presentan.
3. Utilice el espacio subrayado para indicar información adicional cuando se le solicita.
4. Al final del instrumento hay un espacio para que sugiera nuevos elementos que no considera esta lista y que, a su parecer, son importantes para determinar rasgos de imprecisión, ambigüedad y no verificabilidad en un requerimiento de software.

1) RS-1: *The system shall update the elapsed time indicator display every 100 ms.*
RS-2: *The system shall complete the IBITE test within 500 ms of command acknowledgement.*

Uno de los errores en RS-1 y RS-2 es que carecen de un valor de tolerancia para las restricciones de tiempo (1*00 ms y 500 ms*).

a) De acuerdo b) Los RS no tienen errores c) No tengo criterio para decidir. d) Los RS tienen errores pero el indicado no es uno de ellos. Explique: _____

Si su respuesta NO fue “a” en la pregunta anterior, siga con la pregunta 2.

1.a) En general, no especificar valores de tolerancia para mediciones de tiempo constituye un error en un RS.

a) De acuerdo b) En desacuerdo. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 2.

1.b) El error del que habla la pregunta 1 es un fallo de:

(Puede elegir más de una opción, excepto la "e", que descalifica las elecciones anteriores)

a) Precisión b) Ambigüedad c) Verificabilidad d) Otro e) No tengo criterio para decidir

Explique: _____

2) RS-3: *The system shall transmit ARINC 227 at a rate of 500 Hz in McMSP5.*

Uno de los errores en RS-3 es que carece de un valor de tolerancia para la restricción de frecuencia (*500 Hz*).

a) De acuerdo b) El RS no tiene errores c) No tengo criterio para decidir. d) El RS tiene errores pero el indicado no es uno de ellos. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 3.

2.a) En general, no especificar valores de tolerancia para mediciones de frecuencia constituye un error en un RS.

a) De acuerdo b) En desacuerdo. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 3.

2.b) El error del que habla la pregunta 2 es un fallo de:

(Puede elegir más de una opción, excepto la "e", que descalifica las elecciones anteriores)

a) Precisión b) Ambigüedad c) Verificabilidad d) Otro e) No tengo criterio para decidir

Explique: _____

3) RS-4: *The system shall light the expected_current LED if current equals 2 amps.*

Uno de los errores en RS-4 es que carece de un valor de tolerancia para la restricción de corriente (*current equals 2 amps*).

a) De acuerdo b) El RS no tiene errores c) No tengo criterio para decidir. d) El RS tiene errores pero el indicado no es uno de ellos. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 4.

3.a) En general, no especificar valores de tolerancia para mediciones de corriente constituye un error en un RS.

a) De acuerdo b) En desacuerdo. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 4.

3.b) El error del que habla la pregunta 3 es un fallo de:

(Puede elegir más de una opción, excepto la "e", que descalifica las elecciones anteriores)

a) Precisión b) Ambigüedad c) Verificabilidad d) Otro e) No tengo criterio para decidir

Explique: _____

4) En general, no especificar valores de tolerancia para mediciones físicas, tales como temperatura, presión, altitud, velocidad, y otras constituye un error en un RS.

a) De acuerdo b) En desacuerdo. Explique: _____

---

5) RS-5: *The system shall perform a CRC test of the config data in memory.*

Uno de los errores en RS-5 es que no especifica la condición que se debe cumplir para que se realice la acción principal (*CRC test*).

a) De acuerdo b) El RS no tiene errores c) No tengo criterio para decidir. d) El RS tiene errores pero el indicado no es uno de ellos. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 6.

5.a) En general, la ausencia de una condición para la acción principal constituye un error en un RS. Una excepción son los procesos que por su naturaleza se pueden reconocer como permanentes y periódicos, e.j. *Whatch dog timers*, *continuous built-in-tests*.

a) De acuerdo b) En desacuerdo. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 6.

5.b) El error del que habla la pregunta 5 es un fallo de:

(Puede elegir más de una opción, excepto la "e", que descalifica las elecciones anteriores)

a) Precisión b) Ambigüedad c) Verificabilidad d) Otro e) No tengo criterio para decidir

Explique: _____

6) RS-6: *The system shall continually perform a wrap test with the display head microcontroller.*

Uno de los errores en RS-6 es que utiliza el término *continually* que es no-determinístico o inexacto.

a) De acuerdo b) El RS no tiene errores c) No tengo criterio para decidir. d) El RS tiene errores pero el indicado no es uno de ellos. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 7.

6.a) En general, el uso del término *continually* para indicar cuando se ejecuta una acción constituye un error en un RS.

a) De acuerdo b) En desacuerdo. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 7.

6.b) El error del que habla la pregunta 6 es un fallo de:

(Puede elegir más de una opción, excepto la "e", que descalifica las elecciones anteriores)

a) Precisión b) Ambigüedad c) Verificabilidad d) Otro e) No tengo criterio para decidir

Explique: _____

7) RS-7: *The system shall process VHF signals from label 0xDB8 every second (+/- 100 ms) .*
RS-8: *The system shall monitor the status of the VHF line when landing.*

Uno de los errores en RS-7 y RS-8 es que utilizan los verbos process y *monitor* que no denotan una acción concreta.

a) De acuerdo b) Los RS no tienen errores c) No tengo criterio para decidir. d) Los RS tienen errores pero el indicado no es uno de ellos. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 8.

7.a) En general, el uso de verbos que no denotan acciones concretas (tales como *support*, *process*, *monitor*, *use* y otros) contituye un error en un RS.

a) De acuerdo b) En desacuerdo. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 8.

7.b) El error del que habla la pregunta 7 es un fallo de:

(Puede elegir más de una opción, excepto la "e", que descalifica las elecciones anteriores)

a) Precisión b) Ambigüedad c) Verificabilidad d) Otro e) No tengo criterio para decidir

Explique: _____

8) RS-9: *The system shall not transmit time and date information during an IBITE test.*

Uno de los errores en RS-9 es que expresa una de las infinitas acciones que el sistema no debe hacer (*shall not transmit [...]*) en vez de expresar exactamente lo que el sistema debe hacer (p.ej. *shall stop transmission of [...]*) durante un *IBITE test*.

a) De acuerdo b) El RS no tiene errores c) No tengo criterio para decidir. d) El RS tiene errores pero el indicado no es uno de ellos. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 9.

8.a) En general, describir funcionalidades que el sistema no debe realizar constituye un error en un RS.

a) De acuerdo b) En desacuerdo. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 9.

8.b) El error del que habla la pregunta 8 es un fallo de:

(Puede elegir más de una opción, excepto la "e", que descalifica las elecciones anteriores)

a) Precisión b) Ambigüedad c) Verificabilidad d) Otro e) No tengo criterio para decidir

Explique: _____

9) RS-10: *The system shall only enable one character size per menu page.*

Uno de los errores en RS-10 es que el término "*only*" restringe incorrectamente la acción del verbo principal (*enable*) en lugar de la cantidad de tamaños de caracteres (lo que implica que dicho sistema no hace otra cosa más que *allow one character size [...]*).

a) De acuerdo b) El RS no tiene errores c) No tengo criterio para decidir. d) El RS tiene errores pero el indicado no es uno de ellos. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 10.

9.a) En general, restringir la acción del verbo principal mediante el término "*only*" constituye un error en un RS.

a) De acuerdo b) En desacuerdo. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 10.

9.b) El error del que habla la pregunta 9 es un fallo de:

(Puede elegir más de una opción, excepto la "e", que descalifica las elecciones anteriores)

a) Precisión b) Ambigüedad c) Verificabilidad d) Otro e) No tengo criterio para decidir

Explique: _____

10) RS-11: *The system shall always display true airspeed in the upper left corner of the MFD.*
RS-12: *The system shall never display ground speed in the upper right corner of the MFD.*

Uno de los errores en RS-11 y RS-12 es que los términos "*always*" y "*never*" restringen incorrectamente la acción del verbo principal (*display*) de modo que se necesitaría tiempo infinito para verificar que estas acciones se cumplan.

a) De acuerdo b) Los RS no tienen errores c) No tengo criterio para decidir. d) Los RS tienen errores pero el indicado no es uno de ellos. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 11.

10.a) En general, el uso de los términos "*always*" y "*never*" constituye un error en un RS.

a) De acuerdo b) En desacuerdo. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 11.

10.b) El error del que habla la pregunta 10 es un fallo de:

(Puede elegir más de una opción, excepto la "e", que descalifica las elecciones anteriores)

a) Precisión b) Ambigüedad c) Verificabilidad d) Otro e) No tengo criterio para decidir

Explique: _____

11) RS-13: *The system shall consider cabin temperature valid for 10s (+/- 1 s) after it has been received.*

Uno de los errores en RS-13 es que el verbo principal denota una acción que no puede realizar un software *(to consider).*

a) De acuerdo b) El RS no tiene errores c) No tengo criterio para decidir. d) El RS tiene errores pero el indicado no es uno de ellos. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 12.

11.a) En general, el uso de verbos que denotan acciones que técnicamente no puede realizar un software constituye un error en un RS. p. ej. c*onsider, determine, examine, analyze* y otros.

a) De acuerdo b) En desacuerdo. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 12.

11.b) El error del que habla la pregunta 11 es un fallo de:

(Puede elegir más de una opción, excepto la "e", que descalifica las elecciones anteriores)

a) Precisión b) Ambigüedad c) Verificabilidad d) Otro e) No tengo criterio para decidir

Explique: _____

12) RS-14: *The system shall save the fault records and the faults log and the events log shall be cleared.*
RS-15: *The system shall display the Present Uncorrected Altitude as the own aircraft's altitude when the ARINC 429 label 204 Present Corrected Altitude message is not available or if its BSSM field is set to NCD and the ARINC 429 label 203 Present Uncorrected Altitude message is available and its BSSM field is set to Normal Operation or Functional Test, and the TCAS display is in Absolute mode.*

Considere la siguiente transformación para RS-15:
A- = ARINC 429 label 204 Present Corrected Altitude message is not available, B = BSSM field
C+ = ARINC 429 label 203 Present Uncorrected Altitude message is available, D = TCAS display
The system shall display the Present Uncorrected Altitude as the own aircraft's altitude when when (A- OR B=NCD AND C+ AND B=Normal Operation OR B=Functional Test) AND (TCAS=Absolute Mode).

Uno de los errores en RS-14 y RS-15 es que las agrupaciones de ANDs y ORs carecen de puntuación adecuada para indicar el sentido de las agrupaciones deseables.

a) De acuerdo b) Los RS no tienen errores c) No tengo criterio para decidir. d) Los RS tienen errores pero el indicado no es uno de ellos. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 13.

12.a) En general, no indicar claramente las agrupaciones de ANDs y ORs constituye un error en un RS.

a) De acuerdo b) En desacuerdo. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 13.

12.b) El error del que habla la pregunta 12 es un fallo de:

(Puede elegir más de una opción, excepto la "e", que descalifica las elecciones anteriores)

a) Precisión b) Ambigüedad c) Verificabilidad d) Otro e) No tengo criterio para decidir

Explique: _____

13) RS-16: *The system shall remove all sensors labeled as wrecked before computing rate of decend.*

Uno de los errores en RS-16 es que el verbo principal (*remove*) tiene varios significados en el contexto en que se utiliza.

a) De acuerdo b) El RS no tiene errores c) No tengo criterio para decidir. d) El RS tiene errores pero el indicado no es uno de ellos. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 14.

13.a) En general, el uso de verbos con multiples significados constituye un error en un RS.

a) De acuerdo b) En desacuerdo. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 14.

13.b) El error del que habla la pregunta 13 es un fallo de:

(Puede elegir más de una opción, excepto la "e", que descalifica las elecciones anteriores)

a) Precisión b) Ambigüedad c) Verificabilidad d) Otro e) No tengo criterio para decidir

Explique: _____

14) RS-17: *The system shall immediately log a sensor wrecked event when the label 255 bit 12 is set.*

Uno de los errores en RS-17 es que el adverbio (*immediately*) que modifica al verbo principal (*to log*) es muy general.

a) De acuerdo b) El RS no tiene errores c) No tengo criterio para decidir. d) El RS tiene errores pero el indicado no es uno de ellos. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 15.

14.a) En general, el uso de adverbios o adjetivos generales (tales como *acceptable*, *adequate*, *applicable*, *efficient*, *immediately*) constituye un error en un RS.

a) De acuerdo b) En desacuerdo. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, siga con la pregunta 15.

14.b) El error del que habla la pregunta 14 es un fallo de:

(Puede elegir más de una opción, excepto la "e", que descalifica las elecciones anteriores)

a) Precisión b) Ambigüedad c) Verificabilidad d) Otro e) No tengo criterio para decidir

Explique: _____

15) RS-18: *The system shall display any text on any menu that appears next to an "<" or ">" symbol, in capital letters.*
RS-19: *The system shall retrieve and/or store fault records from/into NVM following operator´s commands.*
RS-20: *The system shall be capable of operating under low power mode when power fault is active.*

Uno de los errores en RS-18, RS-19 y RS-20 es que utilizan los términos *"any"*, *"and/or"* y *"capable of"* que son no-determinísticos o inexactos y aceptan múltiples interpretaciones.

a) De acuerdo b) Los RS no tienen errores c) No tengo criterio para decidir. d) Los RS tienen errores pero el indicado no es uno de ellos. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, ya ha terminado con el cuestionario.

15.a) *En general, el uso de términos o construcciones inexactas o con múltiples significados (tales como "and/or","any", "not limited to", "capable of"*) constituye un error en un RS.

a) De acuerdo b) En desacuerdo. Explique: _____

Si su respuesta NO fue "a" en la pregunta anterior, ya ha terminado con el cuestionario.

15.b) El error del que habla la pregunta 15 es un fallo de:

(Puede elegir más de una opción, excepto la "e", que descalifica las elecciones anteriores)

a) Precisión b) Ambigüedad c) Verificabilidad d) Otro e) No tengo criterio para decidir

Explique: _____

Aporte de nuevos elementos:

Si usted lo considera necesario describa otros ejemplos de RS que presentan elementos o aspectos que le parece importante considerar para encontrar rasgos de ambigüedad, imprecisión y no-verificabilidad y que no están incluidos en este instrumento.

D

Resultados para la Evaluación de Elementos

Elementos

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
	Resultado A						Resultado B						Resultado C						Resultados				
Pregunta	Respuestas						Respuestas						Respuestas						Respuestas Finales				
#	a)	b)	c)	d)	e)	Comentario	a)	b)	c)	d)	e)	Comentario	a)	b)	c)	d)	e)	Comentario	a)	b)	c)	d)	e)
1	1						1					"Within" de cierta manera conlleva una tolerancia.	1						3				
1.a	1						1						1						3				
1.b	1						1						1						**3**				
2	1						1						1						3				
2.a	1						1						1						3				
2.b	1						1						1						**3**				
3	1						1						1						3				
3.a	1						1						1						3				
3.b	1						1						1						3				
4	1						1						1					Precisión para el verificador no para el programador, por tratarse de una entrada.	**3**				
5	1						1						1						3				
5.a	1						1						1						3				
5.b			1					1							1					1	**2**		
6	1						1						1						3				
6.a	1						1						1						3				
6.b			1					1							1					1	**2**		
7	1						1						1						3				
7.a	1						1						1						3				
7.b			1					1							1			Verbos que no denotan una acción concreta presentan cierta ambiguedad.		1	**2**		
8	1						1						1						3				
8.a	1						1						1						3				
8.b			1					1	1						1					1	**3**		
9	1						1						1						3				
9.a	1						1						1						3				
9.b			1						1						1						**3**		
10	1						1						1						3				
10.a	1						1						1						3				
10.b			1						1						1						**3**		
11	1						1					"Receive" también es un verbo que no realiza un SW.	1						3				
11.a	1						1						1						3				
11.b			1						1						1						**3**		
12	1						1						1						3				
12.a	1						1					RS-14 también tiene 2 shalls lo cual no es apropiado	1						3				
12.b		1						1	1					1						**3**	1		
13	1						1						1						3				
13.a	1						1						1						3				
13.b		1						1						1						**3**			
14	1						1						1						3				
14.a	1						1						1						3				
14.b		1					1							1				¨Inmediatamente¨ es muy ambiguo	1	**2**			
15	1						1						1						3				
15.a	1						1						1						3				
15.b		1						1						1						**3**			
Comentarios adicionales							**Comentarios adicionales**						**Comentarios adicionales**										
The system shall perform a FFT to the RF samples using a DIT algorithm. Este tipo de RS presenta el problema de precisión debido a que los algoritmos matemáticos aunque son conocidos, su implementación puede tener matices, restricciones de precisión, etc, de forma que son ambiguos a menos que se exprese explícitamente las restricciones del caso.							Dos shalls en un mismo requerimiento.						Sólo se están considerando los RS con ¨shall¨. El uso de ¨should¨, ¨will¨, ¨might¨ hace el RS no verificable.											

Notas:
1. Los grupos de columnas (B-G), (H-M) y (N-S) tienen las respuestas de los evaluadores.
2. Los resultados finales estàn en las columnas (T-X).
3. En las columnas (T-X), las celdas de color gris indican que esa es la respuesta esperada por el autor. Los nùmeros indican el nùmero de evaluadores que seleccionaron esa respuesta en el instrumento.
4. Una "coincidencia" con el autor ocurre cuando la mayorìa de votos està en una celda de color gris.

E

Instrumento de Evaluación de Puntajes

Proyecto: *Automatización parcial de la revisión de aspectos de precisión, no-ambigüedad y verificabilidad de requerimientos de software escritos en lenguaje natural*
Instrumento: Cuestionario de Evaluación de Pesos
Versión: 1.0

Instrucciones Generales:

1. Lea detenidamente las siguientes instrucciones.
2. A continuación se presenta una lista de elementos o características que debe tener un requerimiento de software (RS) para satisfacer las propiedades de precisión, verificabilidad y no-ambigüedad. A cada elemento se le ha asignado un puntaje en una escala de 1 a 3 donde, 1 significa que el elemento aporta alguna evidencia de que el RS viola una propiedad; 2 significa que aporta evidencia moderada y 3 significa que aporta mucha evidencia de que el RS viola la propiedad correspondiente.
3. Analice el puntaje que se ha asignado a cada uno de los elementos y marque la respuesta que le parezca más adecuada según las opciones que se presentan:
4. Cuando la elección sea b) utilice el espacio junto al texto "Nuevo Puntaje" para indicar el nuevo puntaje que usted asiganaría a dicho elemento.
5. Cuando la elección sea c) utilice el espacio adicional para indicar porqué razón considera que no tiene criterio para decidir o porqué razón no está de acuerdo con el criterio.

Precisión:

1) Un RS debe indicar valores de tolerancia para mediciones , tales como tiempo, temperatura, corriente eléctrica y voltaje.

Puntaje sugerido: 2 Pregunta: ¿Está usted de acuerdo con el puntaje asignado para este elemento?

a) De acuerdo b) En desacuerdo Nuevo Puntaje: _____ c) No tengo criterio para decidir

Explique: _____

2) Un RS debe indicar, de manera explícita, la(s) condición(es) que se deben cumplir para que se ejecute la acción principal.

Nota: En pocas ocaciones esto no es necesario por dos razones: (a) la acción es permanente, como un watchdog, (b) la condición se puede derivar claramente del contexto, no es recomendable pero sucede.

Puntaje sugerido: 3 Pregunta: ¿Está usted de acuerdo con el puntaje asignado para este elemento?

a) De acuerdo b) En desacuerdo Nuevo Puntaje: _____ c) No tengo criterio para decidir

Explique: _____

3) Un RS no debe incluir términos imprecisos o no determinísticos, tales como *continually* y *periodically*, que funcionan como adverbios para describir un evento.

Puntaje sugerido: 1 Pregunta: ¿Está usted de acuerdo con el puntaje asignado para este elemento?

a) De acuerdo b) En desacuerdo Nuevo Puntaje: _____ c) No tengo criterio para decidir

Explique: _____

4) Un RS no debe utilizar verbos que indican procesos generales que no se pueden asociar directamente con una acción concreta o específica. Algunos de los verbos a evitar son: *process*, *compare*, *check*, *monitor*, *use*, *define*, *interface*, *determine*, *support*, *ignore* y *meet.*

Puntaje sugerido: 1 Pregunta: ¿Está usted de acuerdo con el puntaje asignado para este elemento?

a) De acuerdo b) En desacuerdo Nuevo Puntaje: _____ c) No tengo criterio para decidir

Explique: _____

Verificabilidad:

5) Un RS no debe expresar la acción principal como una acción negativa.

Nota: Hay infinidad de acciones que el sistema no realiza en cada momento. Verificar que el sistema efectivamente no realiza dichas acciones es imposible en términos prácticos. Así, es necesario que el RS indique únicamente lo que el sistema debe hacer y no lo que no debe hacer.

Puntaje sugerido: 2 Pregunta: ¿Está usted de acuerdo con el puntaje asignado para este elemento?

a) De acuerdo b) En desacuerdo Nuevo Puntaje: _____ c) No tengo criterio para decidir

Explique: _____

6) Un RS debe utilizar el término "*only*" correctamente para evitar que este restrinja inadecuadamente la acción del verbo principal.

Nota: Cuando un RS tiene esta forma: *"... shall only display ...", "... shall only extract …" el verbo se* restringe de tal manera que el RS describe un sistema que sólo es capaz de realizar la acción descrita en ese RS (p. ej. *display* y *extract*), por ende es un sistema casi inutil. Si existe otro RS que indique otra acción para dicho sistema entonces habrá un conflicto entre dos RS incompatibles.

Puntaje sugerido: 3 Pregunta: ¿Está usted de acuerdo con el puntaje asignado para este elemento?

a) De acuerdo b) En desacuerdo Nuevo Puntaje: _____ c) No tengo criterio para decidir

Explique: _____

7) Un RS no debe utilizar los términos "*always*" y "*never*" para restringir la acción que describe un requerimiento pues estos hacen que se necesite tiempo infinito para verificar la acción descrita por el RS..

Puntaje sugerido: 2 Pregunta: ¿Está usted de acuerdo con el puntaje asignado para este elemento?

a) De acuerdo b) En desacuerdo Nuevo Puntaje: _____ c) No tengo criterio para decidir

Explique: _____

8) Un RS no debe utilizar verbos que describan acciones que no se puedan atribuir a un computador tales como c*onsider, determine, examine, analyze* y otros.

Puntaje sugerido: 1 Pregunta: ¿Está usted de acuerdo con el puntaje asignado para este elemento?

a) De acuerdo b) En desacuerdo Nuevo Puntaje: _____ c) No tengo criterio para decidir

Explique: _____

Ambigüedad:

9) Un RS debe describir las agrupaciones de ANDs y ORs mediante paréntesis explícitos o mediante la utilización de puntuación adecuada para evitar interpretaciones ambiguas.

Puntaje sugerido: 2 Pregunta: ¿Está usted de acuerdo con el puntaje asignado para este elemento?

a) De acuerdo b) En desacuerdo Nuevo Puntaje: _____ c) No tengo criterio para decidir

Explique: _____

10) Un RS no debe utilizar como verbo principal un verbo que tenga múltiples significados en el mismo contexto.

Puntaje sugerido: 1 Pregunta: ¿Está usted de acuerdo con el puntaje asignado para este elemento?

a) De acuerdo b) En desacuerdo Nuevo Puntaje: _____ c) No tengo criterio para decidir

Explique: _____

11) Un RS no debe utilizar adverbios vagos o generales que no indican con claridad alguna propiedad.

Nota: Algunos de dichos términos son: *about, acceptable, accurate, adequate, adjustable, applicable, appropriate, average, better, dependable, desirable, easy, efficient, excessive, immediately, insufficient, known, low, necessary, normal, optimum, other, possible, practicable, proper, quick, reasonable, recognized, relevant, safe, secure, significant, similar, simple, stable, suitable, temporary, timely, variable.*

Puntaje sugerido: 2 Pregunta: ¿Está usted de acuerdo con el puntaje asignado para este elemento?

a) De acuerdo b) En desacuerdo Nuevo Puntaje: _____ c) No tengo criterio para decidir

Explique: _____

12) Un RS no debe utilizar estructuras o formas que se saben no determinísticas, por ejemplo, *"and/or", "any", "not limited to", "because", "capable of".*

Nota: El problema de estas estructuras es que son generales y no indican referencias claras con lo que entorpecen los RS en vez de hacerlos descriptivos.

Puntaje sugerido: 2 Pregunta: ¿Está usted de acuerdo con el puntaje asignado para este elemento?

a) De acuerdo b) En desacuerdo Nuevo Puntaje: _____ c) No tengo criterio para decidir

Explique: _____

F

Resultados para la Evaluación de Puntajes

Puntajes

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
	Resultado A					Resultado B					Resultado C					Resultados Promediados					
Pregunta	Respuestas			Nuevo		Respuestas			Nuevo		Respuestas			Nuevo		Resultados			Viejo	Nuevo	
#	a)	b)	c)	puntaje	Comentario	a)	b)	c)	puntaje	Comentario	a)	b)	c)	puntaje	Comentario	a)	b)	c)	puntaje	puntaje	Balance
1	1					1						1		3		2	1		2	2.33	0.33
2	1					1					1					3			3	3	0
3	1						1		2			1		2		1	2		1	1.67	0.67
4	1					1						1		2		2	1		1	1.33	0.33
5	1				Una excepción es cuando hay variables discretas como booleanos o enumeraciones involucradas		1		3			1		3		1	2		2	2.67	0.67
6	1					1						1		2		2	1		3	2.67	-0.33
7		1		3			1		3			1		3			3		2	3	1
8	1				Usualmente always y never son términos que evidencias requerimientos problemáticos.	1						1		3		2	1		1	1.67	0.67
9	1					1					1					3			2	2	0
10	1						1		2			1		2		1	2		1	1.67	0.67
11	1					1					1			3		3			2	2	0
12	1					1					1					3			2	2	0

Notas:
1. Los grupos de columnas (B-F), (G-K) y (L-P) tienen las respuestas de los evaluadores.
2. Los resultados finales estàn en las columnas (Q-V).
3. En la columna U, el valor se calcula con esta fórmula ((IF(C*i*=1;E*i*;T*i*) + IF(H*i*=1;J*i*;T*i*) + IF(M*i*=1;O*i*;T*i*)) / 3) donde *i* es el número de fila correspondiente.

G

Ejemplos de Reportes del Prototipo

Reporte #7	
Title: Miscellaneous.	
Started:	Fri May 15 18:09:02 2009
Ended:	Fri May 15 19:05:44 2009
Elapsed time:	3402 seconds aprox 56.70 mins
Rqmts Analyzed:	1698
Average duration:	2.00 seconds/Rqmt

Descripción de cada uno de los elementos que utiliza en prototipo en los reportes (ver Apéndice B)

Atributo	Elemento	Número	Significado
Precisión	Tolerancia para tiempo	1	Un requerimiento debe indicar un valor de tolerancia para acciones que se realizan de manera periódica con mediciones físicas.
	Tolerancia para frecuencia	2	Un requerimiento debe indicar un valor de tolerancia para mediciones de tiempo.
	Tolerancia para voltaje	3	Un requerimiento debe indicar un valor de tolerancia para las mediciones de voltaje y corriente eléctrica.
	Condición explícita	4	Un requerimiento debe incluir una condición que indique cuándo se debe realizar la acción principal.
	No determinismo	5	Un requerimiento no debe incluir formas no determinísticas que funcionan como adverbios.
	Verbos generales	6	Un requerimiento no debe utilizar verbos que indican procesos generales que no se pueden asociar directamente con una acción atómica o específica.
Ambigüedad	Agrupaciones explícitas	1	Un requerimiento debe describir las agrupaciones de ANDs y ORs mediante signos de puntuación adecuados o bien utilizando paréntesis explícitos.
	Verbos ambiguos	2	Un requerimiento no debe utilizar como verbo principal un verbo que tenga múltiples significados en el contexto en que se utiliza.
	Adverbios vagos	3	Un requerimiento no debe utilizar adverbios vagos o adverbios generales para describir la acción principal.
	No determinismo	4	Un requerimiento no debe utilizar estructuras o formas que se saben no determinísticas.
Verificabilidad	Requerimiento negativo	1	Un requerimiento no debe expresar la acción principal como una acción negativa.
	Uso de *only*	2	Un requerimiento debe utilizar el término ``only'' correctamente cuando este trata de restringir únicamente la acción del verbo principal.
	Requerimiento infinito	3	Un requerimiento no debe utilizar los términos ``always'' y ``never'' para restringir la acción que describe un requerimiento.
	Acciones humanas	4	Un requerimiento no debe utilizar verbos que describan acciones que no se puedan atribuir a un computador.

H

Instrumento de Evaluación de Requerimientos

Proyecto: *Automatización parcial de la revisión de aspectos de precisión, no-ambigüedad y verificabilidad de requerimientos de software escritos en lenguaje natural*
Instrumento: Cuestionario de Evaluación de Requerimientos
Versión: 1.0

Instrucciones Generales:
- Lea detenidamente cada uno de los requerimiento de software que se presentan (RS).
- Luego, marque con una "x" una o más de las opciones según considere necesario.
- Utilice el espacio hacia la derecha de cada línea para explicar su seleeción pues esta información es sumamente valiosa (si es necesario use el dorso de la hoja).

G1 #	Texto del Requerimiento	Es impreciso	Es ambiguo	Es no-verificable	Tiene otro problema	No tiene ningún problema	No tengo criterio para decidir	Por favor explique su respuesta
1	The OFP CSCI shall remove all TCAS Intruder Icons and No Bearing Intruder Messages when in TA Fail mode.							
2	The OFP CSCI shall interface to the ARINC 453 buses as defined in the Application Processor CCA Design Specification for the SuperSet Family.							
3	The DCP CSCI shall turn on the fan when the average temperature reading from both backlight temperature sensors, the power supply temperature sensor and the display head card temperature sensor is greater than twenty-five (25) degrees C.							
4	The OFP CSCI shall display the OFP, DCP, and Configuration Database CRC values when the active page is set to the software version overlay.							
5	The FTS CSCI shall support printing the display serial number.							
6	The OFP CSCI shall use the range received in the Weather Radar Display Data Bus message (ARINC Characteristic 708A-3, message label 055, bits 43 through 48) as the Display Range when the WX overlay is currently being displayed. Received range of all 0's means 320 nm.							
7	The OFP CSCI shall parse bits 11 through 15 of message labels 130, 131, and 132 as the TCAS Intruder Target Number.							
8	The OFP CSCI shall display the Flight ID tag in the same color as its associated Intruder Icon when the Flight ID has been received for that Intruder, the ADS-B File Receive Enable is set, the ADS-B Control Select is set to 1, the configuration database indicates that TA Intruder's will have Flight IDs, and no RA Intruders are present.							
9	The OFP CSCI shall display the current temperatures of the display when the active page is set to the status page.							
10	The OFP CSCI shall display the n number of TCAS Intruder Icons with the highest priority, where n is determined by the TCAS Display Limit value.							
11	When a Verify Flash command is received, the System Boot Program shall respond with a Command Accepted response.							
12	The System shall pass the BMU revision to the fast and slow analog airframe software.							
13	The File Manager Component shall be capable of handling up to four (4) clients simultaneously.							
14	The System shall make available a data parameter indicating the percentage of used memory for the current data cartridge. This data parameter may be used for display to the aircrew, or to display a warning to the aircrew when the data cartridge is more than certain percentage full. This percentage should not reflect any reserved or unusable space on the data cartridge.							
15	The System Boot Program shall respond with a Flash Ok response if the calculated CRC of Flash memory due to Program Flash commands matches the CRC contained in the initial Program Flash command.							
16	The System Boot Program shall send a Command Rejected response when an unrecognized command is received.							
17	The System shall transfer to the repository accelerometer and frequency built-in test status.							
18	The System shall write all commands received on the Ethernet or Serial interfaces to the DSP processor memory at the configured address for Host to DSP commands.							
19	The MIL_STD-1553 Interface shall validate all received MIL-STD-1553 messages.							
20	The System shall update the optional time synchronization discrete, if configured, to indicate the real-time clock has been updated.							

Proyecto: *Automatización parcial de la revisión de aspectos de precisión, no-ambigüedad y verificabilidad de requerimientos de software escritos en lenguaje natural*
Instrumento: Cuestionario de Evaluación de Requerimientos
Versión: 1.0

Instrucciones Generales:
- Lea detenidamente cada uno de los requerimiento de software que se presentan (RS).
- Luego, marque con una "x" una o más de las opciones según considere necesario.
- Utilice el espacio hacia la derecha de cada línea para explicar su seleeción pues esta información es sumamente valiosa (si es necesario use el dorso de la hoja).

G2 #	Texto del Requerimiento	Es impreciso	Es ambiguo	Es no-verificable	Tiene otro problema	No tiene ningún problema	No tengo criterio para decidir	Por favor explique su respuesta
1	The OFP CSCI shall accept the range of values of 0 through n-1 for the Text ID fields of the Text Attribute Databases Table where n is defined by the Text Count entry of the Configuration Database.							
2	The OFP CSCI shall discard an ARINC 429 Intruder File if an ARINC 429 label 133 BSSM is to set to any value except "NCD" and does not match the SSM of the ARINC 429 label 130 BSSM received for the same Intruder Number.							
3	The OFP CSCI shall place the SINKRATE text message as defined by the SINKRATE entry of the Text Attribute Database.							
4	The DCP CSCI shall report a POWER SUPPLY TEMP SENSOR FAULT to the application processor when requested if the power supply temperature sensor reading is less than minus fifty-five (-55) degrees C.							
5	The OFP CSCI shall provide operator controls, as specified in the Configuration Database, to command the display of either a reduced scan angle view or a normal scan angle view to the WX radar.							
6	The OFP CSCI shall parse bits 23, 24 and 25 of the TCAS Mode Compatible message as the Sensitivity Level Status.							
7	The OFP CSCI shall process a maximum of thirty (30) TCAS Intruder Target inputs.							
8	The OFP CSCI shall periodically perform a Frame Rate Test when neither initiated BIT or start-up BIT are being performed at minimum once every five (5) seconds.							
9	The OFP CSCI shall transmit a status request to the DCP CSCI as specified in the DCP SRD at a minimum 20 Hz rate.							
10	The OFP CSCI shall parse the TERRAIN AWARENESS CAUTION bit from the TAWS Alert Indication #2 data.							
11	The OFP CSCI shall ignore all messages whose Source Destination Identifier does not specify that the LRU in which the OFP is operation is a destination for the message.							
12	The OFP CSCI shall only process ARINC 429 Intruder Files when the following conditions are met and E-TCAS Capability is enabled:							
13	The OFP CSCI shall only process an ADS-B Intruder File if the following conditions are met:							
14	The OFP CSCI shall not display the Altitude nor Vertical Direction for a TCAS No Bearing Target when the Altitude for the No Bearing Target is unavailable.							
15	The OFP CSCI shall only process received ARINC 429 ADS-B Intruder File messages when the ARINC 429 label 013 TCAS DITS Control message ADS-B File Receive enable (bit 15) is set.							
16	If the WX Slave mode control is set to a Discrete input pin by the Configuration Form in appendix F, then the MFD shall enter Slave mode when the pin is low (0)							
17	If WX Right is selected, the MFD shall position the Aircraft and the origin of the radar display radials, on the left lower corner of the Radar picture display area							
18	The OFP CSCI shall display the Below Altitude Band text message as defined by the Below Altitude Band text entry of the Text Attribute Database, when the Altitude Select mode is set to 2.							
19	The OFP CSCI shall display the overlay selection text as specified by the disabled overlay entry of the Text Attribute Database, when the given overlay is disabled.							
20	The OFP CSCI shall display the Indicator Fault text message as specified in the Configuration Database when display fault (bit 20 =1) is received in the Weather Radar Display Data Bus message (ARINC Characteristic 708A-3, message label 055) and the WX overlay is currently being displayed.							

Proyecto: *Automatización parcial de la revisión de aspectos de precisión, no-ambigüedad y verificabilidad de requerimientos de software escritos en lenguaje natural*
Instrumento: Cuestionario de Evaluación de Requerimientos
Versión: 1.0

Instrucciones Generales:
- Lea detenidamente cada uno de los requerimiento de software que se presentan (RS).
- Luego, marque con una "x" una o más de las opciones según considere necesario.
- Utilice el espacio hacia la derecha de cada línea para explicar su seleeción pues esta información es sumamente valiosa (si es necesario use el dorso de la hoja).

G3 #	Texto del Requerimiento	Es impreciso	Es ambiguo	Es no-verificable	Tiene otro problema	No tiene ningún problema	No tengo criterio para decidir	Por favor explique su respuesta
1	The OFP CSCI shall not expect a TCAS Intruder File if the Block Word Count is set to 1.							
2	The OFP CSCI shall determine that the received data is in Normal Operation when the DSSM is set to 0.							
3	The OFP CSCI shall only accept the ADS-B Intruder File if the Data Type Continuation Bit for ADS-B Intruder Flight ID message 3, and ADS-B Intruder Extended TCAS message 2 are set to 1.							
4	The OFP CSCI shall only accept the ADS-B Intruder File if the Ending ADS-B Intruder File message's ADS-B Message Count matches the Starting ADS-B Intruder File message's ADS-B Message Count.							
5	The OFP CSCI shall determine whether to use the Control Data portion of the Weather Radar display data passed in from the R/T or ignore it based upon the value of the Control Accept bits.							
6	The OFP CSCI shall support the display of TCAS traffic advisory and TCAS mode settings as part of its TCAS overlay.							
7	The MFD shall convert the weather precipitation data received on the Data lines in to color data properly located on the screen.							
8	The OFP CSCI shall interpret bits 17 through 18 of a received ARINC 429 label 366 Extended Range message as the TCAS Intruder Target's Vertical Direction.							
9	The FTS CSCI shall support displaying a fault log.							
10	The OFP CSCI start-up BIT shall perform all tests executed in continuous BIT .							
11	The DCP CSCI shall use the FPGA only after receiving the 'Configuration Done' signal, indicating the FPGA is operational.							
12	The FTS CSCI shall support testing the fans.							
13	The FTS CSCI shall support reseting a temperature log.							
14	The FTS CSCI shall support toggling between the NVIS and Daylight modes.							
15	The OFP CSCI shall interpret bits 30 and 31 of the following received ARINC 429 message labels as the Binary Sign Status Matrix (BSSM): Label 130, label 131, label 132, label 133, label 203, and label 204.							
16	Weather and Terrain shall not be displayed together (overlaid)							
17	The OFP CSCI shall only remain in Absolute TCAS Altitude mode for 30 seconds before automatically switching back to Relative TCAS Altitude mode when the Present Corrected Altitude is not available, and the Present Uncorrected Altitude is less than 18,000 feet.							
18	The OFP CSCI shall only display one color per line of text.							
19	The OFP CSCI shall only update a valid character size with the character size in a message containing the page erase command							
20	The OFP CSCI shall only display one flashing status (i.e. no flash or flashing) per line of text.							

Proyecto: *Automatización parcial de la revisión de aspectos de precisión, no-ambigüedad y verificabilidad de requerimientos de software escritos en lenguaje natural*
Instrumento: Cuestionario de Evaluación de Requerimientos
Versión: 1.0

Instrucciones Generales:
- Lea detenidamente cada uno de los requerimiento de software que se presentan (RS).
- Luego, marque con una "x" una o más de las opciones según considere necesario.
- Utilice el espacio hacia la derecha de cada línea para explicar su seleeción pues esta información es sumamente valiosa (si es necesario use el dorso de la hoja).

G4 #	Texto del Requerimiento	Es impreciso	Es ambiguo	Es no-verificable	Tiene otro problema	No tiene ningún problema	No tengo criterio para decidir	Por favor explique su respuesta
1	The DCP CSCI shall report a BACKLIGHT WATCHDOG FAULT to the application processor when requested if the backlight watchdog fails to disable the backlight after not being strobed for greater than one (1) second.							
2	The OFP CSCI shall display the bezel key label text and perform the bezel key actions as defined in the Configuration Database for the SDBYoverlay.							
3	The OFP CSCI shall display the message of an off-scale TA intruder in yellow.							
4	The OFP CSCI shall use the Present Corrected to compute the intruder's absolute altitude when the ARINC 429 label 204 Present Corrected Altitude message is available and its BSSM field is set to Normal Operation or Functional Test, and the TCAS display is in Absolute mode.							
5	The OFP CSCI shall send a request to the DCP CSCI for display head status as defined in the OFP/DCP ICD at a 60 Hz rate.							
6	The OFP CSCI shall Interpret range for TCAS Intruder's at a resolution of 1/16 of a Nautical Mile giving the range value a range of 0 to 127.9375 Nautical Miles.							
7	The OFP CSCI shall parse bits 26 through 29 of the TCAS Mode Compatible message as the Reply Information Status.							
8	The OFP CSCI shall display the bezel key label text and perform the bezel key actions as defined in the Configuration Database for the TCAS overlay.							
9	The OFP CSCI shall declare an I/O FPGA READ/WRITE FAULT if the register read/write test on the I/O FPGA fails.							
10	The ABP CSCI shall send a shutdown message on the maintenance channel if the CRC test of the factory test software fails.							
11	The OFP CSCI shall discard an ARINC 429 Intruder File if an ARINC 429 label 132 BSSM is to set to any value except "NCD" and does not match the SSM of the ARINC 429 label 130 BSSM received for the same Intruder Number.							
12	The OFP CSCI shall place a duplicate label of the control option that the bezel rotary dial has control of above the value setting when the bezel rotary dial has control of a control option.							
13	The OFP CSCI shall clear out any TCAS Intruder's data when the data for TCAS Intruder has been dropped from the current TCAS Intruder File.							
14	The FTS CSCI shall only be accessible when the external pins are configured for the factory test mode.							
15	The OFP CSCI shall only process a received ARINC 429 label 203 Present Uncorrected Altitude message when its BSSM field is set to Normal Operation or Functional Test.							
16	The OFP CSCI shall only display a TCAS NT or PA Intruder Icon when the relative altitude of the intruder is unavailable or the altitude of the intruder is within the selected altitude band.							
17	The OFP shall only process the Control Data (bits 11 through 64) of a received TAWS Display Radial message (ARINC Characteristic 708A-3, message label 055) if the message's Control Accept field Bit 9 = 1 and the Ind 2 Discrete is high.							
18	The OFP CSCI shall only process an ADS-B Intruder Packet as valid if the following conditions are met:							
19	The OFP CSCI shall only parse ADS-B Intruder Extended TCAS messages 1 and 2 for Intruder IDs 0 through 31 when the Intruder has not already been defined by the TCAS Intruder File.							
20	The OFP CSCI shall only process the TCAS Intruder Target's Relative Altitude when the received ARINC 429 label 131 TCAS Intruder Altitude and Vertical Direction message's BSSM field is set to Normal Operation or Functional Test.							

I

Participantes en los Experimentos

Eduardo Trejos

EXPERIENCE:

01/2005 - Present **Software Engineer / Team Lead / Software Quality Engineer**

- Requirements-based Testing and Structural Coverage Analysis for a Level A Instrumented Landing System (ILS) using processor ADSP 2181 from Analog Devices, emulator ADSP EZICE 218X and EZICE software.

- Requirements-based Testing and Structural Coverage Analysis for a level B Display Control Program of Multi-function Display (MFD) using microcontroller ATMEL ATmega128L (C and assembly), JTAG ICE emulator, AVR Studio and IAR Systems software.

- Requirements-based Testing and Structural Coverage Analysis for a level A Arc Fault Circuit Breaker using microcontroller PIC16F684 (C and assembly), MPLAB ICE 2000 and HI-TECH PICC C Compiler.

- Requirements-based Testing and Structural Coverage Analysis for a level B Multi-Function Display (MFD) and Primary Flight Display (PFD).

- Development and Requirements-based Testing of a level D Flight Deck Area Illumination software in C using microcontroller Freescale MC9S12C64 (C) and Cosmic Tools.

- DO-178B Configuration Management: Software Configuration Control Board Records, Configuration Management Records, Scheduling Reviews, Control of Artifacts.

- Development of High Level Requirements for a level A Vertical Instrument Display.

- Team Lead, Development of High and Low Level Requirements, Requirements-based Testing and Structural Coverage Analysis for a level C Bootloader using Green Hills MULTI software for a MPC5554 processor.

- Team Lead for a level C OS Requirements development and Testing using Green Hills MULTI and Integrity software for a MPC5554 processor.

Eduardo Trejos

- Team Lead for a level C Onboard Maintenance System Verification of a Digital Low Range Altimeter for a TI 5509 DSP processor.
- Software Quality Engineer for a level C 787 Audio Subsystem

EDUCATION:

12/2004 **Post-Bachelor Degree -** Electronics Engineering
Costa Rica Institute of Technology, Cartago, Costa Rica

SKILLS:

Skill Name	Skill Level	Last Used/Experience
8086 Assembler	Intermediate	3.5 years ago/1 year
VHDL	Basic	3.5 years ago/0.5 years
C	Intermediate	Currently used/2.5 years
C++	Basic	1 year ago/0.5 years
PICC	Intermediate	1.5 years ago/2 years
LabView	Intermediate	2.5 years ago/1 year
PIC 16F87X microcontrollers	Intermediate	1.5 years ago/2 years
PIC 16F684 Microcontroller	Intermediate	1.5 years ago/0.5 years
MC9S12C64	Basic	1 year ago/0.5 years
MPC5554	Basic	Currently used/0.5 years
CCNA 1 : Networking Basics (Cisco)	Intermediate	3 years ago/0.5 years

Roger Fernandez.

EXPERIENCE:

09/2007 - Present **Software Verification Engineer**

- Dry running, run for score and run for coverage of test procedures for a Level A Vertical Indicator Display using a Hitachi SH7708 processor (C++).

- Requirements based testing and structural coverage analysis for a level C Digital Low Range Radio Altimeter (specifically on the Onboard Maintenance System).
  - Experience gained in Texas Instruments 5509A processor and development tools such as Code Composer Studio, ASTAT and C programming language.
  - Activities include test case and test procedure development, formal reviews and structural coverage analysis following guidelines in DO-178B standard. Also memory, stack and timing analysis.

- Requirements based testing and structural coverage analysis for a level C Distance Measuring Equipment (specifically on the Onboard Maintenance System).
  - Activities include team leading and other project management activities, test case and test procedure development, informal reviews and timing analysis. Also structural coverage analysis following guidelines in DO-178B standard.

Roger Fernandez.

EDUCATION:			
09/2008	**Post-Bachelor Degree -** Electronics Engineering		
	Costa Rica Institute of Technology, Cartago, Costa Rica		

SKILLS:	Skill Name	Skill Level	Last Used/Experience
Programming Languages	8086 based Assembler	Intermediate	1 year ago/2.5 years
	Verilog	Intermediate	1 year ago/2 year
	C++	Intermediate	2 years ago/0.5 years
	C	Intermediate	Currently used/4 years
	PICC	Intermediate	1.5 year ago/1.5 years
	C #	Intermediate	1 year ago/1 year
Tools	CodeComposer	Intermediate	Currently used/1.8 years
	ASTAT	Intermediate	Currently used/1.8 years
	Matlab	Intermediate	1 year ago/2 years
Protocols	ARINC429	Intermediate	Currently used/1.8 years
	USB	Intermediate	1.5 years ago/1.5 years
	RS232	Intermediate	1.5 years ago/1 year
Other skills	CCNA 1 : Networking Basics (Cisco) CCNA 2: Routers and Routing Basics	Intermediate	1 years ago/0.5 years

Leonardo Jiménez

EXPERIENCE:

2007-2008 **Jr. Software Engineer**

- Verification of a Digital Low Range Radio Altimeter On Board Maintenance System
  - Design of test cases and procedures for verification of low and high level software requirements.
  - Acquired experience in C programming language for a Texas Instruments DSP 5509A processor.
  - Experience in Code Composer Studio debugging environment.
  - Experience gain with formal review of documents and code coverage activities.
  - Experience gain in configuration management as support of team lead.
  - Experience gain in software configuration management (SCM) tools.

- Verification of Digital Low Range Radio Altimeter Primary Calculation Module.
  - Design of test cases and procedures for verification of high level software requirements.
  - Acquired experience in DSP algorithms (FFT and FIR filters) using the Texas Instruments DSPs.

- Development of Avionyx Software Test Automation Tool (ASTAT).
  - Development of embedded software testing environment module for Texas Instruments processors.
  - Design and Coding of source code in C++ and Java programming languages.
  - Development of communication protocol via TCP/IP.
  - Writing of API documentation for all currently available functions.
  - Experience gain with test procedure automation for embedded software.
  - Experience with Eclipse and Rhino development and debugging.

Leonardo Jiménez

EDUCATION:

1999 - 2006 **Bachelor's Degree –** 5-year-program Electronics Engineering in the Technological Institute of Costa Rica.

Accredited engineering program by the Canadian Engineering Accreditation Board (CEAB) with international education level ISCED 5

June - 2006 **Advanced Level** – English language

Diploma of conclusion, of the British Institute, Costa Rica

2002 - 2005 **Basic Level** – German language

German basic level international certificate. Goethe Zentrum. Costa Rica

SKILLS: Skill Name	Skill Level	Last Used/Experience
DO-178B	Advanced	Currently Used/1 year
C/C++	Advanced	Currently Used/3 years
TI Code Composer Studio	Advanced	Currently Used/1 year
PIC microcontrollers	Advanced	Currently Used/3 years
iNotion	Advanced	Currently Used/1 year
Tortoise SVN	Advanced	Currently Used/1 year
ASTAT	Advanced	Currently Used/1 year
Mechatronics	Beginner	Currently Used /6 months
PHP & MySQL	Beginner	Currently Used/1 year
DSP	Intermediate	1 year/1 year
MATLAB	Beginner	1 year/2 years
Microprocessor Architecture	Intermediate	2 years/1 year
FPGA design	Intermediate	2 years/2 year
Verilog (VHDL)	Intermediate	2 years/1 year
Assembly for PC	Intermediate	2 years/1 year
IEEE – 754	Beginner	2 years/6 months

DSP: Digital Signal Processing

DO-178B: Software Considerations in Airborne Systems and Equipment Certification

Esteban Sánchez Chinchilla

Project Manager, Electronic Engineer

Cinco Esquinas, Tibás SAN JOSÉ, COSTA RICA- Single
Id: 111490017 - Age: 26
Home phone: 22580913 –Mobile phone: 83169324 - est2001@hotmail.com

Professional Background

Electronic Engineer graduated from ITCR (Costa Rican Institute of Technology). 2 years of experience working as a development engineer for safety critical embedded systems in the avionics industry. 2 years of experience as project manager, directing software development and verification/validation projects following PMI Project Management Body of Knowledge processes. Wide experience in the following areas: PMI PMBOK, Software Life Cycle Development Process, software verification and validation (DO-178B process), reverse-engineering processes. Experience working on-site in the United States, providing consulting, training and audit support. Skills include but are not limited to: Interpersonal skills, team spirit, negotiation skills, motivational skills, Leadership, quality, fully bilingual (Spanish/English), objective-oriented, responsibility, commitment and thoroughness.

Work Experience

4 years of experience

Avionyx S.A

Position: Project Manager
Period: May, 2005 – Currently working
Functions and Achievements:
Started the career as a SW engineer in charge of the activities for the SW life cycle process under the DO-178B standard, such as: requirements capture, design, coding, SW-HW integration, verification and validation.
Promoted to Project Team Leader after 1 year of excellent performance and finally promoted to Project Manager after 2 years. In this position, the responsibilities are the typical project management activities such as scheduling, monitoring and controlling, risk management, development strategy, status report to customer, team oversight. The major achievements include the successful completion (under budget and schedule) of the following projects:

- Verification of the Operational Software and the Smoke Detection Software for the Smoke Detection System
- Flight Deck Area Illumination software
- Stall Warning and Protection Computer verification

ICE

Position: Software Developer
Period: July, 2004 – January, 2005
Functions and Achievements:
Functions: Development engineer for the Project “Analysis of the telecommunication infrastructure”, which consisted on the development of a software tool based on

geographical information systems (GIS). I was in charge of the software development, test and debugging, as well as the delivery of training for the final users. Achievements: Provided the company with a valuable software tool to aid in the optimization of the telecommunications network, as well as the planning of new infrastructure.

Formal Education

Costa Rican National University
Master degree in Administration of information technology, with emphasis on Project Management
Currently studying

Costa Rican Institute of Technology
Bachelor degree in electronic engineering
November, 2004
LICENCIATURA

Languages

Native Language: Spanish Second language: English 100 %

Informal Education

Aerotica
Ground Lessons (Pilot lessons)
Currently coursing

RTCA Inc.
DO-254
June, 2008

Avionyx SA
Effective Training
May, 2007

New Horizons
Microsoft Project
March, 2007

English Learning Center
English Certification
November, 2005

Avionyx SA
DO-178B
July,2005

Fernando Bogarín Bonilla

Personal information

ID number: 1-960-935
Status: Single
Date of Birth: February 8th, 1977
Telephones C R: 2237-2126, 8363-1803 (cell phone)
Postal Address CR: Calle 5, Avenidas 11 y 13, 3000 Heredia
E-mail: bogarin@gmail.com

Education

- Colegio Universitario de Alajuela, Degree in Tourism with emphasis in Travel Agencies and Transportation, 1997.
- Escuela de Ingeniería en Computación, Instituto Tecnológico de Costa Rica (ITCR), 2006.

Languages

- English: Read, write, and speak proficiently.
- Spanish: Read, write, and speak proficiently.
- Japanese: Very basic read and write skills.

Programming languages

- Scheme
- Assembly
- Java
- C, C++
- SQL Server
- Macromedia Director (Lingo) (Basic)
- .NET Framework 2003 (currently self learning)

Work experience

- Best Western San José Downtown Hotel.
  - Receptionist, phone operator, cashier and night-shift auditor, 1997-1998;
- Barceló Parque del Lago Hotel.
  - Receptionist, phone operator, cashier and night-shift auditor, 1998-1999.
- Avionyx S.A. 2006 - current
  - Validation and Verification Engineer, Software Quality Assurance Engineer.

Other studies

- New Horizons open courses (Microsoft Office Packages).1998.
- Japanese Level I and II. Embassy of Japan. 1998.

Academic Experience

- Teacher assistant:
  - With Professor Milton Villegas Lemus. Escuela de Ingeniería en Computación, ITCR. November 2004 – June 2006.
    - Computer Organization and Assembly Language course.
    - Computers Architecture course.
    - Compilers and Interpreters course.
- Collaborator in learning by Construction Camps.

- With Professor Milton Villegas Lemus as coordinator in the Palmares Public High school. November 2005, March and April 2006.

Academic projects

- Data structures:
  - Object oriented programming, low level programming, sort, search and other system management.
- Electronic components:
  - Circuit construction with PIC 16F84A y 16F876.
  - Circuit design, virtual and physical.
- Data bases:
  - Geographic Information Systems.
  - Graphic and Analytical systems.
  - Financial Systems.
- Software Verification:
  - Proficient in the methods, rules and tools utilized in the development of the software requirements
  - Experienced in the design and development of software requirements, software architecture, and software design description
  - Experienced in embedded software development and verification to DO-178B standards using tools such as GreenHills (Integrity RTOS, MULTI and AdaMULTI), CodeWarrior, Visual DSP and Microsoft Visual Studio
  - Experienced in manual and automated software testing techniques to include system level test as well as unit testing with VectorCAST (for C)
  - Experienced in manual and automated software testing techniques to include system level test as well as unit testing with Rational Test Real Time
  - Experienced with various programming languages to include C/C++, LabVIEW, Assembly for several architectures and various scripting languages
  - Experienced in software configuration management throughout software life cycle processes to DO-178B standards using tools like Perforce and DOORs
  - Knowledgeable of the following communication protocols: ARINC 429, ARINC 653, CAN Bus (Controller Area Network), SPI (Serial Peripheral Interface) and AFDX (Avionics Full-Duplex Switched Ethernet)

For references

- Milton Villegas Lemus. Professor of the Instituto Tecnológico de Costa Rica (ITCR). Telephone: 8812-3609.
- Yuen Law. Professor of the Instituto Tecnológico de Costa Rica (ITCR). Telephone: 8843-1932
- Ana Quirce. Organization and Methods Assistant. Banco Lafise. Telephone: 2246-0859.
- Eduardo Montero. General Manager Hampton Inn. Telephone: 2436-0005.

Daniel Pérez
dperez@avionyx.com

Summary of Skills

- Experienced in the design and development of software requirements, software architecture, and software design description
- Proficient in the methods, rules and tools utilized in the development of the software requirements
- Experienced in the DO-178B software life cycle processes for levels A through D
- Experienced in embedded software development and verification to DO-178B standards using tools such as GreenHills (Integrity RTOS, MULTI and AdaMULTI), CodeWarrior, Visual DSP and Microsoft Visual Studio
- Proficient in manual and automated software testing techniques to include system level test as well as unit testing with tools such as VectorCAST (for C and Ada) and Rational Test Real Time
- Highly skilled in software structural coverage tools such as VectorCOVER, CodeTest and Rational Test Real Time
- Experienced with various programming languages to include C/C++, Ada95, LabVIEW, Verilog, Assembly for several architectures and various scripting languages
- Experienced in software configuration management throughout software life cycle processes to DO-178B standards using tools like Perforce and DOORs
- Knowledgeable of the following communication protocols: ARINC 429, ARINC 615, ARINC 665-3, Byteflight, CAN Bus

Professional Experience

Avionyx Inc., Heredia, Costa Rica (2006 – present)

Held multiple positions of increasing responsibility on FAA software certification projects for US clients in partnership to accomplish milestones and meet deadlines while maintaining profitability.

Key projects for Avionyx:

Cabin Pressure Control Integrated System

7 months

As the software team lead, guided and managed a team of 5 engineers to develop and verify the embedded software for a Cabin Pressure Control Integrated System using the ARM 7 processor and Keil IDE (C language) according to DO-178B Level C and D

- Led the development of the plans and standards with the client, including: PSAC, SVP, SDP, SCMP, SRS, SCS, and SDS
- Designed and developed high level software requirements from the system requirements and customer specifications
- Designed the software and architecture and developed the software design description
- Developed the source code according to the software requirements and software architecture
- Designed automated white box (emulator) and black box requirement based test cases and procedures
- Contributed to the system's hardware design to include all considerations for software development and verification

Audio Control Panel and Audio Gateway Unit, Seattle, WA

12 months

As software verification and development engineer, verified embedded software for an Audio Control Panel and an Audio Gateway Unit using the Blackfin 535 processor and Visual DSP IDE (C language) according to DO-178B Level C

- Designed manual white box (emulator) and black box requirement based test cases and procedures
- Managed verification of data loader software compliant to the ARINC 615 and ARINC 665-3 protocols
- Instrumented application source code to gather structural coverage using VectorCOVER overcoming memory and timing limitations
- Managed the structural coverage analysis of a verification team comprised of more than 10 engineers to achieve statement coverage
- Developed software verification environment setup procedures
- Performed code reviews of source code to meet software coding standards
- Maintained Software Detailed Design documentation in accordance with requirements and software updates

Board Support Package

8 months

As Software Verification Engineer verified embedded software for a Board Support Package using a Freescale 9S12 microcontroller and CodeWarrior development studio (C++) according to DO-178B Level B

- Designed manual white box (emulator) and black box requirement based test cases and procedures
- Performed code reviews of source code to meet software coding standards
- Instrumented application source code to gather structural coverage using CodeTEST overcoming memory limitations
- Analyzed and assessed structural coverage results to achieve decision coverage

Daniel Pérez
dperez@avionyx.com

Key projects for Avionyx (continued)**:**

9 months

Integrated Vehicle Health Management Unit System

As Software Verification Engineer, Configuration Manager and Technical Lead, verified embedded software for a Integrated Vehicle Health Management Unit System using a Freescale Power PC Architecture processor and Green Hills Integrity RTOS in conjunction with AdaMULTI (Ada95) according to DO-178B Level B

- Reviewed software requirements to comply with the project standards and DO-178B objectives
- Designed automated white box (unit level) requirement based test cases and procedures using VectorCAST
- Managed configuration control of the software throughout the software life cycle
- Analyzed and assessed structural coverage results to achieve decision coverage
- Redesigned and maintained software testing environment for increased efficiency, from a 3-person station to an expandable network-based system that supports verification team of more than 12 engineers to work remotely
- Developed software verification environment setup procedures
- Trained verification team of more than 12 engineers on the VectorCAST/Ada testing tool usage and provided support throughout the verification effort, solving issues individually or contacting the tool manufacturer

4 months

Electronic Flight Bag System

As Software Verification Engineer, Configuration Manager and Technical Lead verified embedded software for a Electronic Flight Bag System using a Freescale Power PC Architecture processor and Metrowerks CodeWarrior Development Studio (C language) according to DO-178B Level C

- Reviewed software requirements to comply with the project standards and DO-178B objectives
- Designed manual and automated white box (emulator) and black box requirements-based test cases and procedures
- Analyzed and assessed structural coverage results to achieve statement coverage
- Maintained and supported testing environment (hardware/software) for a verification team of more than 5 engineers

2 months

Vertical Instrument Display

As Software Verification Engineer, verified embedded software for a Vertical Instrument Display using a Hitachi processor (C++) according to DO-178B Level A

- Instrumented application source code to gather structural coverage using VectorCOVER overcoming memory and timing limitations
- Analyzed and assessed structural coverage results to achieve modified condition decision coverage (MC/DC)

5 months

Electronic Flight Instrument System

As Software Verification Engineer verified embedded software for a Electronic Flight Instrument System using a Freescale Power PC architecture processor and Metrowerks CodeWarrior development studio (C++) according to DO-178B Level B

- Designed automated white box (simulator) requirements-based test cases and procedures
- Analyzed and assessed structural coverage results to achieve decision coverage
- Maintained and supported testing environment (hardware\software) for a verification team of more than 10 engineers

4 months

Communication Navigation and Surveillance System

As Software Verification Engineer verified embedded software for a Communication Navigation and Surveillance System using a Power PC Architecture processor and Metrowerks CodeWarrior Development Studio (C++) according to DO-178B Level B

- Designed manual white box (emulator) and black box requirements-based test cases and procedures
- Analyzed and assessed structural coverage results to achieve decision coverage

Education

Post-Bachelor Degree in Electronics Engineering
Instituto Tecnológico de Costa Rica (Costa Rica Institute of Technology)
Degree certified by the Canadian Engineering Accreditation Board (CEAB)

J

Resultados del Proceso de Evaluación de Requerimientos

Pregunta	Grupo	Evaluador A						Evaluador B						Evaluador C						Resultados						Esperado					
		Impreciso	Ambiguo	No-verificable	Otro error	Correcto	Sin criterio	Impreciso	Ambiguo	No-verificable	Otro error	Correcto	Sin criterio	Impreciso	Ambiguo	No-verificable	Otro error	Correcto	Sin criterio	Impreciso	Ambiguo	No-verificable	Otro error	Correcto	Sin criterio	Impreciso	Ambiguo	No-verificable	Otro error	Correcto	Sin criterio
1	G1			1				1										1		1		1					3				
2	G1	1						1									1			2			1			3					
3	G1	1										1			1					1	1			1		3					
4	G1					1						1						1						3						3	
5	G1		1							1							1				1	1	1			3	3				
6	G1		1							1							1				1	1	1			3	3				
7	G1		1									1					1				1		1	1						3	
8	G1		1					1										1		1	1			1		1					
9	G1					1						1						1						3						3	
10	G1	1										1			1					1	1			1						3	
11	G1					1			1									1			1			2		3	3				
12	G1		1						1							1					2	1								3	
13	G1		1					1		1								1		1	1	1		1		3	3				
14	G1	1							1								1			1	1		1				3	3			
15	G1		1						1									1			2			1		3					
16	G1					1						1						1						3						3	
17	G1		1						1					1						1	2									3	
18	G1	1										1		1			1			2			1	1						3	
19	G1		1					1									1			1	1		1				3				
20	G1	1										1		1						2				1		3					
1	G2		1						1								1				2		1				3				
2	G2	1							1										1	1	1				1		3				
3	G2		1						1									1			2			1			3				
4	G2		1						1					1						1	2						3				
5	G2		1						1									1			2			1			3				
6	G2		1									1					1				1	3	1	1		3					
7	G2		1					1									1			1	1		1			3					
8	G2	1						1										1		2				1		3					
9	G2	1										1						1		1				2		3					
10	G2		1					1						1						2	1					3					
11	G2	1						1		1						1				2		2					3				
12	G2			1						1				1		1				1		3					3				
13	G2			1						1				1		1				1		3					3				
14	G2			1						1							1					2	1				3				
15	G2			1						1							1					2	1				3				
16	G2					1						1						1					1	2						3	
17	G2	1								1								1		1		1		1						3	
18	G2					1						1						1						3						3	
19	G2	1										1						1		1				2						3	
20	G2					1		1		1								1		1		1		2						3	
1	G3			1						1						1						3					3	3			
2	G3		1									1									1			1			3	3			
3	G3			1								1					1					1	1	1			3	3			
4	G3			1								1					1					1	1	1			3	3			
5	G3		1					1									1			1	1		1				3	3			
6	G3		1					1		1				1						2	1	1				3	3				
7	G3	1								1				1		1				2		2				3	3				
8	G3		1							1				1						1	1	1				3	3				
9	G3		1					1		1							1			1	1	1	1			3	3				
10	G3		1					1										1		1	1			1		3	3				
11	G3		1							1							1				1	1	1			3	3				
12	G3		1					1									1			1	1		1			3	3				
13	G3		1					1									1			1	1		1			3	3				
14	G3		1					1									1			1	1		1			3	3				
15	G3		1							1				1						1	1	1				3	3				
16	G3			1						1							1					2	1			3		3			
17	G3			1						1							1					2	1			3		3			
18	G3			1				1		1									1	1		2			1	3		3			
19	G3			1						1				1						1		2				3		3			
20	G3			1				1		1							1					2	1			3		3			
1	G4		1										1		1						2				1	3					
2	G4		1									1		1						1	1			1		3					
3	G4					1						1			1						1			2		3					
4	G4		1									1			1						2			1		3					
5	G4	1							1					1						2	1					3					
6	G4		1						1							1					2	1				3					
7	G4		1					1									1			1	1		1			3					
8	G4		1									1		1						1	1			1		3					
9	G4	1							1									1		1	1			1			3				
10	G4		1						1									1			2			1			3				
11	G4	1								1									1	1		1			1		3				
12	G4		1										1		1						2				1		3				
13	G4		1					1						1						2	1						3				
14	G4			1						1						1	1					3	1					3			
15	G4			1						1						1						2	1					3			
16	G4			1												1						1	1					3			
17	G4			1						1						1						2	1					3			
18	G4			1						1							1					3						3			
19	G4			1				1						1						2		1						3			
20	G4			1						1								1				2		1				3			

Fondo oscuro = indica que esa celda es la respuesta esperada

La marca “1” = indica la respuesta que eligió el evaluador

Fondo oscuro y fuente en negrita = indica que el evaluador escogió la opción de respuesta esperada.

Fondo oscuro y fuente normal = indica que el evaluador escogió la opción de respuesta que no era la esperada.

Fondo oscuro = indica que esa celda es la respuesta esperada.

Bibliografía

[1] Baker, C., C. Fillmore y J. Lowe: *The Berkeley FrameNet project*. En *Proceedings of the COLING-ACL*, Montreal, Canada., 1998.

[2] Berry, D., E. Kamsties y M. Krieger: *From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity*, Noviembre 2003.

[3] Board, IEEE Standards: *IEEE Standard Glossary of Software Engineering Terminology*, Setiembre 1990. Std 610.12-1990.

[4] Carroll, J., G. Satta y H. Bunt (editores): *New Developments in Parsing Technology*. Kluer Academic, 2004.

[5] CLAIR: *The Clair Library*. Sitio oficial del grupo CLAIR. URL `http://belobog.si.umich.edu/clair/clair/downloads.html` visitado el 12 de marzo de 2009.

[6] CPAN: *Comprehensive Perl Archive Network*. Sitio oficial del grupo CPAN.org. URL `http://cpan.uwinnipeg.ca/htdocs/faqs/cpan-search.html` visitado el 12 de marzo de 2009.

[7] Drazen, M., P. Berander, L. Damm, J. Eriksson, T. Gorschek, K. Henningsson, P. Jonsson, S. Kagstrom, F. Martensson, K. Ronkko y P. Tomaszewski: *Software quality attributes and trade-offs*, capítulo *Software Quality Models and Philosophies*. Blekinge Institute of Technology, Junio 2005.

[8] Firesmith, D.: *Specifying Good Requirements*. Journal of Object Technology, 2(4), Julio-Agosto 2003.

[9] Fuchs, N. y R. Schwitter: *Attempto Controlled Natural Language for Requirement Specifications*. En *Seventh ILPS 95 Workshop on Logic Programming Environments*, Portland, Oregon. USA., Diciembre 1995.

[10] Galin, D.: *Software Quality Assurance From theory to implementation*. Pearson Education Ltd., 2004.

[11] Gervasi, V. y B. Nuseibeh: *Lightweight Validation of Natural Language Requirements: a case study*. En *Fourth International Conference on Requirements Engineering (ICRE'00)*, página 140, 2000.

[12] Hooks, I.: *Writing Good Requirements*. En *Third International Symposium of the NCOSE*, volumen 2. International Council of Systems Engineering, 1993.

[13] Kipper, K.: *VerbNet: A broad-coverage, comprehensive verb lexicon*. Tesis de Doctorado, Computer and Information Science Dept. University of Pennsylvania, Philadelphia, PA, June 2005.

[14] Kipper, K., H. Dang y Palmer M.: *Class-based construction of a verb lexicon*. En *AAAI/IAAI*, páginas 691–696, 2000.

[15] Kipper, K., A. Korhonen, N. Ryant y M. Palmer: *Extending VerbNet with Novel Verb Classes*. En *5th international conference on Language Resources and Evaluation*, Genova, Italia, 2006.

[16] Korhonen, A. y T. Briscoe: *Extended Lexical-Semantic Classification of English Verbs*. En Moldovan, Dan y Roxana Girju (editores): *HLT-NAACL 2004: Workshop on Computational Lexical Semantics*, páginas 38–45, Boston, Massachusetts, USA, May 2 - May 7 2004.

[17] Lami, G., S. Gnesi, F. Fabbrini, M. Fusani y G. Trentanni: *An Automatic Tool for the Analysis of Natural Language Requirements*. Informe técnico, C.N.R. Information Science and Technology Institute, Pisa, Italia, Setiembre 2004.

[18] Leffingwell, D. y D. Widrig: *Managing Software Requirements*. Addison-Wesley, 2003.

[19] Levin, B.: *English Verb Classes and Alternation: A Preliminary Investigation*. The University of Chicago Press, 1993.

[20] Manning, C. y H. Schütze: *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, Mayo 1999.

[21] Marasco, J.: *The importance of testing software requirements*. Software Quality News, October 2007. URL: `http://searchsoftwarequality.techtarget.com/news/column/0,294698,sid92_gci1275907,00.html` visitado el 18 de Enero de 2009.

[22] Marcus, M., B. Santorini y M.A. Marcinkiewicz: *Building a large annotated corpus of English: The Penn Treebank*. Computational Linguistics, 19(2):313–330, 1993.

[23] McClosky, D., E. Charniak y M. Johnson: *Reranking and Self-Training for Parser Adaptation*. En *21st International Conference on Computational Linguistics*, páginas 337–344, Sydney, Julio 2006.

[24] Miller, G., R. Beckwith, C. Fellbaum, D. Gross y K. Miller: *Introduction to WordNet: An Online Lexical Database*. Informe técnico, Cognitive Science Laboratory, Princeton University, Revised August 1993.

[25] Nugues, P.M.: *An Introduction to Language Processing with Perl and Prolog*. Springer-Verlag, Alemania, 2006.

[26] Pereira, F. y S. Shieber: *Prolog and Natural-Language Analysis*. Microtome Publishing, 2002. URL `http://www.mtome.com/Publications/PNLA/prolog-digital.pdf` visitado el 18 de Enero de 2009.

[27] Pressman, R.: *Software Engineering A Practitioner's Approach*. McGraw-Hill, 2001.

[28] RAE: *"Ambiguo"*. Real Academia Española en línea, Enero 2009. URL `http://www.rae.es` visitado el 22 de Enero de 2009.

[29] RTCA-Inc.: *FAA Advisory Circular*. AC 20-115B, Enero 1993.

[30] RTCA/EUROCAE: *DO-178 Software Considerations for Airborne Systems and Equipment Certification*. RTCA, Inc., Washington D.C., U.S.A., Diciembre 1992.

[31] Shi, Lei: *A general purpose semantic parser using framenet and wordnet*. Master's thesis, University of North Texas, Mayo 2004.

[32] Wilson, W., L. Rosenberg y L. Hyatt: *Automated Analysis of Requirement Specifications*. En *Nineteeth International Conference on Software Engineering (ICSE-97)*, Boston, MA, Mayo 1997.