Universidad de las Ciencias Informáticas

Facultad 3

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Simulador del Transmisor de Shannon (STX): Herramienta Educativa para la asignatura Teleinformática

Autores

Maidelys Jaca Aviles

Julio Cesar Ocaña Bermúdez

Tutores

Ing. Yoan Antonio López Rodríguez

Ing. Katia Monjes Machado

La Habana, Junio 2014
"Año 56 de la Revolución"

"Cualquier hombre puede hacer ciencia, pero sólo un gran hombre puede escribirla"

Claude E. Shannon

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Maidelys Jaca Aviles

_____

Firma del Autor

Julio César Bermúdez Ocaña

_____

Firma del Autor

Ing. Yoan Antonio López

_____

Ing. Katia Monjes Machado

_____

Firmas Tutores

DATOS DE CONTACTO

Ing. Yoan Antonio López Rodríguez

Ingeniero en Ciencias Informáticas, graduado en Julio del 2008

Profesor Asistente

Dpto.: Soluciones Financieras y Aduanales

Centro: Informatización de Entidades

Facultad 3

Universidad de las Ciencias Informáticas (UCI)

Líneas investigativas en las que ha trabajado: Software de gestión y Software educativo

Ing. Katia Monjes Machado

Ingeniero en Telecomunicaciones

Profesor Asistente

Dpto.: Técnicas de Programación

Facultad 3

Universidad de las Ciencias Informáticas (UCI)

Líneas investigativas en las que ha trabajado: Gestión de redes y Software educativo

DEDICATORIA

La vida no tiene sentido alguno si uno no tiene un sueño al cual dedicar su esfuerzo. Sencillamente, este nuestro sueño por fin materializado, va dirigido con todo mi amor y cariño:

A mi mamita, Amada Luisa, por apoyarme siempre sin esperar nada a cambio, por su infinito amor, paciencia, dedicación y comprensión. Por soportar estos 5 largos años alejadas y aun así guiarme en los buenos y malos momentos. Por ayudarme y esperar con confianza a que este momento llegara.

A mis cositas chiquitas, Titi (Yadel), el bebé que hice mío por estos 5 años, a Nanda, Martuchy, Alainsito, Sheily, Vouly, Jonhy y la más peque Leah Valeria…

Maidelys (Jakyta)

A mis padres por ser ejemplo y guía en mi vida.

A mi hermanita por darme mucho apoyo y ser paciente conmigo porque de veras no soy fácil.

A mi abuela por estar pendiente a mí durante todos estos años.

A mi familia por su preocupación y cariño.

A todas las personas que de una manera u otra me han ayudado y han estado a mi lado en todo momento.

Julinho

AGRADECIMIENTOS

Primeramente agradecer a todos por estar aquí hoy presentes aquí conmigo dándome apoyo, de veras lo aprecio mucho porque sé que muchos de los que están aquí están enredados todavía y cogieron un poco de su tiempo para dedicármelo a mí.

Quisiera agradecer a la revolución por crear esta universidad y darme la oportunidad de estudiar en ella, en la cual he conocido a muchas personas tan buenas que jamás olvidare.

A todos los profesores que han ayudado en mi formación como ingeniero y persona, especialmente a aquellos con los que compartí momentos fuera de las aulas.

A mi tutor Yoan por darme la oportunidad de desarrollar esta tesis de la cual el creo el perfil.

A mis compañeros de aula que fueron muchos desde que entre en primer año, a Jeandy, Yandry, Eiler, Gabriel, Medinilla mi peluquero y a todos los demás que formaron parte de los grupos en los que estuve.

A los equipos de programación, que fueron muchas las competencias a las que fuimos, Gabi, Jorge Jesús, Jesús y Javier.

A la gente del Rubin Kazan que desde que estamos en segundo año hemos ganado todos los mundialitos de la facultad.

A mis amigos del politécnico que aunque algunos no estudiaron aquí en la UCI conmigo, siempre estuvieron al tanto de mi carrera, especialmente a Enrique, Miguel Antonio, Raidel.

A mis tíos y primos Dulce, Carmen, Luis, Julio, Javier, David que siempre estuvieron pendientes de mí.

A mi novia por ser paciente conmigo porque en estos días de trabajo continuo no le he podido dedicar todo el tiempo que ella se merece. Mi niña, te quiero mucho.

A mi abuela que la quiero mucho y desde que soy un niño siempre me ha cuidado mucho.

A mi hermana por apoyarme y estar aquí conmigo.

A mi compañera de tesis que fueron muchos los días que casi no dormimos terminando el documento y la aplicación.

Y por último a las dos personas que han estado conmigo en todo momento desde que nací, este logro es para ustedes, gracias mami y papi por darme tanto amor y cariño.

Julio C

AGRADECIMIENTOS

A mis padres por su confianza, apoyo, dedicación y paciencia. Por su amor y sacrificio incondicional.

A Pilar Aragón, mi abuelita del alma, que aunque sé que no tiene idea qué diablos estudié siempre se adelantaba y orgullosa decía: Mi nieta estudia en la UCI y va a ser ingeniera, hoy por fin, mimita, podemos afirmarlo.

A mis tutores en especial a Yoan, perdón al ing.Yoan, que no dudó en mudarnos para su casa hasta limar cada detallito de la tesis, que se volvió diseñador, poeta y loco.

A mi compañero de tesis que tanta lucha me dio y me sacó de quicio pero al fin salí de él…jeje

A Tahimi mi hermanita linda, la Dra. de la familia, que siempre me apoyó y aconsejó.

A mi primi Isita, mi ejemplo.

A mis 11 tías y compañía, en especial a Angela Emilia, Cuqui, Elisa, Estrella, Maida, Martha y Saray (ordenados alfabéticamente) y a mi familia que siempre estuvieron listas para brindarme toda su ayuda, ahora me toca regresar un poquito de todo lo inmenso que me han otorgado.

A mis amis que nunca me abandonaron en especial a Rosi por soportar cada una de mis malacrianzas, por ser mi despertador y mi cable a tierra.

A mi mimi Jenny, mi gordis, por hacerme reir y disfrutar de los pequeños y GRANDES detallitos lindos de la vida.

A Lisy y Ozky, mis loquillas, por su mal ejemplo arrastrándome a todas las fiestas. A Mariam, Baby y Rosana por sus consejos, a Yely, mi compañera del café. A Sary, la chica XP modo Office que tanto me ayudó con el doc.

A las chicas de mi apto, Jessy, Deby y a Juan por su apoyo moral.

A Yairon, mi putiti, mi pedacito de bichito enfermero, mi cocinerito improvisado, que preferiste sacrificar tu tiempo para que yo pudiera cumplir con el mío. Por tu paciencia, amor, respeto y comprensión, que me inspiraron a ser mejor persona, ahora puedo decir que esta tesis lleva mucho de ti, gracias por estar siempre a mi lado.

Gracias a mi súper grupazo, en especial a los mangotes y los vaqueros, con sus waperías y lazos convertían esos 90 minutos en muchos más divertidos, en especial a Randyloo, Renesinho y Andres.

AGRADECIMIENTOS

A Willy pecesito y Fabra con sus excelentes ppt haciendo historia y a Jose Potter por hacernos explotar la creatividad.

A Maurice, Zumetinha, Rosalina: esas personitas que fueron profes y ahora mis amigos. A otros tantos que de alguna u otra forma influyeron con sus lecciones y experiencias en mi formación profesional asentando las bases para lograr esta meta.

A los papis, vecinos y familia de mi novio, que en tiempo record se han ganado mi cariño y respeto, en especial a Hairam mi suegro que me enseño que ningún precio es demasiado alto para pagar el privilegio de ser dueño de mis propios sueños.

A las familias de mis amigas Marian, Rosi, Osky y Jenny, por sentirlas mias en estos 5 años.

A mis nuevos amis Pablito y Manolo, que junto a Yairon crearon una brigada de cocina y fregado para que la chica estresada por la tesis pudiese alimentarse.

En fin, con todo mi cariño y mi amor gracias a todas las personas que hicieron de todo en la vida para que yo pudiera lograr mis sueños, por motivarme y darme un timbrazo de aliento cuando sentía que el camino se terminaba, a ustedes por siempre mi corazón y mi agradecimiento.

Jakyta

RESUMEN

La siguiente investigación aborda el sistema Simulador del Transmisor de Shannon (STX), el cual permite representar los subsistemas funcionales concernientes al esquema ampliado del transmisor de Claude E. Shannon, visto a través de 5 subsistemas (Codificación, Compresión, Cifrado, Modulación y Multiplexión) y que en su conjunto contrarrestan las dificultades a las que se expone la información antes de ser recibida por el receptor.

La explicación de este contenido es objetivo de Teleinformática, asignatura impartida en la Universidad de las Ciencias Informáticas (UCI), por lo que los profesores deben lograr que los estudiantes asimilen correctamente dicho conocimiento, sin embargo, el tiempo para su impartición y apropiación es limitado y se carecen de medios para esclarecer los procesos que ocurren en cada subsistema.

El sistema STX se presenta como una herramienta educativa de apoyo a la asignatura Teleinformática, que facilita la caracterización de los algoritmos de cada uno de los subsistemas del transmisor, utilizando como guía los seudocódigos definidos en el libro “Data Compression” en su cuarta edición, bibliografía básica de la disciplina. Para su desarrollo se realizó un análisis de las principales metodologías y tecnologías utilizadas para este tipo de herramienta, a partir del cual se seleccionó la plataforma Java con su IDE Netbeansv7.3, como metodología de desarrollo Extreme Programming, Visual Paradigm como herramienta CASE y UML como lenguaje de modelado.

PALABRAS CLAVES

Codificación, compresión, educativa, herramienta, transmisor.

ÍNDICE

ÍNDICE ..... **VIII**

INTRODUCCIÓN **1**

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA **6**

1.1 Introducción 6
1.2 Sistema de Transmisión de Datos 6
1.2.1 Codificación 8
1.2.1.1 Codificación eficiente 9
1.2.1.1.1 Shannon-Fano 9
1.2.1.1.2 Huffman 11
1.2.1.2 Codificación redundante 12
1.2.1.2.1 Hamming 13
1.2.1.2.2 Códigos Cíclicos 13
1.2.2 Cifrado 14
1.2.3 Compresión 15
1.2.3.1 Compresión con pérdidas 15
1.2.3.2 Compresión sin pérdidas 16
1.2.3.2.1 Run Length Encoding (RLE) 16
1.2.3.2.2 LZ77 16
1.2.3.2.3 LZ78 18
1.2.3.2.4 LZW 19
1.2.3.2.5 Aritmético 21
1.2.4 Multiplexión 23
1.2.5 Modulación 23
1.2.6 Resultado del Análisis 24
1.3 Herramientas para la simulación de la transmisión de datos 24
1.3.1 Matlab 25
1.3.2 VirtualNet 25

1.3.3 Resultado del Análisis.....26
1.4 Tecnologías y herramientas.....27
1.4.1 Metodología de Desarrollo.....27
1.4.1.1 Programación Extrema (XP).....29
1.4.1.2 SCRUM.....31
1.4.1.3 Metodología seleccionada.....32
1.4.2 Lenguaje Unificado de Modelado (UML).....32
1.4.3 Lenguajes de programación.....33
1.4.3.1 Java.....34
1.4.3.2 C++.....35
1.4.4 Entorno de Desarrollo Integrado.....36
1.4.5 Modelado de Negocio.....37
1.4.6 Resultado del análisis.....38
1.5 Conclusiones parciales.....38
CAPÍTULO 2: PROPUESTA DE SOLUCIÓN.....**39**

2.1 Introducción.....39
2.2 Solución propuesta.....39
2.3 Personas relacionadas con el sistema.....40
2.4 Planificación.....40
2.4.1 Requerimientos de software.....40
2.4.2 Técnicas de obtención de requerimientos de software.....40
2.4.3 Técnicas de validación de requerimientos de software.....41
2.4.4 Resultado del análisis.....42
2.4.5 Requisitos funcionales.....42
2.4.6 Requisitos no funcionales.....44
2.4.7 Historias de usuario.....45
2.4.8 Estimación de esfuerzos por Historia de usuarios.....47
2.4.9 Plan de iteraciones.....49
2.4.10 Plan de entregas.....50
2.5 Diseño de la solución.....52

2.5.1 Tarjetas CRC (Clase, Responsabilidad y Colaboración) 52
2.5.2 Estándares de codificación 53
2.5.3 Estándares en la interfaz de la aplicación. 54
2.6 Validación del diseño 55
2.6.1 Tamaño Operacional de Clase (TOC) 55
2.6.2 Métrica Relaciones entre Clases (RC) 56
2.7 Conclusiones del capítulo 58
CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS **59**

3.1 Introducción 59
3.2 Prueba 59
3.3 Métodos de prueba 59
3.3.1 Tipos de pruebas: 60
3.4 Diseño y ejecución de las pruebas de software 60
3.5 Conclusiones del capítulo 64
CONCLUSIONES GENERALES **65**

RECOMENDACIONES **66**

ÍNDICE DE FIGURAS

Figura 1. Sistema de transmisión de datos. 7
Figura 2. Proceso de Codificación mediante el método de Shannon-Fano. Extraída del libro Data Compression, 4ta Edición. 10
Figura 3. Proceso de Codificación mediante el método de Huffman. 12
Figura 4. Representación polinómica de secuencia. 14
Figura 5. Funcionamiento del algoritmo LZ77. 17
Figura 6. Búsqueda de coincidencias para algoritmo LZ77. 18
Figura 7. Secuencia 19
Figura 8. Seudocódigo algoritmo LZW. 20
Figura 9 Resultados TOC. 56
Figura 10. Resultados RC. 58
Figura 11. Método run(). 61
Figura 12 A la izquierda la clase Modulo_CodificacionTest y a la derecha Modulo_CompresionTest 63
Figura 13. Resultados de la generación de casos de prueba 64

ÍNDICE DE TABLAS

Tabla 1. Diccionario inicial LZ78 19
Tabla 2. Procedimientos algoritmo LZ78 19
Tabla 3. Diccionario de código LZW. 21
Tabla 4. Compresión aritmética 22
Tabla 5. Comparación entre metodología ágil y metodología tradicional. 28
Tabla 6. Comparación de las metodologías 32
Tabla 7. Requisitos no funcionales del software 45
Tabla 8. Modelo de Historias de Usuarios 47
Tabla 9. Historias de Usuario Codificación Shannon Fano. 47
Tabla 10. Estimación de esfuerzos 48
Tabla 11. Plan de duración de iteraciones 49
Tabla 12. Plan de Entregas 51
Tabla 13. Modelo de Tarjetas CRC 52
Tabla 14. Tarjeta CRC AlgoritmoCodificacion.java 53
Tabla 15. Tarjeta CRC InputStringCodif.java 53
Tabla 16. Tarjeta CRC de la clase CRC.java 53
Tabla 17. Métrica TOC ..... **¡Error! Marcador no definido.**
Tabla 18. Rango de valores para la evaluación técnica de los atributos de calidad relacionados con la métrica 56
Tabla 19. Métricas RC. 57
Tabla 20. Rango de valores para la evaluación técnica de los atributos de calidad relacionados con la métrica 57

INTRODUCCIÓN

La sociedad ha manifestado cambios con respecto a la educación, lo cual se aprecia en las sensibles mejorías constatadas en el proceso enseñanza - aprendizaje a partir del uso de nuevas herramientas didácticas. En este sentido, es favorable el aporte de las Tecnologías de la Información y las Comunicaciones (TIC), cuyo uso deviene en la actualidad en una de las vías principales para lograr una mayor interacción entre estudiantes y profesores; mediante la integración de dichas herramientas a los procesos de enseñanza-aprendizaje se facilita y enriquece el trabajo de los docentes. El desarrollo continuo que se percibe en la accesibilidad, almacenamiento y utilización de la información propician la aparición de nuevos medios didácticos con tecnologías y multimedia para transmitir el conocimiento.

Cuba no está exenta de las transformaciones substanciales que originan en la contemporaneidad los avances en las TIC. En aras de lograr la independencia tecnológica se crea un centro de altos estudios llamado Universidad de las Ciencias Informáticas (UCI), que tiene entre sus objetivos la formación de profesionales que contribuyan a la producción de aplicaciones y servicios informáticos, a partir de la vinculación estudio-trabajo. Las altas exigencias que encierra en ejercicio de esta profesión merecen de un constante perfeccionamiento de los medios y herramientas educativas que se emplean en la impartición de las materias académicas. Enfrascados en esta meta es preciso desarrollar habilidades en los educadores que faciliten la transmisión de conocimientos, así como de una alta capacidad de auto aprendizaje en el estudiantado a través del uso de medios didácticos que los apoyen.

Muchas de las asignaturas del plan curricular demandan el uso de dichos adelantos. Teleinformática es una de ellas, basada principalmente en el estudio de modelos comunicativos de la teoría de la información en el que se encuentra el modelo de Claude E. Shannon. Este modelo comunicativo encuentra su impulso principal en algunos trabajos de ingeniería de las telecomunicaciones. Estudios realizados en la primera mitad del siglo XX impulsaron dicho modelo, sustentado en investigaciones acerca de la velocidad de transmisión de los mensajes telegráficos desarrollado por Harry Nyquist Eriksdotter en 1924 y trabajos sobre la medida de la cantidad de información realizados cuatro años más tarde por Ralph Vinton L. Hartley. En 1948 es publicado por Shannon como parte de un artículo, en dos números de la revista The Bell System Technical Journal y editado en 1949 bajo el título The Mathematical Theory of Communication

El esquema del modelo de Shannon "ilustra el hecho de que en cada proceso comunicativo existe siempre una fuente o manantial de la información, desde la cual, a través de un aparato transmisor, es emitida una señal; esta señal viaja a través de un canal a lo largo del cual puede ser interferida por un ruido. Al salir del canal, la señal es recogida por un receptor que la convierte en un mensaje. Como tal, el mensaje es comprendido por el destinatario". (Wolf, 2005).

El transmisor del esquema de Shannon en su forma ampliada es visto a través de 5 subsistemas (Codificación, Compresión, Cifrado, Modulación y Multiplexión) que en su conjunto contrarrestan las dificultades del medio de transmisión. La explicación de este contenido es uno de los objetivos de la asignatura, por lo que los profesores deben lograr que los estudiantes asimilen correctamente dicho conocimiento. Este proceso de enseñanza resulta difícil debido a que el tiempo para su impartición y apropiación es limitado y además no se dispone de recursos que faciliten la visualización, comprensión e interpretación de los procesos que ocurren en cada subsistema.

En la UCI se hace necesario crear herramientas educativas que apoyen el proceso de enseñanza-aprendizaje de una manera creativa y ágil, la impartición del Esquema Ampliado del Transmisor de Shannon, como parte de la asignatura Teleinformática, podría valerse en gran medida del uso de esas herramientas.

A partir de los aspectos expuestos anteriormente sobre las limitaciones que presenta la asignatura Teleinformática y la asimilación de este contenido se esboza como **problema a resolver:** No se cuenta con herramientas educativas de apoyo, en el proceso de enseñanza-aprendizaje del Esquema Ampliado del Transmisor de Shannon, como parte de la asignatura Teleinformática.

Se define como **objeto de estudio:** Herramientas de apoyo a la docencia y como **campo de acción:** Herramientas de apoyo a la docencia en la asignatura Teleinformática.

Se tiene como **objetivo general:** Desarrollar una herramienta educativa que apoye en el proceso de enseñanza del Esquema Ampliado del Transmisor de Shannon en la asignatura Teleinformática.

Para alcanzar el objetivo general se deben cumplir los siguientes **objetivos específicos:**

1. Fundamentar la investigación mediante la elaboración del Marco Teórico que sustenta los

conceptos y la propuesta de desarrollo de la herramienta educativa Simulador del Transmisor de Shannon (STX).

2. Realizar el análisis y diseño de la herramienta educativa Simulador del Transmisor de Shannon (STX).
3. Implementar la herramienta educativa Simulador del Transmisor de Shannon (STX).
4. Validar la solución propuesta para verificar la calidad del sistema implementado.

Como **idea a defender** se plantea que, si se desarrolla el sistema STX entonces se contará con una herramienta educativa de apoyo al proceso de enseñanza-aprendizaje del Esquema Ampliado del Transmisor de Shannon como parte de la asignatura Teleinformática.

Para dar cumplimiento a los objetivos específicos se proponen las siguientes **tareas de investigación:**

1. Revisión de bibliografía actualizada para la elaboración del marco teórico-conceptual.
2. Generación del marco teórico-conceptual a partir del análisis de herramientas educativas de apoyo a la docencia que simulan la transmisión de datos y del sistema STX.
3. Estudio y selección de la metodología, las herramientas, los lenguajes y las tecnologías a utilizar en el desarrollo de la aplicación.
4. Definición de los requerimientos funcionales y no funcionales para su desarrollo.
5. Realización del diseño del sistema STX.
6. Validación del diseño de la solución propuesta.
7. Implementación de las funcionalidades diseñadas.
8. Realización de las pruebas necesarias que orienta la metodología para comprobar que el sistema STX cumpla con todos los requerimientos especificados.

Para dar cumplimiento a las tareas anteriores se aplican los métodos de investigación científicos que a continuación se mencionan:

Del nivel teórico se selecciona:

Análisis Histórico-Lógico: permitió comprender de forma más clara la esencia del objeto de estudio, su concepción histórica y los fenómenos que lo condicionaron. Para ello se realizó un análisis de herramientas educativas de apoyo a la docencia que simulan la transmisión de datos y del sistema de transmisión de Shannon; y una investigación acerca de las diferentes metodologías de desarrollo. Además

de las tecnologías y herramientas usadas para la realización de herramientas educativas, así como las tendencias existentes en Cuba y en el mundo.

Analítico-sintético: permitió identificar y analizar la metodología de desarrollo de software a utilizar adecuándose a las particularidades del problema a resolver y del propósito del producto final. Además posibilitó la descomposición del objeto de estudio en conceptos más pequeños y más fáciles de estudiar, así como conocer sus características generales. Para ello, se estudió la teoría y bibliografía relacionada con el problema. Una vez concluido el análisis, se logró un conocimiento concreto sobre herramientas educativas y su necesidad, se formularon conceptos y definiciones fundamentales relacionados con el tema y se ofreció una propuesta de solución.

Inductivo-deductivo: permitió arribar a conclusiones particulares para la selección de las herramientas y la metodología a utilizar en el desarrollo del sistema.

Del nivel empírico se selecciona:

Análisis documental: permitió mediante su uso la revisión bibliográfica, el estudio de documentos propios de la universidad y de la asignatura en particular, además del análisis de los sistemas de tele-formación en la UCI y el modelo de formación centrado en el aprendizaje que se aplica en la UCI.

Observación: permitió identificar el problema a resolver formulado.

Entrevista: Puede ser estructurada o no estructurada. Para la realización de esta investigación se utilizó la entrevista no estructurada que es más abierta que la estructurada, prevé el tema pero no lleva un cuestionario rígido y puede variar de una persona a otra, es más flexible. Se aplica a especialistas en el tema, es una forma de obtener criterios de expertos. La utilización de la misma permitió obtener información sobre la metodología y herramientas utilizadas para el desarrollo de aplicaciones similares.

El presente trabajo está estructurado de la siguiente manera:

❖ **Capítulo 1. Fundamentación teórica:** En este capítulo se presentan elementos teórico-conceptuales vinculados con la problemática planteada. Se argumenta acerca de algunas funcionalidades de varias herramientas educativas que se comportan como simuladores de transmisión de datos, analizando en cada caso la información que ofrecen. Se hace un estudio de las principales herramientas y

tecnologías para la implementación de la aplicación, así como de las diferentes metodologías de desarrollo de software más utilizadas en la actualidad.

- **Capítulo 2. Propuesta de solución:** En este capítulo se describe la propuesta de solución. Contiene elementos significativos como los requerimientos funcionales y no funcionales que debe cumplir el sistema, así como el diseño del mismo. Además, contiene los artefactos que propone la metodología.

- **Capítulo 3. Implementación y pruebas:** Este capítulo se centra en los procesos de implementación y pruebas de las funcionalidades del sistema desarrollado. Contiene los casos de prueba y otras técnicas de prueba utilizadas para validar las funcionalidades desarrolladas.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

Durante el proceso evolutivo de las comunidades humanas, el hombre convive en sociedades, asociaciones tribales y grupos étnicos, por lo que necesitó de la comunicación por distintas vías para transmitir sus necesidades. Así, desde los inicios de la especie, surgieron los primeros intentos de transmisión de información a largas distancias mediante la utilización de sus brazos, para emitir gestos y otros métodos como señales de humo, aves, entre otras, hasta alcanzar el punto cumbre de la evolución tecnológica: las telecomunicaciones, que nace para acercar espacios y tener mayor velocidad en el proceso.

Para alcanzar un entendimiento y desarrollo óptimo del tema, es preciso trabajar desde su génesis, partiendo de materias que se dedican a su estudio, conceptos, técnicas, procedimientos y herramientas con el objetivo de aplicar estas teorías para el diseño de sistemas de comunicación, de modo que la información presente el mínimo deterioro cuando se transmite a través de un medio particular.
La Teleinformática, materia que pertenece al plan de estudio de la UCI, sujeta un conjunto de técnicas, procedimientos, normas y teorías con las que se hace posible la transmisión de información a distancias arbitrarias. Los procesos de comunicación a los que hace referencia, son aquellos en los que de alguna manera, se hace necesario el uso de dispositivos programables para el procesamiento de datos, de modo que cualquier situación en la que exista la necesidad de una comunicación “inteligente” entre dos o más dispositivos, podrá ser abordada, estudiada y resuelta en términos de Teleinformática. Estos procesos de comunicación se pueden representar claramente por medio de un sistema de transmisión de datos y con el estudio del mismo se hace imprescindible el análisis de sus componentes esenciales y de sus limitaciones en el proceso de transmisión, así como la comprensión de los conceptos básicos de la Teoría de la Información.

1.2 Sistema de Transmisión de Datos

La transmisión de la información es la operación de enviar y recibir datos, mediante el empleo de

determinados procesos para eliminar problemas de información redundante (ruido), erradicar riesgos de intercepción o modificación de un mensaje, corregir errores que posea la información y otros eventos que podrían provocar que el proceso no se ejecute de forma efectiva y eficiente.

A continuación, en la Figura 1, se muestra un sistema de transmisión de datos.

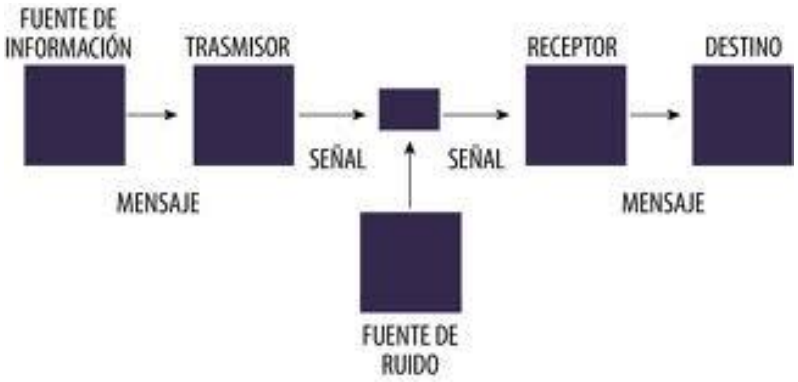

Figura 1. Sistema de transmisión de datos.

El esquema del sistema de transmisión de datos está compuesto por cinco elementos: una fuente de información, un transmisor, un canal que puede ser perturbado por una fuente de ruido, un receptor y un destino.

Los elementos se explican a continuación:

- Fuente de información: es el elemento inicial del proceso de comunicación que produce un cierto número de palabras o signos que forman el mensaje a transmitir.
- Transmisor: es el emisor técnico encargado de transformar el mensaje emitido en un conjunto de señales o códigos que serán adecuados al canal encargado de transmitirlos.
- Canal: es el medio técnico que transporta las señales codificadas por el transmisor.
- Receptor: es el receptor técnico, cuya actividad es la inversa del transmisor.
- Destinatario: constituye el verdadero receptor a quien está destinado el mensaje.

Para contrarrestar los problemas de ruido que afectan los mensajes enviados por la fuente, el transmisor visto desde su esquema ampliado, lleva a cabo un grupo de procesos que permiten que los sistemas actuales sean más seguros, eficientes y rápidos a la hora de establecer la comunicación entre un emisor y un receptor. Tales procesos realizados en el transmisor son: codificación, cifrado, compresión,

multiplexación y modulación para luego realizar los procesos inversos como decodificar, descifrar, descomprimir, demultiplexar y desmodular para que llegue a su destino sin que haya sufrido ningún cambio o alteración.

A continuación se presenta cada uno de estos procedimientos, haciendo énfasis en los que, rigiéndose por el plan de estudio de la asignatura Teleinformática, se orienta sean los que los estudiantes deben dominar y por ende conciernen el desarrollo del sistema STX.

1.2.1 Codificación

La codificación consiste en establecer una correspondencia entre cada uno de los símbolos de un alfabeto fuente y una secuencia de símbolos de un alfabeto destino. Al alfabeto destino se le denomina alfabeto código y a cada una de las secuencias de símbolos de este alfabeto que se corresponda con un símbolo del alfabeto fuente se denomina palabra de código. El alfabeto fuente contiene los símbolos originales que se quieren codificar. El alfabeto código contiene las palabras de código equivalentes en que se codificarán los símbolos originales. (Según se plantea en: U.d Conceptos generales sobre codificación, Valladolid).

El objetivo de la codificación es obtener una representación eficiente de los símbolos del alfabeto fuente, en la cual, para que sea eficiente es necesario tener un conocimiento de las probabilidades de cada uno de los símbolos del alfabeto. El dispositivo que realiza esta tarea es el codificador, este debe cumplir el requisito de que cada palabra de código debe de codificarse de forma única, tal que la secuencia original sea reconstruida perfectamente a partir de la secuencia codificada. Concentrados en el propósito de que en este proceso se garantice la fiabilidad, es necesario que, los códigos generados cumplan una serie de condiciones.

Las condiciones que se deben respetar son:

- **No singularidad:** consiste en que a cada mensaje se le asigna una secuencia distinta de símbolos del código. (Ejemplo m1-00, m2-01, m3-10).
- **Separable:** Únicamente decodificable. No basta que las diferentes palabras del código $X_{i,}$ sean diferentes (no singulares), sino que las secuencias de palabras del código: X_i, X_{i2}, X_{i3}… se corresponda con una única secuencia de símbolos. (Ejemplo m1-0, m2-10, m3-110).
- **Instantáneos:** Debe ser no singular y separable. Cumple con la condición de los prefijos (no existe

palabra que sea prefijo de otra de longitud mayor). (Ejemplo m1-0, m2-10, m3-110).

Una correcta codificación permite mayor rapidez en la comunicación mediante códigos más cortos y con el mismo nivel de información que el mensaje.

La codificación es el primer subsistema del transmisor y para su correcta asimilación es preciso conocer términos generales de la teoría de la información tales como: cantidad de información[1], fuente[2], entropía[3], dificultades del medio[4], llegando al entendimiento por parte de los estudiantes de la importancia de una adecuada codificación tanto eficiente como redundante. La codificación redundante surge debido a las modificaciones que sufre la información al viajar por el canal de transmisión.

1.2.1.1 Codificación eficiente

La codificación eficiente se basa en asignar las palabras de código más cortas a los mensajes más probables de la fuente. Puede presentar algunas deficiencias especificadas a continuación:

- Se genera una dificultad práctica el tratar con códigos de longitud variable.
- Se acumulan símbolos a la entrada del codificador.
- Las perturbaciones pueden alterar la longitud de una palabra de código.

Los algoritmos estudiados para realizar la codificación eficiente son Shannon-Fano y Huffman.

1.2.1.1.1 Shannon-Fano

La codificación Shannon-Fano fue la primera técnica de compresión desarrollada para obtener códigos de prefijo. La técnica fue propuesta por Claude Elwood Shannon, en “Una Teoría Matemática de la Comunicación”, su artículo de 1948 introduciendo el campo de la teoría de la información.

Pasos para codificar por el método de Shannon Fano teniendo S símbolos. S= {S1, S2……, Sn}.

1. Los mensajes se colocan en una tabla por orden de probabilidades de forma decreciente.

[1]medida de la libre elección de un mensaje, no referida a los mensajes individuales, sino a la situación en su totalidad.
[2]fuente de información y transmisor (en la emisión), siendo la fuente la capacidad del sistema para generar señales y el transmisor el instrumento (tecnológico) de que se sirve.(Serrano, 2005)
[3]medida de desorganización, la información suministrada por un grupo de mensajes es una medida de organización"(Saladrigas, 2005). Entonces a mayor desorden o entropía, mayor es la cantidad de información necesaria para recuperar un mensaje.
[4]efectos indeseables que debe enfrentar el sistema de transmisión.

2. Se divide la tabla en **r** grupos (**r** es la base del código elegido para codificar) de forma tal que la suma de las probabilidades, en cada grupo sea casi igual.
3. Al primer grupo superior de la lista se le asigna el primer dígito del alfabeto de código que se utilice; para el caso del alfabeto binario (**r=2**), los del primer subconjunto recibirán el "0" y los del segundo el "1" o viceversa, que equivale a la cantidad de dígitos con que cuente el alfabeto.
4. Recursivamente se aplica el mismo procedimiento a cada uno de los grupos.

A continuación, en la Figura 2 se muestra el resultado obtenido en el proceso de codificación mediante el método de Shannon-Fano.

	Probabilidad	Pasos				Final
1	0,25	11				:11
2	0,2	10				:10
3	0,15	0	11			:011
4	0,15	0	10			:010
5	0,1	0	0	1		:001
6	0,1	0	0	0	1	:0001
7	0,05	0	0	0	0	:0000

Figura 2. Proceso de Codificación mediante el método de Shannon-Fano. Extraída del libro Data Compression, 4ta Edición.

Entre mejor sea la división realizada entre los símbolos, mejores resultados se obtendrán y esto se logra cuando la división es lo más pareja posible. Cuando los dos subconjuntos iniciales tienen la misma proporción se produce un código con el mínimo tamaño promedio alcanzando una redundancia de cero. Este método produce mejores resultados cuando el símbolo tiene probabilidades de ocurrencia que son potencias de dos.

No es óptimo en el sentido de que no consigue la menor longitud de palabra de código esperada posible como en la codificación Huffman.

1.2.1.1.2 Huffman

Con el Método de Huffman, su autor solucionó la mayor parte de los errores presentes en el algoritmo de codificación Shannon-Fano. La solución se basa en el proceso de construcción del árbol de abajo hacia arriba y no de arriba hacia abajo, con el alfabeto de código con que se cuente.

La técnica utilizada es el propio algoritmo de Huffman, el cual consiste en la creación de un árbol en el que se etiquetan los nodos hoja con los caracteres junto a sus frecuencias y, de forma consecutiva, se van uniendo en parejas los nodos que menos frecuencia sumen, pasando a crear un nuevo nodo intermedio etiquetado con dicha suma.

Se procede a realizar esta acción hasta que no queden nodos hoja por unir a ningún nodo superior y hasta que se haya formado el árbol. La cantidad de nodos hijos de cada nodo superior está determinada por el alfabeto de código que se utilice; para el alfabeto de código binario la cantidad de nodos inferiores por nodos superiores es 2 (0 y 1). Utilizando este alfabeto posteriormente, se etiquetan las aristas que unen cada uno de los nodos con ceros y unos (0 a la izquierda y 1 a la derecha). El código resultante para cada carácter es la lectura, siguiendo la rama, desde la raíz hacia cada carácter (o viceversa) de cada una de las etiquetas de las aristas. (García, 2010).

Las probabilidades usadas pueden ser genéricas para el dominio de la aplicación, las cuales están basadas en el caso promedio, o bien, pueden ser las frecuencias reales encontradas en el texto que se está comprimiendo. (Esta variación requiere que la Tabla de frecuencias u otra estructura utilizada para la codificación sean almacenadas con el texto comprimido; las implementaciones emplean varios mecanismos para almacenar tablas de manera eficiente).

La codificación Huffman es óptima cuando la probabilidad de cada símbolo de entrada es una potencia negativa de dos. Los códigos prefijos tienden a ser ligeramente ineficientes en alfabetos pequeños, donde las probabilidades normalmente se encuentran entre esos puntos óptimos.

Huffman es considerado el algoritmo de mayor eficiencia. El resto de los códigos pueden igualarlo en determinados casos, pero nunca superarlo, por las siguientes características.

- ❖ No hay dos mensajes con iguales secuencias.

- ❖ Cumple la condición de los prefijos.
- ❖ Los símbolos más probables son codificados con palabras de código de menor longitud.
- ❖ Los dos mensajes menos probables son codificados con palabras de igual longitud de código.
- ❖ Los mensajes del alfabeto se colocan en una columna en orden decreciente de las probabilidades.
- ❖ Los dos últimos mensajes se unen en uno auxiliar al que se le atribuye la probabilidad suma.
- ❖ Las probabilidades de los mensajes que no han participado en la unión así como la probabilidad suma obtenida, se ordenan otra vez en forma descendente en una columna adicional.
- ❖ Los pasos anteriores se repiten hasta obtener un único mensaje auxiliar con probabilidad igual a la unidad.

A continuación, en la Figura.3, se muestra el resultado obtenido en el proceso de codificación mediante el método de Huffman.

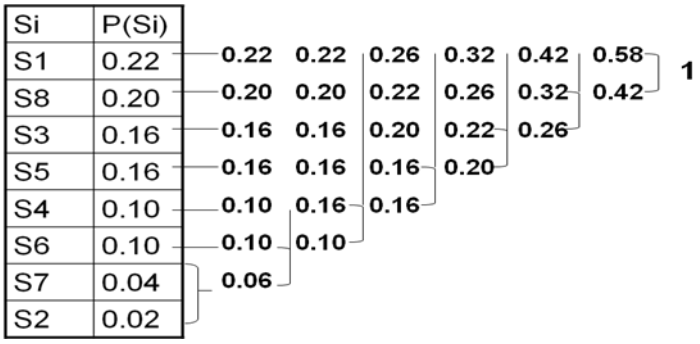

Figura 3. Proceso de Codificación mediante el método de Huffman.

El algoritmo Huffman constituye el algoritmo de codificación más eficiente y más usado. Es bastante simple desde el punto de vista computacional y produce códigos que siempre consiguen la menor longitud esperada de palabra de código.

1.2.1.2 Codificación redundante

Se basa en asignar a los mensajes una codificación que no consiste en una única elección entre posibilidades de igual probabilidad, sino en elecciones sucesivas entre posibilidades de probabilidad variable y dependiente. (López y otros, 1995). Mediante este método se agregan a los bits de información que se desean transmitir, unos bits de control, que ayudará a determinar si la información llega con o sin errores.

La integración de código redundante permite realizar la corrección en cierta medida de los errores presentados en la transmisión; sin embargo hace menos eficiente el proceso de codificación, por lo cual se deberá lograr un equilibrio entre codificación redundante y eficiente dadas las características del canal. Como algoritmos de codificación redundante se representan en el sistema: Hamming y Códigos Cíclicos.

1.2.1.2.1 Hamming

El código de Hamming es un sistema de detección y corrección automática de errores en información electrónica que lleva el nombre de su inventor, Richard Hamming. Este código asocia una serie de bits de validación o paridad a los bits de datos de tal forma que una alteración en cualquiera de esos bits de datos; pueda ser detectada y corregida adecuadamente. En los datos codificados en Hamming se pueden detectar errores en un bit y corregirlos, sin embargo no se distingue entre errores de dos bits y de un bit, y para este proceso de identificación se usa Hamming extendido. Esto representa una mejora respecto a los códigos con bit de paridad, que pueden detectar errores en sólo un bit, pero no pueden corregirlo.

1.2.1.2.2 Códigos Cíclicos

El uso de los códigos cíclicos también llamados Códigos de Redundancia Cíclica (CRC) o códigos polinómicos, está muy extendido porque pueden implementarse en hardware con mucha facilidad. Un código cíclico es un código de bloque lineal donde si c es una palabra de código, también lo son todos los desplazamientos cíclicos de c. Ejemplo: {000, 110, 101, 011}. Estos códigos se basan en el uso de un polinomio generador G(X) de grado: b= n – k donde:

- n: Bits Totales n = k + b.
- k: Bits de información.
- b: Bits de Chequeo.

A continuación, en la Figura 4, se muestra la representación polinómica de una secuencia de bits.

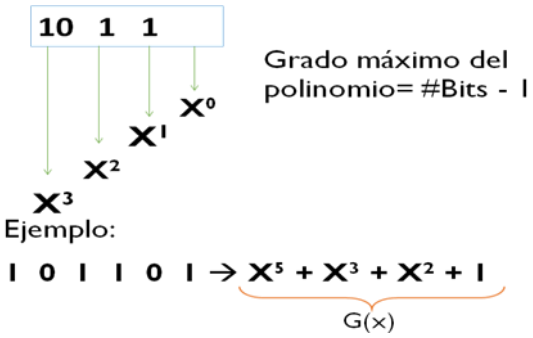

Figura 4. Representación polinómica de secuencia.

Rendimiento del código cíclico:

- Si P(x) es distinto del polinomio 1, entonces detecta todos los errores simples.
- Si P(x) es múltiplo de (x+1), entonces detecta todos los errores impares.
- Si P(x) tiene como factor a un polinomio primitivo, entonces puede detectar todos los errores dobles.

1.2.2 Cifrado

El cifrado es el método siguiente a la codificación que consiste en la transformación del contenido de la información, mensaje o archivo en un texto totalmente distinto sin ningún significado llamado criptograma mediante la utilización de algoritmos matemáticos. Este proceso trabaja directamente en base a apoyar la seguridad, garantizando la confidencialidad y la integridad de la información, generando una clave de seguridad. (Aguirre, 2005).

Los algoritmos de encriptación son:

Sustitución: técnica que establece una correspondencia con los símbolos del alfabeto del mensaje original con otros, que pueden ser del mismo o de uno distinto. (López, 1999).

Transposición: técnica que consiste en intercambiar los símbolos del mensaje original colocándolos de maneras diferentes. (López, 1999).

Existen varios tipos de cifrado que se citan a continuación:

Simétrico: Cuando se emplea la misma clave en las operaciones de cifrado y descifrado, utilizando algoritmos como IDEA, RC5, DES, TRIPLE DES, entre otros. (Aguirre, 2005).

Asimétrico: Cuando cada usuario crea un par de claves, una privada y otra pública, inversas dentro de un

cuerpo finito. Lo que se cifra en emisión con una clave, se descifra en recepción con la clave inversa. La seguridad del sistema reside en la dificultad computacional de descubrir la clave privada a partir de la pública. Para ello, usan funciones matemáticas de un solo sentido o con trampa, como RSA, Diffie-Hellman, etc. (Aguirre, 2005).

Luego del cifrado se pasa a la compresión que se encarga de reducir el tamaño de la información y transmitirla en menor tiempo, siendo menos la información que se envía.

1.2.3 Compresión

El proceso de compresión de la información consiste básicamente en un conjunto de técnicas que permiten transformar una secuencia de símbolos fuente en códigos claves, los cuales presentarán una longitud media de palabra menor con respecto a los símbolos originales. Este proceso se realiza sin alterar el significado de la información que contiene y a la vez para superar las deficiencias físicas de la red y de los equipos de conexión. Básicamente, consiste en reducir el peso de un archivo digital a partir de una reducción del volumen de los datos tratables, es decir que tras la compresión, ocupará menos espacio el archivo en cuestión. Se trata de eliminar redundancia en los archivos a transmitir.

Tipos de información que transmiten los datos:

- **Redundante:** información repetitiva o predecible.
- **Irrelevante:** información que no podemos apreciar y cuya eliminación por tanto no afecta al contenido del mensaje.
- **Básica:** la relevante. La que no es ni redundante ni irrelevante. La que debe ser transmitida para que se pueda reconstruir la señal.

La tipología de compresión, por lo tanto, puede ser sin pérdidas reales, subjetivamente sin pérdidas o subjetivamente con pérdidas resumidas en dos familias:

- Técnicas de compresión sin pérdidas.
- Técnicas de compresión con pérdidas.

1.2.3.1 Compresión con pérdidas

Las técnicas de compresión con pérdida, son utilizadas en los casos en que se permite cierta pérdida de

información, asegurando a la vez que se pueda recuperar la mayor parte de la información procesada.

La comprensión con pérdidas se emplea en la compresión de imágenes y videos. (Briceño M, 2005).

1.2.3.2 Compresión sin pérdidas

Las técnicas de compresión sin pérdida, son utilizadas cuando es imprescindible que se recupere toda la información, como es el caso de las cuentas bancarias, bases de datos, entre otras. Las técnicas de comprensión sin pérdidas que se presentan son de tipo estadísticas y basadas en diccionario.

La compresión sin pérdida es aplicada fundamentalmente en los ficheros de textos. (Briceño M, 2005).

Dentro de esta familia se encuentra los algoritmos que se describen a continuación.

1.2.3.2.1 Run Length Encoding (RLE)

La compresión RLE es una forma muy simple de compresión de datos en la que secuencias de datos con el mismo valor son almacenadas como un único valor más su recuento. RLE se usa para comprimir cualquier tipo de secuencia de datos repetidos.

Se sigue el siguiente patrón:

Sc: Símbolo especial que indica que sigue la compresión.	Cc: Número de veces que se repite el símbolo comprimido.	X: Símbolo de dato repetido.

A continuación se muestra un ejemplo de comprensión RLE:
Cadena original de datos: A******45.76, parabbbbbesto.
Cadena comprimida: AS_c6*45.76, para$S_c5$besto.

1.2.3.2.2 LZ77

El algoritmo de compresión LZ77 es muy usado porque es fácil de implementar y es bastante eficiente. En 1977 Abraham Lempel y Jacob Ziv presentaron su modelo de compresión basado en diccionario para compresión de texto. Hasta esa fecha todos los algoritmos de compresión desarrollados eran básicamente compresores estáticos. El nuevo modelo fue llamado LZ77 (por razones obvias). La salida consistía siempre en desplazamientos o corrimientos y tamaños del texto. Este algoritmo utiliza parte de la

secuencia de entrada previa a la actual como diccionario. El diccionario es una porción de la secuencia previamente codificada.

El codificador examina la secuencia de caracteres de entrada a través de una ventana deslizante que consta de dos partes:

- Un buffer de búsqueda (search buffer): porción de la secuencia recién codificada.
- Un buffer de anticipación (look-ahead buffer): siguiente porción a codificar.

Para codificar la secuencia que hay en el buffer de anticipación se mueve un puntero de búsqueda hacia atrás sobre el buffer de búsqueda hasta que encuentra un carácter que concuerda con el primero del buffer de anticipación.

A continuación, en la Figura 5, se muestra el funcionamiento del algoritmo LZ77.

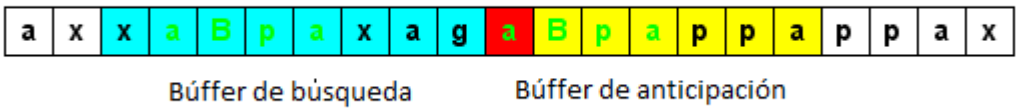

Figura 5. Funcionamiento del algoritmo LZ77.

La distancia del puntero al buffer de anticipación se denomina offset (o). Según el procedimiento se comprueba si los símbolos que vienen a continuación del puntero concuerdan con los símbolos consecutivos del buffer de anticipación.

El número de símbolos consecutivos en el buffer de búsqueda que concuerdan con los símbolos consecutivos del buffer de anticipación se llama longitud (l).

Según el procedimiento se busca la coincidencia de mayor longitud.

A continuación, en la Figura 6 se muestra la búsqueda de coincidencias consecutivas para algoritmo LZ77.

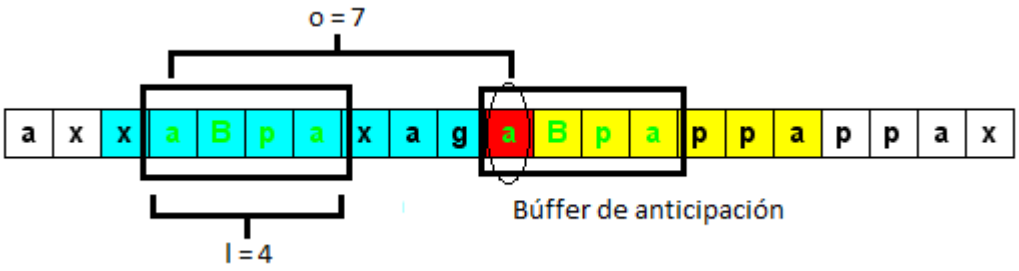

Figura 6. Búsqueda de coincidencias para algoritmo LZ77.

Una vez que el puntero ha encontrado la coincidencia de mayor longitud, la codifica con una tripleta <o, l, c>, donde c es la palabra de código correspondiente al símbolo del buffer de anticipación que viene a continuación de la coincidencia.

En el ejemplo, el bloque **"aBpa"** se codifica como <o, l, c> = <7, 4, C (p)>.

1.2.3.2.3 LZ78

Es el algoritmo de compresión más utilizado. De él se derivan el compress de Unix, el formato GIF y el algoritmo LZW. Se divide la cadena en trozos (frases) de forma que cada trozo es uno de los anteriores más un símbolo.

En lugar de ventana deslizante, construye explícitamente un diccionario idéntico para el compresor y para el descompresor.

El algoritmo genera parejas **<i, C>:**

- **i**: es un índice que indica la posición dentro del diccionario de la secuencia previa más larga coincidente con la entrada.
- **C**: es el código del carácter que viene a continuación de la secuencia coincidente.

Si no se encuentra coincidencia, **i = 0** cada nueva pareja generada **<i, C>** se añade al diccionario.

A continuación, en la Figura 7, se muestra un ejemplo de secuencia.

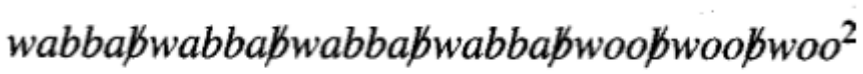

Figura 7. Secuencia

Para codificar la secuencia por este método se comienza con un diccionario vacío.

1. Los tres primeros caracteres no están en el diccionario por lo que se generan las parejas <0, C(w)>, <0, C(a)>, <0, C(b)>.
2. Se añaden estos caracteres nuevos al diccionario, asignándoles índices correlativos como se muestra en la Tabla 1.

Índice	Entrada
1	w
2	a
3	b

Tabla 1. Diccionario inicial LZ78

3. Se continúa de la misma forma, añadiendo patrones cada vez más largos al diccionario, como se muestra a continuación

Salida	Índice	Entrada
<0,C(w)>	1	w
<0,C(a)>	2	a
<0,C(b)>	3	b
<3,C(w)>	4	ba

Tabla 2. Procedimientos algoritmo LZ78

1.2.3.2.4 LZW

La mayoría de los métodos de compresión se basan en un análisis inicial del texto para identificar cadenas repetidas para armar un diccionario de equivalencias, asignando códigos breves a estas cadenas. En una segunda etapa, se convierte el texto utilizando los códigos equivalentes para las cadenas repetidas. Esto

requiere dos etapas, una de análisis y una segunda de conversión y también necesita que el diccionario se encuentre junto con el texto codificado, incrementando el tamaño del archivo de salida.

La variante más conocida del LZ78 es el algoritmo LZW, propuesto por Terry Welch (1984). Su objetivo es eliminar la necesidad del segundo elemento de la pareja <i, C>: sólo se generan índices como salida.

Para ello arranca con un diccionario inicial que contiene todos los caracteres simples, por ejemplo, la tabla ASCII extendida.

A continuación, en la Figura 8, se muestra el seudocódigo del algoritmo LZW (Salomón, 2007):

```
BEGIN
STRING = get input character;
WHILE there are still input characters DO
    CHARACTER = get input character;
    IF STRING+CHARACTER is in the string table
      then STRING = STRING+CHARACTER
    ELSE
      output the code for STRING;
      add STRING+CHARACTER to the string table;
      STRING = CHARACTER
    END of IF
END of WHILE
output the code for STRING
END
```

Figura 8. Seudocódigo algoritmo LZW.

El procedimiento se describe a continuación utilizando la secuencia:
wabba_wabba_wabba_wabba_woo_woo_woo

A continuación, en la Tabla 3, se muestra como queda el diccionario de código:

Diccionario de decodificación			
Entrada	**Índice**	**Diccionario**	**Salida**
	1	_	
	2	a	
	3	b	
	4	o	
	5	w	

5			w
2	6	wa	a
3	7	ab	b
3	8	bb	b
2	9	ba	a
1	10	a_	_
6	11	_w	wa
8	12	wab	bb
10	13	bba	a_
12	14	a_w	wab
9	15	wabb	ba
11	16	ba_	_w
7	17	_wa	ab
16	18	abb	ba_
5	19	ba_w	w
4	20	wo	o
4	21	oo	o

Tabla 3. Diccionario de código LZW.

1.2.3.2.5 Aritmético

Con esta técnica de compresión no es necesario que las probabilidades de los símbolos del alfabeto fuente sean potencias de dos para obtener una eficiencia óptima. La misma comprime una secuencia de entrada de símbolos del alfabeto fuente mediante un número representado en punto flotante. El proceso se basa en asignar a cada símbolo un intervalo entre 0 y 1, de forma que la amplitud de cada intervalo sea igual a la probabilidad de cada símbolo.

Para hallar los nuevos intervalos se usan las siguientes formulas:

- Inf(i) = Inf (i-1) + (sup (i-1)-Inf (i-1)) * Inf (i)
- Sup (i) = Inf (i-1) + (sup (i-1)-Inf (i-1)) * sup (i)

Donde:

Inf**:** intervalo inferior.

Sup**:** intervalo superior.

A continuación se muestra un ejemplo de la compresión Aritmética:

Símbolo	Probabilidad	Intervalo
X1	0.1	[0 , 0.1]
X2	0.3	[0.1 , 0.4]
X3	0.4	[0.4 , 0.8]
X4	0.15	[0.8 , 0.95]
X5	0.05	[0.95 , 1]

Tabla 4. Compresión aritmética

Secuencia a Enviar: $X_3X_5X_2X_1X_4X_5$

Se toma el primer símbolo de la cadena de entrada:

X_3 [0.4, 0.8] amplitud del intervalo: 0.4

Se toma el siguiente símbolo de la cadena de entrada y se calculan los nuevos límites de su intervalo asociado:

X_5 [0.95, 1.0]

Nuevo límite inferior = 0.4+ (0.95*0.4)=0.78

Nuevo límite superior = 0.4+ (1.0*0.4)=0.8

Resultado: x5 [0.78, 0.8] amplitud del intervalo: 0.02

X_2 [0.1, 0.4]

Nuevo límite inferior = 0.78+ (0.02*0.1)=0.782

Nuevo límite superior = 0.78+ (0.02*0.4)=0.788

Resultado: x2 [0.782,0.788] amplitud del intervalo: 0.006

X_1 [0.0, 0.1]

Nuevo límite inferior = 0.782+ (0.006*0.0)=0.782

Nuevo límite superior = 0.782+ (0.006*0.1)=0.7826

Resultado: x1 [0.782,0.7826] amplitud del intervalo: 0.0006

X_4 [0.8, 0.95]

Nuevo límite inferior = 0.782+ (0.0006*0.8)=0.78248

Nuevo límite superior = 0.782+ (0.0006*0.95)=0.78257

Resultado: x4 [0.78248, 0.78257] amplitud del intervalo: 0.00009

X_5 [0.95, 1.0]

Nuevo límite inferior = 0.78248+ (0.00009*0.95)=0.7825655

Nuevo límite superior = 0.78248+ (0.00009*1.0)=0.78257

Resultado: X_5 [0.7825655, 0.78257]

Finalmente se elige un valor perteneciente al último intervalo que se obtiene y este valor es el que se transmite a través del canal.

1.2.4 Multiplexión

Es habitual que dos estaciones que se vayan a comunicar no utilicen toda la capacidad del enlace de datos. Por cuestiones de rendimiento, es conveniente compartir esa capacidad. El término genérico que alude a ese proceso de compartir es la multiplexión. (Stallings, 2004).

En la práctica es necesario enviar simultáneamente una gran cantidad de mensajes diferentes por un medio de transmisión dado. El proceso de operación multicanal permite, mediante las técnicas llamadas de “multiplex” o “multiplexamiento”, combinar en el transmisor los mensajes de varias fuentes de información, transmitirlos como un solo bloque y luego separarlos en el receptor. (Márquez, 2005).

Los tipos de multiplexión más usados según el sistema de comunicación son: (Márquez, 2005)
Por división de tiempo o TDM (Time division multiplexing): usada con señales digitales o con señales analógicas que transportan datos digitales y estos datos procedentes de varias fuentes se transmiten en tramas repetitivas.

- De frecuencia o FDM (Frequency división multiplexión): usada con señales analógicas de una banda de frecuencia diferente para cada señal.
- En código o CDM (Code division multiplexing): usada para transmitir tanto señales analógicas como digitales, haciendo uso de una señal analógica.

Terminado el procedimiento de multiplexión el sistema pasa a la modulación que nace de la necesidad de transportar una información a través de un canal de comunicación a la mayor distancia y menor costo posible.

1.2.5 Modulación

Posterior al proceso de multiplexado de la información se procede a la modulación de los impulsos de la

misma que consiste en colocar la información contenida en una señal, generalmente de baja frecuencia (señal moduladora), sobre una señal de alta frecuencia (señal portadora).

En la transmisión de señales portadoras es posible identificar dos tipos básicos de modulación: (Márquez, 2005)

- ❖ Modulación de Señales Continuas: proceso continuo apropiado para señales que varían en forma constante en el tiempo. La señal portadora es sinusoidal.
- ❖ Modulación de Impulsos: proceso discreto, apropiado para la transmisión de mensajes o información de naturaleza discreta en el sentido de que los impulsos están presentes en ciertos intervalos de tiempo. La señal portadora es un tren de impulsos.

1.2.6 Resultado del Análisis

Los procesos que intervienen en la transmisión de datos son de sumo interés para que la información llegue rápidamente, no sufra cambios y no sea interceptada por agentes externos a la comunicación. De los procesos estudiados en el Esquema Ampliado del Transmisor de Shannon se seleccionó para ser representados en el sistema a desarrollar los siguientes:

- ❖ Del subsistema Codificación los algoritmos Shannon-Fano y Huffman de la familia de las técnicas eficientes y de la técnicas redundantes, lo algoritmos Hamming y Códigos Cíclicos.
- ❖ Del subsistema Compresión, los algoritmos RLE (Run-length encoding) con mapeo de bits, la familia de los LZ que comprende al LZ77, LZ78 y LZW y el aritmético, todos de la familia de las técnicas de compresión sin pérdidas.

La elección de los algoritmos se realiza en base al programa de la asignatura Teleinformática siendo estos los principales algoritmos estudiados.

1.3 Herramientas para la simulación de la transmisión de datos

En el Departamento de Sistemas Digitales de la UCI, en reiteradas ocasiones los resultados académicos en la asignatura Teleinformática, específicamente en el tema de transmisión de datos, no han sido lo esperados ya que no cuentan con una herramienta para visualizar y evaluar los procesos que tienen lugar en la transmisión de datos, por lo que se ha pensado en el empleo de herramientas que muestren cómo

ocurren los procesos que intervienen en la comunicación. Para el estudio de esas herramientas se tuvieron en cuenta básicamente dos indicadores: que simularan procesos y que se usaran en la UCI actualmente, con vistas a poder desarrollar una herramienta que fuera de fácil entendimiento para los estudiantes. Las herramientas estudiadas fueron:

1. Matlab
2. VirtualNet

1.3.1 Matlab

Matlab es una herramienta educativa que facilita la resolución de funciones matemáticas y permite realizar operaciones estadísticas. Se comporta como un simulador de redes neuronales y permite además, la simulación del proceso de transmisión de datos. Para la simulación de la emisión y recepción de la información este software brinda las siguientes opciones:

- Simulink.
- Funciones .m utilizadas desde la línea de comandos.
- Funciones .m utilizadas mediante Interfaces Gráficas de Usuario (GUI).

Las últimas versiones de Matlab incorporan la posibilidad de utilizar interfaces gráficas de usuario para trabajar con sus archivos .m, esto permite tener un entorno amigable para pedir datos y mostrar resultados al usuario, además puesto que es posible implementar funciones .m propias. El sistema que se desea simular estará totalmente personalizado desde el punto de vista de las funcionalidades y de las interfaces de usuario.

Es utilizada con fines didácticos en la impartición de la asignatura Matemática Numérica correspondiente a la disciplina de Ciencias Básicas de la Universidad de las Ciencias Informáticas.

1.3.2 VirtualNet

VirtualNet es una herramienta de simulación de redes de computadoras que integra los contenidos de diseño de redes, direccionamiento IP y de enrutamiento ejecutándose a partir del diseño de una red WAN. Es utilizada con fines didácticos en la impartición de la asignatura Teleinformática correspondiente a la disciplina de Sistemas Digitales de la Universidad de las Ciencias Informáticas (UCI). Se desarrolló sobre

la plataforma Java con su IDE de desarrollo NetBeans 6.0, RUP como metodología de desarrollo, Visual Paradigm como herramienta CASE y UML como lenguaje de modelado.

Entre sus principales funcionalidades se encuentran:

- ❖ Mostrar el funcionamiento de un sistema de transmisión de datos.
- ❖ Permitir diseñar redes.
- ❖ Permitir direccionar redes.
- ❖ Permitir realizar los algoritmos de enrutamiento Vector Distancia y Estado de Enlace.
- ❖ Mostrar paso a paso los procedimientos de direccionamiento y enrutamiento.
- ❖ Mostrar ayuda.

1.3.3 Resultado del Análisis

Se puede concluir que las herramientas antes estudiadas presentan características útiles para la realización de la herramienta educativa Simulador del Transmisor de Shannon.

Se tiene en cuenta que:

1. Son herramientas educativas de apoyo a la docencia.
   - ❖ VirtualNet apoya la asignatura Teleinformática correspondiente a la disciplina de Sistemas Digitales de la Universidad de las Ciencias Informáticas.
2. Son vistas como un simulador.
   - ❖ VirtualNet simula redes de computadoras, tema asociado a la transmisión de datos, que aunque no trate los procesos que intervienen en el mismo y los algoritmos mediante los cuales se desarrollan, sirvió de base pues da la idea de cómo debe verse un sistema de esta índole. Aportó la idea de la demostración paso a paso de los algoritmos de la herramienta, el uso de una ayuda de apoyo para los estudiantes, entre otros aportes.
   - ❖ Matlab simula la transmisión de datos y permite tener un entorno amigable para pedir datos y mostrar resultados al usuario.
3. Metodologías y herramientas de desarrollo.
   - ❖ VirtualNet se desarrolló sobre la plataforma Java con su IDE NetBeans 6.0, Visual Paradigm como herramienta CASE y UML como lenguaje de modelado.

Teniendo en cuenta que hoy no existe una aplicación que simule los algoritmos de los distintos subsistemas que abarca el esquema Ampliado del Transmisor de Shannon, capaz de usarse como herramienta de apoyo a la impartición de dicho contenido, se tuvo la necesidad de realizar un sistema nuevo y se decidió poner en práctica durante su desarrollo todo lo aprendido de las herramientas estudiadas.

1.4 Tecnologías y herramientas

En la realización del producto, se hizo necesario el estudio de diferentes metodologías, herramientas y lenguajes de programación, que al elegir la apropiada en cada uno de los casos, se realice el diseño e implementación de la aplicación. Después de un estudio exhaustivo son seleccionadas debido a sus buenas prestaciones y características para llevar a cabo el desarrollo del mismo.

1.4.1 Metodología de Desarrollo

Las metodologías de desarrollo de software surgen ante la necesidad de utilizar una serie de procedimientos, técnicas, herramientas y soporte documental a la hora de desarrollar un producto de software. Dichas metodologías pretenden guiar a los desarrolladores al crear un nuevo software, pero los requisitos de un software son tan variados y cambiantes, que ha dado lugar a que exista una gran variedad de metodologías para la creación del software. (Jacobson y otros).

En la actualidad se clasifican en ágiles o tradicionales.

Metodologías tradicionales: Metodologías orientadas al control de los procesos, estableciendo rigurosamente las actividades a desarrollar, herramientas a utilizar y notaciones a usar. Dentro de las metodologías tradicionales se encuentran RUP, Microsoft Solution Framework (MSF), etc.

Metodologías ágiles: Metodologías orientadas a la interacción con el cliente y el desarrollo incremental del software, mostrando versiones parcialmente funcionales del software al cliente en intervalos cortos de tiempo, para que pueda evaluar y sugerir cambios en el producto según se va desarrollando. Dentro de las ágiles se encuentra Crystal, Método de Desarrollo de Sistemas Dinámicos (DSDM), Programación Extrema (XP) y SCRUM.

A continuación, en la Tabla 5, se muestra la comparación entre ambos tipos de metodologías.

Metodología Ágil	Metodología Tradicional
Basadas en heurísticas provenientes de prácticas de producción de código.	Basado en normas provenientes de estándares seguidos por el entorno de desarrollo.
Especialmente preparados para cambios durante el proyecto.	Cierta resistencia a los cambios.
Impuestas internamente (por el equipo).	Impuesta externamente.
Proceso menos controlado, con pocos principios.	Proceso mucho más controlado, con numerosas políticas y normas.
No existe contrato tradicional o al menos es bastante flexible.	Existe un contrato prefijado.
El cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante reuniones.
Grupos pequeños y trabajando en el mismo sitio.	Grupo grande y posiblemente distribuido.
Pocos artefactos.	Más artefactos.
Pocos roles.	Más roles.

Tabla 5. Comparación entre metodología ágil y metodología tradicional.

La Tabla muestra las diferencias entre las metodologías ágiles y tradicionales que al ser estudiadas, se selecciona la metodología ágil para darle solución a la propuesta, ya que fue indispensable aplicar un proceso de ingeniería inversa, por la necesidad urgente del cliente en obtener el producto, por ende fue la pieza clave que llevó a utilizar este tipo de metodología además de ser la que más se acoplaba a las necesidades del cliente y del pequeño equipo de desarrollo.

En la presente investigación se analizan las metodologías ágiles XP y SCRUM las cuales muestran efectividad en proyectos con requisitos muy cambiantes y cuando se decide reducir drásticamente los tiempos de desarrollo permiten mantener una alta calidad, además dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas.

1.4.1.1 Programación Extrema (XP)

XP es una de las metodologías ágiles de desarrollo de software más exitosa cuyo autor es Kent Beck, programador de vasta experiencia que eligió las mejores características de las metodologías y profundizó en las relaciones de estas y como se reforzaban unas a otras. Por tanto, la XP no se basa en principios nuevos, sino que todas, o casi todas sus características ya se conocen dentro de la ingeniería del software, las cuales se complementan para minimizar los tópicos problemas que pueden surgir en todo desarrollo de proyectos software. (Valverde, 2012).

Es utilizada para proyectos de corto plazo, con poco personal de trabajo, consiste en una programación rápida o extrema cuya particularidad es tener como parte del equipo al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto. Está diseñada para entregar el software que los clientes quieren en el momento en que lo necesitan, alienta a los desarrolladores a responder a los requerimientos cambiantes de los clientes, aún en fases tardías del ciclo de vida del desarrollo, y el cliente se siente satisfecho de recibir un software que se adapte exactamente a sus deseos. La metodología XP define cuatro variables para cualquier proyecto de software: costo, tiempo, calidad y alcance.

Una de las características del comienzo de la XP dentro de un equipo es que esta metodología incluye la programación en parejas, pero en la práctica se acepta rápidamente y de forma entusiasta. El hecho de que todas las decisiones las tomen al menos dos personas proporciona un mecanismo de seguridad enormemente valioso.

Otra práctica fundamental de la Programación Extrema es utilizar diseños tan simples como sea posible, para que todo funcione. Se evita diseñar características extras porque en la práctica la experiencia indica que raramente se puede anticipar qué necesidades se convertirán en reales y cuáles no. La metodología XP se divide en 4 fases descritas a continuación.

Fase de Planificación

En esta fase es donde se interactúa con el usuario, para hacer una recopilación de los requerimientos del proyecto, especificando las características que son adicionadas al sistema mediante las Historias de Usuario (HU). Se planifica con todos los desarrolladores de lo que se quiere, para obtener buenos

resultados y lograr el objetivo final. El tiempo ideal de desarrollo para una HU es entre 1 y 3 semanas. Después de concluida las HU se hace un plan (Release) de publicaciones donde se plasman las historias de usuario por cada versión según la prioridad del cliente y las fechas correspondientes en la que se publicará dejando bien claro los objetivos que se deben cumplir, el tiempo en que tardarán en desarrollarse, el personal que va a estar involucrado y definir cómo se va a evaluar el trabajo realizado. En esta fase es donde se especifican las iteraciones, se estima la velocidad del proyecto y se reúnen a diario los desarrolladores para explicar sus problemas y llegar a una solución.

Fase de Diseño

XP sugiere que hay que conseguir diseños simples y sencillos, procurar lo menos complicado posible obtener un diseño fácil de entender por el usuario o el cliente posibilitando una mejor interacción, en esta fase se crea la parte física del proyecto, la interfaz gráfica. También se define entre todos una metáfora para seguir un mismo lenguaje que sea entendible por todos y se crean las tarjetas CRC (Clase-Responsabilidades-Colaboración)

Fase de Codificación

En esta fase los clientes y los desarrolladores del proyecto deben elevar el nivel de comunicación favoreciendo a que los desarrolladores puedan codificar lo que se requiere para el proyecto, siguiendo los estándares de codificación para facilitar la comprensión a la hora de programar. La programación en pareja permite un código más eficiente que almacenado en un repositorio.

Fase de Pruebas

Es donde se comprueba el funcionamiento de la aplicación después de haberla implementado, verificando que todas las funcionalidades se ejecuten correctamente.

Roles XP

Los roles de acuerdo con la propuesta original de Beck son:

- **Programador:** El programador escribe las pruebas unitarias y produce el código del sistema.
- **Cliente:** Escribe las historias de usuario y las pruebas funcionales para validar su implementación.

Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio.

- **Encargado de pruebas (Tester):** Ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.
- **Encargado de seguimiento (Tracker):** Proporciona realimentación al equipo. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada iteración.
- **Entrenador (Coach):** Es responsable del proceso global. Debe proveer guías al equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.
- **Consultor:** Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir problemas.
- **Gestor (Big boss):** Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

1.4.1.2 SCRUM

Es una metodología ágil de administración de proyectos, entre ellos, proyectos de desarrollo de software. Con SCRUM los proyectos progresan a través de una serie de iteraciones que duran entre una o dos semanas y un mes, llamadas "Sprint". No es una metodología de análisis, ni de diseño, es una metodología de gestión del trabajo.

Esta metodología está basada en el modelo "espiral" que se aplica cuando no se conoce el dominio de aplicación del problema, cuando los desarrolladores y usuarios tienen poca experiencia en el tema, hay falta de precisión sobre los problemas a resolver, los requisitos son inestables y hay condicionamientos estrictos en cuanto a tiempo y costo. Sus prácticas implementan la visibilidad, adaptación e inspección. Los roles en esta metodología se dividen en dos categorías: los implicados (usuarios finales, áreas comerciales, áreas contables, marketing, etc.) y los comprometidos (jefe del equipo, dueño del producto y el equipo) para garantizar que las personas que tienen la responsabilidad poseen además la autoridad necesaria para poder lograr el éxito del proceso y que quienes no la posean no produzcan interferencias innecesarias. (Kniber, 2007).

1.4.1.3 Metodología seleccionada

A continuación, en la Tabla 6, se muestra las características de ambas metodologías.

	SCRUM	**XP**
Características	❖ Reuniones a lo largo del proceso de desarrollo. ❖ Desarrollo en iteraciones denominadas sprint. ❖ El resultado de cada sprint es un ejecutable que se muestra al cliente.	❖ Flexibilidad en el tratamiento de historias de usuario. ❖ Programación en pares. ❖ Integración de modo continúo. ❖ Estándares de código. ❖ Constante intercambio con el cliente. ❖ Refactorización.
Flexibilidad a cambios	❖ Facilita cambios durante el proceso de desarrollo	❖ Facilita cambios durante el proceso de desarrollo.
Equipo de desarrollo	❖ Equipos pequeños.	❖ Equipos pequeños, no menos de 10 personas y trabajando en el mismo sitio.
Documentación generada	❖ Poca documentación	❖ Muy poca documentación

Tabla 6. Comparación de las metodologías.

Posterior al estudio de las metodologías se propone aplicar la metodología de desarrollo XP, que requiere un pequeño grupo de programadores para su desarrollo, está mucho más orientada a las personas que a los procesos por lo que no necesita de mucha documentación y así el trabajo no se vuelve tan engorroso. Además, reduce el costo de cambio en las etapas de vida del sistema y presenta un diseño evolutivo, haciendo que no se le dé apenas importancia al análisis como fase independiente, puesto que se trabaja exclusivamente en función de las necesidades del momento, siendo XP una metodología ágil en su desarrollo, posibilitando la modificación continua del código existente. SCRUM se enfoca en las prácticas de organización y gestión, mientras que XP se centra más en las prácticas de programación o creación del producto. SCRUM es una de las más populares en Internet, pero no es la más adecuada para la solución de la propuesta.

1.4.2 Lenguaje Unificado de Modelado (UML)

Como su nombre lo indica, UML es un lenguaje de modelado visual que permite modelar la complejidad de los sistemas a analizar o diseñar, se centra en la representación gráfica de un sistema. Este lenguaje

indica cómo crear y leer los modelos, pero no dice cómo crearlos. Permite modelar sistemas utilizando tecnologías orientada a objetos (OO).

Las funciones fundamentales de UML son: (Orallo, 2001).

- Visualizar: UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
- Especificar: UML permite especificar cuáles son las características de un sistema antes de su construcción.
- Construir: A partir de los modelos especificados se pueden construir los sistemas diseñados.
- Documentar: Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.

Para el modelado se utilizará el UML como lenguaje representativo porque permite la representación gráfica de un sistema con tecnologías orientada a objetos, especifica cuáles son las características antes de su construcción, partiendo de los modelos especificados que contribuyen al diseño y los propios elementos gráficos generados se utilizan como documentación de la herramienta desarrollada que pueden servir para su futura revisión.

Una vez seleccionado el lenguaje de modelado visual se procede a seleccionar el lenguaje de programación para el desarrollo de la solución propuesta.

1.4.3 Lenguajes de programación

Los lenguajes de programación son un conjunto de símbolos, reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos o expresiones, se utilizan para comunicarse con el ordenador enviando y recibiendo información. A través de dichos lenguajes, el usuario podrá decirle al ordenador lo que debe hacer, así como preguntarle por cualquier información. Gracias a la programación, el ordenador realiza las acciones requeridas por el usuario y da respuestas. Al igual que con los lenguajes humanos, esta comunicación se realiza mediante un vocabulario y una gramática ya establecidos.

Entre los lenguajes de programación se encuentran C++ y Java .A continuación una breve descripción de los mismos.

1.4.3.1 Java

Java es un lenguaje de programación creado por SUN Microsystems muy parecido al estilo de programación del lenguaje “C” y basado en lo que se llama programación orientada a objeto. Entre las principales características de Java se pueden citar:

- **Sintaxis similar a la de C++.** Aunque se simplifican algunas características del lenguaje como: la sobrecarga de operadores, la herencia múltiple, el paso por referencia de parámetros, la gestión de punteros, la liberación de memoria y las instrucciones de pre compilación.
- **Soporte homogéneo a la Programación Orientada a Objetos:** A diferencia de C++, que puede considerarse un lenguaje multiparadigma, Java está diseñado específicamente para utilizar el paradigma de orientación a objetos.
- **Independencia de la plataforma:** En Java se pretende que con una sola compilación se obtenga código ejecutable en diferentes Sistemas Operativos e incluso sobre diferente hardware. (Vélez Serrano y otros, 2005).

La compañía SUN describe el lenguaje Java como “simple” ,orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico”. (García de Jalón y otros, 2000).

- Orientado a objetos: Desde un principio fue diseñado orientado a objetos que agrupan en estructuras encapsuladas tanto sus datos como los métodos que manipulan esos datos.
- Distribuido: Proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir sockets y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas.
- Interpretado y compilado a la vez: Java es compilado, en la medida en que su código fuente se transforma en una especie de código máquina, los bytecodes, semejantes a las instrucciones de ensamblador. Por otra parte, es interpretado, debido a que los bytecodes se pueden ejecutar directamente sobre cualquier máquina a la cual se hayan portado el intérprete y el sistema de ejecución en tiempo real.
- Robusto: Fue diseñado para crear software altamente fiable. Para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución.
- Sus características de memoria liberan a los programadores de una familia entera de errores como la aritmética de punteros, porque ya en este lenguaje se ha prescindido por completo de los punteros, y la

recolección de basura elimina la necesidad de liberación explícita de memoria.

❖ Multihilo: En la actualidad muchas aplicaciones realizan varias operaciones al mismo tiempo, este lenguaje posee la característica de que los programadores puedan explotar esto, pues java permite este tipo de programación en el cual se crean varios procesos que se encargan de realizar cálculos y operaciones distintas en el mismo instante.

Java está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red, desde Unix a Windows NT, pasando por Mac y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos diversos. La indiferencia a la arquitectura representa sólo una parte de su portabilidad. Además, especifica los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, de manera que los programas son iguales en todas las plataformas.
Estas dos últimas características se conocen como la Máquina Virtual Java (JVM).

1.4.3.2 C++

C++ es un lenguaje de programación que permite particularmente, redefinir operadores y crear nuevos tipos de datos que se comporten como tipos fundamentales.

A pesar de que C++ brinda variadas funcionalidades, el trabajo con punteros se hace engorroso. Este elemento se hace extensivo tanto para su creación como para su eliminación (los llamados destructores). Por lo tanto, teniendo en cuenta además que para trabajar con este lenguaje se requiere preparación previa y experiencia, se decidió no utilizarlo para la implementación del sistema simulador del trasmisor de Shannon (STX). (Mozilla, 2014).

Para la elección de uno de los lenguajes que se tomaron como referencia, se tuvieron en cuenta factores que influyen a la hora de elegir a uno de los dos. Entre estos factores se pueden mencionar el significado de software libre, esto se debe a que para Java existen potentes IDE de desarrollo que son libres y C++ posee algunos que todavía están empezando a desarrollarse. Otro factor es que las aplicaciones desarrolladas en Java pueden ejecutarse en diferentes sistemas operativos siempre que estos tengan la máquina virtual instalada, sin embargo las que están desarrolladas en C++ no poseen esta facilidad.

Por estas razones se ha elegido al Java como el lenguaje idóneo para la implementación del software.

1.4.4 Entorno de Desarrollo Integrado

Existen diferentes formas de introducir un programa Java en una computadora. Se puede utilizar un Ambiente de Desarrollo Integrado (IDE, por sus siglas en inglés) o un editor de textos plano. Un IDE es más que una larga pieza de software, que permite introducir, compilar y ejecutar programas. La introducción, compilación y ejecución son parte del desarrollo de un programa y están integradas juntas en un ambiente, de ahí el nombre ambiente de desarrollo integrado. Algunos IDE son gratuitos y otros muy caros. Dentro de los entornos de desarrollo integrado (IDE) para el desarrollo de aplicaciones usando como lenguaje de programación Java se encuentra NetBeans.

NetBeans

NetBeans es un entorno de programación para varios lenguajes, incluyendo a Java y C++.
Este desarrollo es de fuente abierto, es decir, se proporciona el código fuente del entorno para que se pueda modificar de acuerdo a ciertos parámetros de licencia, es un producto libre y gratuito sin restricciones de uso. (Universidad Carlos III, 2002). La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados subsistemas o módulos.

Para el desarrollo de la aplicación se utilizó **NetBeans v7.4** que presenta un detalle habitual genérico en todas las versiones que se basa en que son demasiado “pesadas”, es decir, necesitan una máquina muy potente para poder ejecutarse de forma satisfactoria.

Java Development Kit 1.8 o (JDK):

Es un software que provee herramientas de desarrollo para la creación de programas en Java. Puede instalarse en una computadora local o en una unidad de red. En la unidad de red se pueden tener las herramientas distribuidas en varias computadoras y trabajar como una sola aplicación.

Posterior a la selección de la metodología de desarrollo de software, del lenguaje de modelado visual y del lenguaje de desarrollo para aplicaciones de escritorio se realizará un análisis de las herramientas CASE que le dan soporte a la selección anterior.

1.4.5 Modelado de Negocio

Visual Paradigm for UML 8.0

Visual Paradigm es una herramienta UML CASE, muy completa y fácil de usar, con soporte multiplataforma y que proporciona excelente facilidades de interoperabilidad con otras aplicaciones. Permite modelar todo tipo de diagramas UML, generar código desde diagramas, generar documentación, realizar ingeniería tanto directa como inversa, entre otras funciones. Soporta el ciclo de vida completo del desarrollo de software que lo automatiza y acelera, permitiendo la captura de requisitos, análisis, diseño e implementación. Permite invertir código fuente de programas, archivos ejecutables y binarios en modelos UML al instante, creando de manera simple toda la documentación. Está diseñado para usuarios interesados en sistemas de software de gran escala con el uso del acercamiento orientado a objeto, además apoya los estándares más recientes de las notaciones de Java y de UML. Incorpora el soporte para trabajo en equipo, que permite que varios desarrolladores trabajen a la vez en el mismo diagrama y vean en tiempo real los cambios hechos por sus compañeros.

Características:

- Producto de calidad.
- Varios idiomas.
- Generación de código para Java.
- Fácil de instalar y actualizar.
- Compatibilidad entre ediciones.
- Se integra la herramienta Java: NetBeans IDE.

Ventajas

- Disponibilidad en múltiples plataformas: Microsoft Windows (98, 2000, XP, o Vista), Linux, Mac OS X, Solaris o Java.
- Generación de código e ingeniería inversa: brinda la posibilidad de generar código a partir de los diagramas, para las plataformas como .Net, Java y PHP, así como obtener los diagramas a partir del código.
- Generación de documentación: brinda la posibilidad de documentar todo el trabajo sin necesidad de utilizar herramientas externas.

1.4.6 Resultado del análisis

Luego del análisis de los lenguajes de programación para aplicaciones de este tipo, se decidió utilizar el lenguaje Java con su IDE de desarrollo NetBeans v7.4 por las múltiples ventajas que éste trae consigo, tales como: es un producto libre, gratuito y sin restricciones de uso, multiplataforma. Además es robusto, de alto rendimiento y posee una gran cantidad de herramientas gratuitas.

1.5 Conclusiones parciales

Luego de terminado el capítulo 1 se concluye lo siguiente:

- Se estudiaron diferentes conceptos referentes a los sistemas de transmisión de datos y sus procesos, que apoyaron la fundamentación teórica de la presente investigación.
- De la búsqueda realizada de sistemas que permitan representar el Esquema Ampliado del Transmisor de Shannon se obtuvo que no existe ningún sistema para tal fin y por ello se hace necesario desarrollar un sistema nuevo.
- Para desarrollar el nuevo sistema se realizó un análisis de sistemas que simulen procesos semejantes.
- Del análisis de sistemas existentes para la simulación de procesos similares, el VirtualNet desarrollado en la UCI, como herramienta educativa de apoyo en la impartición de los contenidos Direccionamiento y Enrutamiento de la asignatura Teleinformática fue el que más aportó. Dentro de los valiosos aportes que se extrajeron del VirtualNet se encuentran: el uso de ayudas de apoyo, la representación del paso a paso de los algoritmos y la representación visual de los procedimientos a partir de asistentes de navegación.
- Del estudio de las herramientas y metodologías realizado se seleccionaron para el desarrollo del sistema: como metodología de desarrollo XP, como lenguaje de modelado UML, como herramienta case para el modelado Visual Paradigm for UML v8.0, como entorno de desarrollo el NetBeans v7.4 y como lenguaje de programación Java.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

2.1 Introducción

El proceso de desarrollo de software, por ser un proceso complejo, consta de muchas partes. Las definiciones de roles, actividades y artefactos, el modelado y la documentación de todos los requerimientos, han demostrado ser prácticas efectivas y necesarias en la creación de sistemas informáticos que permiten dar respuesta a las necesidades del cliente. En la presente investigación se desarrolló una aplicación capaz de ayudar en la asimilación de los conocimientos de la asignatura Teleinformática, específicamente centrados en el esquema ampliado del transmisor de Shannon.

El presente capítulo tiene como objetivo describir detalladamente la solución propuesta para resolver el problema planteado. Tal como establece la metodología XP la solución fue realizada por los programadores en presencia del cliente y los coordinadores. Se comenzó con la recopilación de los datos que fueron recogidos en las Historias de Usuarios y se obtuvo un plan que sirvió como hoja de ruta en el transcurso del proceso de desarrollo de software. Además, se presentan los requerimientos tanto funcionales como no funcionales que el sistema debe cumplir y otros artefactos generados durante el desarrollo de la solución tales como: las tarjetas CRC (Clase-Responsabilidad-Colaboración), que identifican las responsabilidades de las clases como parte del diseño.

2.2 Solución propuesta

Un equipo de desarrollo comienza a trabajar en la creación de una solución informática a partir de una idea que permite imaginar y percibir la utilidad del producto final. La idea de crear un simulador capaz de realizar y mostrar las funcionalidades de los subsistemas de un transmisor de datos genérico, diseñado para contribuir a elevar la cantidad y calidad de los recursos educativos generados en las Instituciones de Educación Superior, constituye la propuesta de solución para el problema a resolver que dará cumplimiento a los objetivos planteados. El sistema bautizado como Simulador del Transmisor de Shannon (STX) en su primera versión cuenta con la implementación de los subsistemas Codificación y Compresión.

2.3 Personas relacionadas con el sistema

Todo sistema creado o en desarrollo, implementa procesos complejos del mundo real, cuenta con funcionalidades que procesan datos y devuelve algún resultado. Una persona relacionada con el sistema es toda aquella que participa directa o indirectamente en su desarrollo o lo utiliza como producto en función de la sociedad o para beneficio propio. La presente propuesta va dirigida principalmente a los profesores del Departamento Central de Técnicas de Programación los cuales en lo adelante se encargarán de distribuirla a todas las facultades de la UCI.

2.4 Planificación

La planificación como primera fase planteada por la metodología utilizada tiene como meta proyectar lo que se quiere, para obtener buenos resultados y lograr el objetivo final. En esta fase se capturan los requisitos, se estima la velocidad del proyecto y se explican y desglosan los problemas, hasta llegar a una solución.

2.4.1 Requerimientos de software

Un requerimiento se define como un atributo necesario dentro de un sistema, que representa una capacidad, una característica o un factor de calidad del mismo, de tal manera que le sea útil a los clientes o a los usuarios finales. Según Somerville y Sawyer, definición tomada para la presente investigación, un requerimiento es una especificación de lo que debería ser implementado, una descripción de cómo el sistema debería comportarse o de una propiedad o atributo del sistema.

A nivel general los requerimientos pueden clasificarse como requerimientos funcionales o no funcionales. Los funcionales son los entregados por el usuario al comienzo del proyecto, en tanto que los no funcionales son aquellos que reflejan la satisfacción de las necesidades del usuario en un sistema en particular. La metodología XP plantea, en su fase de planificación, la necesidad de interactuar con el cliente para la recopilación y especificación de las características del proyecto, que son posteriormente adicionadas al sistema mediante las Historias de Usuario (HU).

2.4.2 Técnicas de obtención de requerimientos de software

Se requiere tiempo para llegar a especificar claramente lo que el interesado espera de la aplicación de

software, por lo que se hace necesario por parte de los analistas el empleo de técnicas que permitan establecer una buena comunicación con los interesados del producto y así lograr la satisfacción del cliente. A continuación se enuncian las principales técnicas utilizadas durante el proceso de desarrollo del software para recopilar los requisitos de software:

- **Sistemas existentes**

Esta técnica consiste en analizar distintos sistemas ya desarrollados que estén relacionados con el sistema que se desea realizar. Por un lado, se puede analizar las interfaces de usuario, observando el tipo de información que se maneja y cómo es manejada, por otro lado se analizan las distintas salidas que los sistemas producen (listados, consultas, etc.), porque siempre pueden surgir nuevas ideas sobre la base de estas.

- **Lluvia de ideas (Brainstorm)**

Este es un modelo que se usa para generar ideas. La intención de este ejercicio es generar, en una primera instancia, la máxima cantidad posible de requerimientos para el sistema. No hay que detenerse en pensar si la idea es o no del todo utilizable. Luego, se irán eliminando en base a distintos criterios tales como: costo, tiempo de desarrollo, recursos humanos, prioridad, entre otros.

2.4.3 Técnicas de validación de requerimientos de software

- **Prototipos**

Luego de la actividad de extracción de requerimientos, es muy importante validar con el cliente los requisitos obtenidos, con el objetivo de evitar gastos innecesarios de recursos en el desarrollo.

Para validar los requerimientos hallados en la presente investigación, se construyen prototipos. Los prototipos son simulaciones del posible producto, que luego son utilizados por el usuario final, permitiendo conseguir una retroalimentación en cuanto a la efectividad de los requisitos.

Para la realización de los prototipos se realiza un “diseño rápido”. Este se centra en una representación de aquellos aspectos del software que serán visibles al usuario, por ejemplo: las entradas y los formatos de las salidas.

2.4.4 Resultado del análisis

Los requerimientos funcionales y no funcionales definidos para desarrollar la propuesta de solución son los descritos a continuación y se recopilan haciendo uso de las técnicas: Lluvia de Ideas y Sistemas Existentes. La lluvia de ideas se realizó con: los tutores, el equipo de desarrollo y miembros del Departamento de Técnicas de Programación Central. El análisis de sistemas existentes permitió extraer conceptos e ideas fundamentales para el funcionamiento y manejo del software, entre las ideas obtenidas se tienen: la ayuda, el paso a paso de los algoritmos y el uso de asistentes para la navegación. La técnica de prototipos usada para la validación permitió mostrarle al cliente una metáfora del producto final y ayudó en la refinación de los requisitos capturados.

2.4.5 Requisitos funcionales

Los requisitos funcionales (RF) describen detalladamente lo que el sistema debe hacer. Dependen del tipo de software que se desarrolle, de los posibles usuarios y del enfoque general tomado por la organización al redactarlos. (Somerville, 2005).

La herramienta a desarrollar, en función de los objetivos específicos planteados en la presente investigación, consta de 30 RF agrupados en 4 calificaciones: RF subsistema Codificación, RF subsistema Compresión, RF genéricos y Otros, téngase en cuenta que los requisitos agrupados en Otros requisitos fueron solicitados por el cliente en la última iteración desarrollada y básicamente comprenden la realización de la codificación redundante sin partir de la codificación eficiente. A continuación se muestra cómo quedan recogidos:

RF genéricos:

RF1: Entrar la cadena.

RF2: Cargar cadena de un fichero del disco.

RF3: Consultar la ayuda.

RF subsistema Codificación:

RF4: Seleccionar el tipo de codificación.

RF5: Codificar eficiente una cadena de símbolos a través del algoritmo Shannon-Fano.

RF6: Mostrar paso a paso la codificación de manera eficiente de una cadena de símbolos a través del algoritmo Shannon-Fano.

RF7: Codificar de manera eficiente una cadena de símbolos a través del algoritmo Huffman.

RF8: Mostrar paso a paso la codificación de manera eficiente de una cadena de símbolos a través del algoritmo Huffman.

RF9: Codificar una cadena de símbolos codificada a través del algoritmo Shannon-Fano de manera redundante a través del algoritmo de Hamming.

RF10: Mostrar paso a paso la codificación de una cadena de símbolos codificada a través del algoritmo Shannon-Fano a través del algoritmo Hamming.

RF11: Codificar una cadena de símbolos codificada a través del algoritmo Huffman de manera redundante a través del algoritmo de Hamming.

RF12: Mostrar paso a paso la codificación de una cadena de símbolos codificada a través del algoritmo Huffman a través del algoritmo de Hamming

RF13: Codificar una cadena de símbolos codificada a través del algoritmo Shannon-Fano de manera redundante a través del algoritmo de Códigos Cíclicos.

RF14: Mostrar paso a paso la codificación de una cadena de símbolos codificada a través del algoritmo Shannon-Fano a través del algoritmo de Códigos Cíclicos.

RF15: Codificar una cadena de símbolos codificada a través del algoritmo Huffman de manera redundante a través del algoritmo de Códigos Cíclicos.

RF16: Mostrar paso a paso la codificación de una cadena de símbolos codificada a través del algoritmo Huffman a través del algoritmo de Códigos Cíclicos.

RF subsistema Compresión:

RF17: Comprimir una cadena de símbolos a través del algoritmo RLE con mapeo de bits.

RF18: Mostrar paso a paso la compresión de una cadena de símbolos a través del algoritmo RLE con mapeo de bits.

RF19: Comprimir una cadena de símbolos a través del algoritmo LZ77.

RF20: Mostrar paso a paso la compresión de una cadena de símbolos a través del algoritmo LZ77.

RF21: Comprimir una cadena de símbolos a través del algoritmo LZ78.

RF22: Mostrar paso a paso la compresión de una cadena de símbolos a través del algoritmo LZ78.

RF23: Comprimir una cadena de símbolos a través del algoritmo LZW.

RF24: Mostrar paso a paso la compresión de una cadena de símbolos a través del algoritmo LZW.

RF25: Comprimir una cadena de símbolos a través del algoritmo Aritmético.

RF26: Mostrar paso a paso la compresión de una cadena de símbolos a través del algoritmo Aritmético.

Otros requisitos:

RF27: Codificar de manera redundante una cadena de símbolos a través del algoritmo Hamming.

RF28: Mostrar paso a paso la codificación de manera redundante de una cadena de símbolos a través del algoritmo Hamming.

RF29: Codificar de manera redundante una cadena de símbolos a través del algoritmo Códigos Cíclicos.

RF30: Mostrar paso a paso la codificación de manera redundante de una cadena de símbolos a través del algoritmo Código Cíclicos.

2.4.6 Requisitos no funcionales

Para los requisitos no funcionales se ofrecen en la literatura diferentes definiciones, entre ellas: los requisitos no funcionales (RNF), son aquellos requerimientos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de éste como la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento. (Somerville, 2005).

Los requerimientos no funcionales son condiciones que debe cumplir un sistema para satisfacer un contrato o una especificación. Están regidos por las necesidades del usuario para poder resolver un problema o conseguir un beneficio determinado. Se refieren a las propiedades emergentes del sistema como la usabilidad, el rendimiento, confiabilidad, etc., por citar algunos.

El sistema STX consta de 11 RNF mostrados a continuación en la Tabla 7:

Requerimientos no funcionales
Usabilidad
RNF1 Facilidad de uso por parte de los usuarios: la solución propuesta debe presentar una interfaz amigable que permita la fácil interacción con el mismo y llegar de manera rápida y efectiva a la operación que desee.
RNF2 El sistema debe poder ser usado por cualquier persona que tenga conocimientos básicos de informática y del esquema del transmisor de Shannon. Debe, además, ser una interfaz de manejo cómodo que posibilite a los usuarios sin experiencia una rápida adaptación.
Rendimiento
RNF3 Los tiempos de respuesta y velocidad de procesamiento de la información deberán ser rápidos, no mayores de 5 segundos para compilaciones y repuestas.
Legales
RNF4 El sistema es de código abierto y acceso libre, por lo que podrá ser utilizado por cualquier

organización, respetando sólo los principios de desarrollo de software.
Confiabilidad
RNF5 El sistema debe estar preparado ante cualquier falla, garantizando la integridad y consistencia de los datos manejados.
Interfaz interna
RNF6 Con un buen diseño para el desarrollo del sistema, se logrará una fácil implementación o reestructuración de sus partes cuando sea necesario.
Software
RNF7 Sistema operativo Linux, Windows XP o superior.
RNF8 La aplicación funciona con la instalación de la máquina virtual de JAVA.
Apariencia
RNF9 La interfaz debe ser sencilla, amigable, legible, simple de usar y manteniendo una misma línea de principio a fin presentando una correcta y adecuada combinación de colores, para que cualquier usuario se sienta identificado con la misma y pueda manejarla sin depender si tiene o no dominio de la herramienta.
Hardware
RNF10 El sistema deberá contar con una computadora con requerimientos mínimos de hardware que permita la ejecución de la máquina virtual de JAVA (512 MB de RAM).
Soporte
RNF11 La aplicación debe ser de fácil instalación, con independencia de sus diferentes partes para el mantenimiento.

Tabla 7. Requisitos no funcionales del software

Estos requerimientos son de gran significación en la aceptación del software, debido a que representan las ventajas más visibles al usuario y repercuten en el óptimo funcionamiento y mantenimiento del sistema.

2.4.7 Historias de usuario

Las historias de usuarios (HU) constituyen un artefacto de requisitos generado por la metodología XP. De forma similar a los casos de uso que construyen otras metodologías, intentan describir los requisitos del sistema pero en muy pocas líneas, de manera clara y sin utilizar lenguaje técnico. Se redactan desde la perspectiva del cliente y el contenido que abarcan debe ser sencillo pero concreto. Las HU permiten en cualquier momento añadir o hacer modificaciones, remplazarlas o eliminarlas. Permiten también estimar el tiempo de desarrollo de la parte de la aplicación.

Las HU deben cumplir las siguientes características para que puedan realizar su función de manera correcta:

- **Independientes:** Deben ser atómicas en su definición.
- **Negociables:** Deben ser ambiguas en su enunciado, para poder debatirlas correctamente.
- **Valoradas:** Deben ser valoradas por el cliente, para poder saber cuánto aporta al valor de la aplicación y junto con la estimación convertirse en un criterio de prioridad.
- **Estimables:** Deben ser estimadas y tener su alcance bien definido.
- **Pequeñas:** Para poder realizar una estimación con cierta validez y no perder la visión de la HU, se recomienda que sean mayores de dos días y menores de dos semanas.
- **Verificables:** Este es el gran avance de las HU, que, junto con el cliente, se acuerdan criterios de aceptación que verifican si se ha cumplido con las funcionalidades descritas y esperadas.

En el proceso de desarrollo de un software, las HU resultan claves para logran una comprensión adecuada de las necesidades del cliente. Además, ayudan a estimar tiempo de desarrollo y sirven de base para realizar las pruebas de aceptación. Generan poca documentación lo que constituye una forma rápida de administrar los requerimientos y responder ante las modificaciones que estos puedan sufrir durante el ciclo de desarrollo del sistema.

Según Kent Beck cada HU recoge al menos los siguientes aspectos:

- **Número:** Posee el número asignado a la HU.
- **Nombre de HU:** Atributo que contiene el nombre de la HU.
- **Usuario:** El usuario del sistema que utiliza o protagoniza la HU.
- **Prioridad en el negocio:** Evidencia el nivel de prioridad de la HU en el negocio.
- **Riesgo de desarrollo:** Evidencia el nivel de riesgo en caso de no realizarse la HU.
- **Puntos estimados:** Este atributo no es más que una estimación hecha por el equipo de desarrollo del tiempo de duración de la HU. Cuando el valor es 1 equivale a una semana ideal de trabajo. En la metodología XP está definida una semana ideal como 5 días hábiles trabajando 40 horas, es decir, 8 horas diarias, por lo que cuando el valor de dicho atributo es 0.5 equivale a 2 días y medio de trabajo, lo que se traduce en 20 horas.
- **Puntos reales:** Igual que el parámetro anterior, pero en este caso será el tiempo real en el que se realizó la HU.
- **Descripción:** Posee una breve descripción de lo que realizará la HU.

Para la creación de las HU en la presente investigación se tomaron un conjunto de los aspectos

propuestos por Kent Beck tal como en el modelo de HU en la Tabla 8:

Historia de usuario		
Nombre: Identifica la HU en cuestión.		**Puntos estimados**: Permite estimar duración de implementación en semanas.
No.: Número sucesivo a partir de 1.	**Usuario:** Quien ejecuta la HU.	**Iteración:** Precisa la iteración a la que pertenece la HU.
Descripción: Explica en qué consiste la HU, teniendo en cuenta las acciones realizadas por el usuario y las respuestas brindadas por el sistema.		
Prioridad: Define la relevancia e impacto de la HU para el negocio de acuerdo con las necesidades del usuario.		**Nivel de complejidad:** Define la dificultad técnica que supone desarrollar la HU desde el punto de vista del programador.
Observaciones: detalle opcional propuesto por el equipo de desarrollo		

Tabla 8. Modelo de Historias de Usuarios

Para contribuir con la implementación de la herramienta educativa Simulador del Transmisor de Shannon (STX) se identificó una HU por cada requisito funcional, para un total de 30 HU.

A continuación se muestra, en la Tabla 9, la HU correspondiente al RF Codificar eficiente una cadena de símbolos a través del algoritmo Shannon-Fano. Ver restantes HU en Anexos 1.

Historia de usuario		
Nombre: Codificación Shannon Fano		**Puntos estimados:** 0,5
No.4	**Usuario:** Todos	**Iteración:** 1
Descripción: El sistema muestra en la ventana correspondiente al subsistema de codificación en las técnicas codificación eficiente las opciones para ejecutar automáticamente el algoritmo Shannon Fano a una cadena de caracteres.		
Prioridad: Alta		**Nivel de complejidad:** Media
Observaciones: Se debe haber cargado o entrado antes una cadena de caracteres.		

Tabla 9. Historias de Usuario Codificación Shannon Fano.

Una vez terminada la confección de las HU, se comienza con la creación del plan de entregas, el mismo tiene como objetivo brindarle a los desarrolladores una comprensión de la estimación de cada una de esas historias.

2.4.8 Estimación de esfuerzos por Historia de usuarios

Para desarrollar correctamente la aplicación propuesta se desarrolló la estimación de esfuerzo para cada HU identificada, lo cual permitió tener una medida de la velocidad del proyecto y brindar una guía a la cual

ajustarse. El procedimiento es sumar los puntos de estimación de todas las HU que se desarrollarán en una iteración para determinar el tiempo total que durará su implementación y de esta manera controlar la velocidad del proyecto.

Los resultados se muestran a continuación, en la Tabla 10. Cada punto de estimación está descrito en la escala de semanas.

Historia de usuario	Puntos de estimación (semanas)
HU_Entrar cadena	0,5
HU_Cargar cadena desde un fichero del disco.	0,5
HU_Seleccionar tipo de codificación	0,5
HU_Codificación Shannon Fano	0,5
HU_Codificación paso a paso Shannon Fano	1
HU_Codificación Huffman	0,5
HU_Codificación paso a paso Huffman	1
HU_Codificación Hamming	0,5
HU_Codificación paso a paso Hamming	1
HU_Codificación Códigos cíclicos	0,5
HU_Codificación paso a paso Códigos cíclicos	1
HU_Codificación Hamming/Shannon	0,5
HU_Codificación paso a paso Hamming/Shannon	1
HU_Codificación Hamming/Huffman	0,5
HU_Codificación paso a paso Hamming/Huffman	1
HU_Codificación C.Cíclicos/Shannon	0,5
HU_Codificación paso a paso C.Cíclicos /Shannon	1
HU_Codificación C.Cíclicos /Huffman	0,5
HU_Codificación paso a paso C.Cíclicos /Huffman	1
HU_Ver Ayuda	0.5
HU_Compresión RLE	0,5
HU_Compresión paso a paso RLE	0,5
HU_Compresión LZ77	0,5
HU_Compresión paso a paso LZ77	1
HU_Compresión LZ78	0,5
HU_Compresión paso a paso LZ78	0,5
HU_Compresión LZW	0,5
HU_Compresión paso a paso LZW	0,5
HU_Compresión Aritmética	0,5
HU_Compresión paso a paso Aritmética	1
Total	20

Tabla 10. Estimación de esfuerzos.

Se puede observar que la estimación de esfuerzos arrojó un resultado de 20 semanas, que equivale a 5 meses u 800 horas si se tiene en cuenta que se trabajan 8 horas diarias por días. Tal como se mencionó anteriormente, una semana equivale a 5 días laborables.

2.4.9 Plan de iteraciones

Dentro de la fase de planificación en la metodología XP se encuentran las iteraciones que son utilizadas para medir el progreso del proyecto, consiste en dividir las HU en iteraciones, ubicando en la primera iteración las HU que tienen gran importancia en la estructura, el diseño de la herramienta y mayor peso en el contenido de la misma (prioridad Alta), en la segunda iteración las que tienen prioridad Media y que también son importantes para el cliente y en la tercera iteración las de prioridad Baja.

A continuación se muestra el plan de iteraciones para el desarrollo del sistema STX en la Tabla 11:

Iteraciones	Orden de las HU a implementar	Duración
1	HU_Entrar cadena. HU_Cargar cadena desde un fichero del disco. HU_Codificación Shannon Fano. HU_Codificación Huffman. HU_Codificación Hamming HU_Codificación Códigos cíclicos HU_Codificación Hamming/Shannon. HU_Codificación Hamming/Huffman. HU_Codificación C.Cíclicos/Shannon. HU_Codificación C.Cíclicos /Huffman. HU_Compresión RLE. HU_Compresión LZ77. HU_Compresión LZ78. HU_Compresión LZW. HU_Compresión Aritmética.	8 semanas
2	HU_Codificación paso a paso Shannon Fano. HU_Codificación paso a paso Huffman. HU_Codificación paso a paso Hamming. HU_Codificación paso a paso Códigos cíclicos. HU_Codificación paso a paso Hamming/Shannon. HU_Codificación paso a paso Hamming/Huffman. HU_Codificación paso a paso C.Cíclicos /Shannon. HU_Codificación paso a paso C.Cíclicos /Huffman. HU_Compresión paso a paso RLE. HU_Compresión paso a paso LZ77. HU_Compresión paso a paso LZ78. HU_Compresión paso a paso LZW. HU_Compresión paso a paso Aritmética.	11 semanas
3	HU_ Seleccionar tipo de codificación. HU_ Ver Ayuda.	1 semana

Tabla 11. Plan de duración de iteraciones.

2.4.10 Plan de entregas

Para entregar el plan estimado de entrega se utilizan como bases las HU. Las mismas son agrupadas y ordenadas para conformar una entrega según establece el cronograma de entregas. El cliente agrupa y ordena las HU según sus prioridades.

El Plan de entregas es el resultado de una reunión entre todos los actores del proyecto. XP denomina a esta reunión “Juego de planeamiento” (“Planning game”), pero puede denominarse de la manera que sea más apropiada al tipo de cliente. Típicamente el cliente ordenará y agrupará según sus prioridades las historias de usuario. El cronograma de entregas se realiza en base a las estimaciones de tiempos de desarrollo realizadas por los desarrolladores. Luego de algunas iteraciones es recomendable realizar nuevamente una reunión con los actores del proyecto y ajustarlo si es necesario. (Joskowicz y otros, 2008).

Para ver el plan de entregas que se realizó para el desarrollo del sistema STX, a continuación se muestra la Tabla 12:

Historias de usuarios	**Iteración 1**	**Iteración 2**	**Iteración 3**
HU_Entrar cadena	Finalizado		
HU_Cargar cadena desde un fichero del disco.	Finalizado		
HU_Seleccionar tipo de codificación	Finalizado		
HU_Codificación Shannon Fano	Finalizado		
HU_Codificación paso a paso Shannon Fano	(-)	Finalizado	
HU_Codificación Huffman	Finalizado		
HU_Codificación paso a paso Huffman	(-)	Finalizado	
HU_Codificación Hamming	Finalizado		
HU_Codificación paso a paso Hamming	(-)	Finalizado	
HU_Codificación Códigos Cíclicos	Finalizado		
HU_Codificación paso a paso Códigos Cíclicos	(-)	Finalizado	
HU_Codificación Hamming/Shannon	(-)	50%	Finalizado

HU_Codificación paso a paso Hamming/Shannon	(-)	30%	Finalizado
HU_Codificación Hamming/Huffman	(-)	50%	Finalizado
HU_Codificación paso a paso Hamming/Huffman	(-)	30%	Finalizado
HU_Codificación C.Cíclicos/Shannon	(-)	50%	Finalizado
HU_Codificación paso a paso C.Cíclicos /Shannon	(-)	30%	Finalizado
HU_Codificación C.Cíclicos /Huffman	(-)	50%	Finalizado
HU_Codificación paso a paso C.Cíclicos /Huffman	(-)	30%	Finalizado
HU_Ver Ayuda	(-)	(-)	Finalizado
HU_Compresión RLE	Finalizado		
HU_Compresión paso a paso RLE	(-)	Finalizado	
HU_Compresión LZ77	Finalizado		
HU_Compresión paso a paso LZ77	(-)	Finalizado	
HU_Compresión LZ78	Finalizado		
HU_Compresión paso a paso LZ78	(-)	Finalizado	
HU_Compresión LZW	Finalizado		
HU_Compresión paso a paso LZW	(-)	Finalizado	
HU_Compresión Aritmética	Finalizado		
HU_Compresión paso a paso Aritmética	(-)	Finalizado	

Tabla 12. Plan de Entregas

Leyenda

- (-): tarea que por cronograma no se desarrolló en la iteración.
- %: porcentaje de culminación de la tarea. La tarea depende de dos algoritmos y ya se culminó en la iteración anterior.
- Finalizado: muestra y entrega de la funcionalidad implementada.

El desarrollo de las tres iteraciones del sistema STX abarcó un período de tiempo desde enero hasta mayo del 2014.

2.5 Diseño de la solución

La segunda fase que plantea la metodología XP es la fase de diseño. El diseño aplicado al desarrollo de software, es una actividad cuyo objetivo es establecer patrones que ayuden a modelar y entender los objetos, procesos y servicios del ciclo de vida de un sistema.

La metodología XP propone un diseño simple y entendible para los clientes. No requiere de la modelación mediante UML, sin embargo, esta alternativa puede utilizarse para mejorar la comunicación entre las partes involucradas.

2.5.1 Tarjetas CRC (Clase, Responsabilidad y Colaboración)

Después de haber dividido las HU en tareas, deben ser transformadas en código, para esto se llevan a cabo reuniones donde se presentan las tarjetas CRC (Clase Responsabilidad Colaboración), que dan una idea de cuales clases se deben implementar.

Para poder diseñar el sistema como un equipo se debe cumplir con tres principios: Cargo o Clase, Responsabilidad y Colaboración (CRC). Las tarjetas CRC permiten desprenderse del método de trabajo basado en procedimientos y trabajar con una metodología basada en objetos, contribuyendo a que el equipo completo contribuya en la tarea del diseño.

Una tarjeta CRC representa un objeto. El nombre de la clase se coloca a modo de título en la tarjeta, las responsabilidades se colocan a la izquierda, y las clases que se implican en cada responsabilidad a la derecha, en la misma línea que su requerimiento correspondiente.

El modelo de Tarjetas CRC se muestra en la Tabla 13.

Clase: Nombre de la clase que se está modelando**.**	
Responsabilidades	**Colaboraciones**
Es una descripción del propósito de la clase.	Indica con cuales clases tiene relación, para cumplir su responsabilidad.

Tabla 13. Modelo de Tarjetas CRC.

A continuación se muestran ejemplos de Tarjetas CRC elaboradas por el equipo de desarrollo, ver tablas

14, 15 y 16. Para ver las restantes tarjetas CRC, consultar Anexos 2:

Tarjeta CRC	
Clase: AlgoritmoCodificacion.java	
Responsabilidades: Clase padre de todos los algoritmos.	**Colaboraciones:**

Tabla 14. Tarjeta CRC AlgoritmoCodificacion.java.

Tarjeta CRC	
Clase: InputStringCodif.java	
Responsabilidades: Es la clase que se encarga de mostrar la cadena de entrada correspondiente al paso a paso de los algoritmos de codificación.	**Colaboraciones:** import Codificacion.CRC.CRC; import Codificacion.Hamming.Hamming; import Codificacion.Huffman.Huffman; import Codificacion.Shannon_Fano.ShannonFano; import Comun.Util; import java.awt.Color; import java.awt.Dimension; import java.awt.Font; import java.awt.Graphics; import java.awt.Graphics2D; import java.awt.RenderingHints; import java.awt.font.FontRenderContext; import java.awt.geom.AffineTransform; import javax.swing.JPanel;

Tabla 15. Tarjeta CRC InputStringCodif.java

Tarjeta CRC	
Clase: CRC.java	
Responsabilidades: Codificar una cadena mediante el algoritmo código cíclico.	**Colaboraciones:** import Codificacion.AlgoritmoCodificacion;

Tabla 16. Tarjeta CRC de la clase CRC.java.

2.5.2 Estándares de codificación

Al realizar las tarjetas CRC se tiene en cuenta, los diferentes estándares existentes para la codificación ya que estos comprenden aspectos de la generación de código que si se usan correctamente por los programadores, se logra mayor coordinación entre ellos. Al incorporarse por parte del proyecto de software, código fuente previo, o bien cuando se realice el mantenimiento de un sistema de software creado anteriormente, el estándar de codificación debería establecer cómo operar con la base de código

existente.

Un código que sea legible permite poder modificar o añadir nuevas características, depurar errores o mejorar el rendimiento.

Las convenciones utilizadas en la implementación del sistema fueron:

- Pascal: El primer carácter de cada palabra es en mayúscula y el resto en minúscula. (Toala, 2011).
- Camel: La primera letra en el identificador está en minúscula y la primera letra de cada subsiguiente palabra concatenada en mayúscula. (Toala, 2011).

Para la implementación de la aplicación se tuvieron en cuenta las siguientes reglas de codificación:

- Utilización de nombres descriptivos que ayuden a una mejor compresión.
- Los nombres de las clases o interfaces tendrán la primera letra de cada palabra en mayúscula, ejemplo: Codificacion.java.
- Los nombres de los métodos reflejan la acción a realizar y siempre comenzando con letra minúscula, en caso de ser un nombre compuesto, todas las palabras que lo componen menos la primera se escriben con letra inicial mayúscula, ejemplo: codificaciónEficiente() y validarConfiguracion().
- Cada símbolo que pertenezca al nombre de la constante se escribirá en mayúscula y en caso de ser un nombre compuesto, cada palabra se separará por un guión bajo "_", ejemplo: MIN_VALUE_K y MAX_VALUE_K.
- Cada funcionalidad debe tener un comentario de su funcionamiento.

2.5.3 Estándares en la interfaz de la aplicación.

Son elementos visuales relacionados entre sí, estos estándares permiten que la aplicación sea lo más amigable posible, para que cuando el usuario interactúe con ella se sienta más cómodo. Para realizar un buen desarrollo de la interfaz de usuario se tienen en cuenta elementos como el uso de contenido visual (gráficas), debido a la gran ayuda que brindan haciéndole llegar más fácil y cómoda la información al usuario. Los botones, opciones, menú, etc. tienen un tamaño y color adecuado en combinación con la herramienta en general.

2.6 Validación del diseño

Para la validación del diseño se emplearon las métricas Tamaño Operacional de Clase (TOC) y Relaciones entre clases (RC), diseñadas para evaluar los siguientes atributos de calidad:

- Responsabilidad: Consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto de la problemática propuesta.
- Complejidad de implementación: Consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.
- Reutilización: Consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.
- Acoplamiento: Consiste en el grado de dependencia o interconexión de una clase o estructura de clase con otras, está muy ligada a la característica de Reutilización.
- Complejidad del mantenimiento: Consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta pero fuertemente en los costes y la planificación del proyecto.
- Cantidad de pruebas: Consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad (unidad) del producto (componente, módulo, clase, conjunto de clases, etc.) diseñado.

2.6.1 Tamaño Operacional de Clase (TOC)

Esta métrica se determina por el número total de operaciones que están encapsuladas dentro de la clase. En cuanto menor sea el valor medio para el tamaño más probable, menor será la responsabilidad y complejidad de las clases y sin embargo mayor nivel de reutilización tendrán las mismas.

Para un mejor entendimiento de la utilización de esta métrica, a continuación se muestra la Tabla 17:

Atributo	Categoría	Criterio
Responsabilidad	Baja	<= Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	> 2* Promedio
Complejidad de Implementación	Baja	<= Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	> 2* Promedio

Reutilización	Baja	> 2* Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	<=Promedio

Tabla 17. Rango de valores para la evaluación técnica de los atributos de calidad relacionados con la métrica.

Después de realizar un análisis de los resultados arrojados por la evaluación bajo los instrumentos de medición de la métrica TOC, se demuestra que se alcanzaron valores aceptables para cada uno de los atributos de calidad evaluados. A continuación se muestran los resultados en la figura 9:

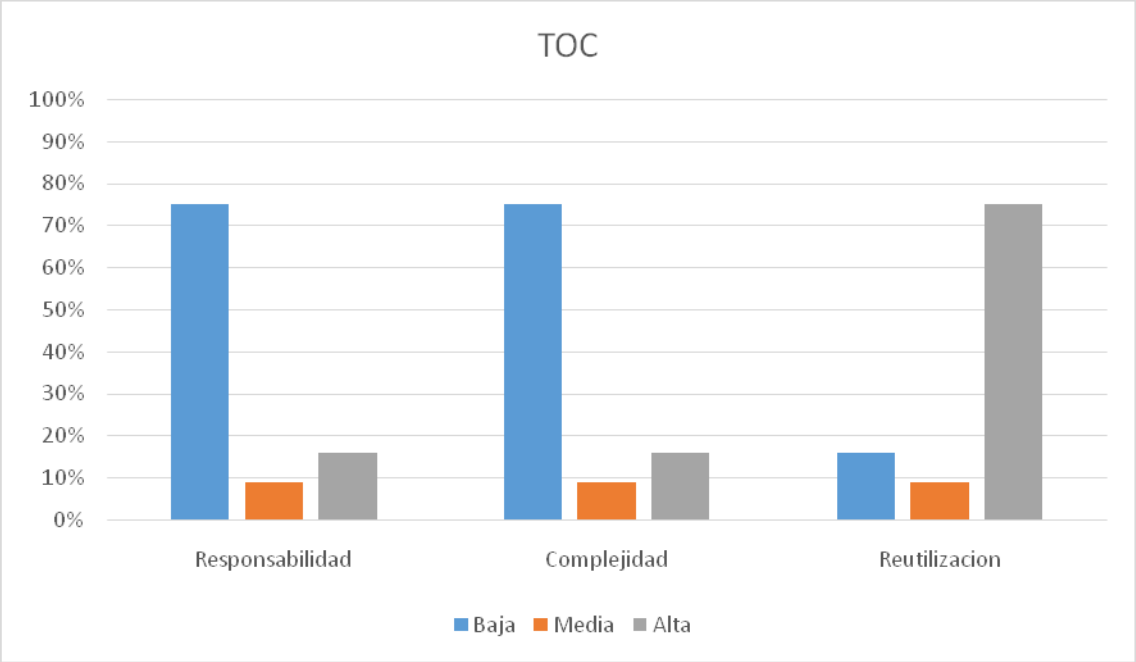

Figura 9 Resultados TOC.

2.6.2 Métrica Relaciones entre Clases (RC)

Relaciones entre clases (RC): Está dado por el número de relaciones de uso de una clase con otras. Ver más detalles sobre la utilización de la métrica en las tablas 19 y 20 que se muestran a continuación:

Atributo que afecta	Modo en que lo afecta
Acoplamiento	Un aumento del RC implica un aumento del Acoplamiento de la clase
Complejidad del mantenimiento	Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.
Reutilización	Un aumento del RC implica una disminución en el grado de reutilización de la clase

Cantidad de pruebas	Un aumento del RC implica un aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

Tabla 18. Métricas RC.

Atributo	Categoría	Criterio
Acoplamiento	Baja	<= Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	> 2* Promedio
Complejidad del mantenimiento	Baja	<= Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	> 2* Promedio
Reutilización	Baja	> 2* Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	<=Promedio
Cantidad de pruebas		<=Promedio
		Entre Promedio y 2* Promedio
		> 2* Promedio

Tabla 19. Rango de valores para la evaluación técnica de los atributos de calidad relacionados con la métrica

Esta métrica se determina por la cantidad de relaciones existentes entre las clases contenidas en el diseño. El número de dependencias es directamente proporcional al nivel de acoplamiento, a la complejidad del mantenimiento y a la cantidad de pruebas así como inversamente proporcional al grado de reutilización de las mismas. La aplicación de esta métrica evidenció que el diseño del sistema es aceptable con respecto a todos los atributos de calidad afectados, arrojando resultados positivos.

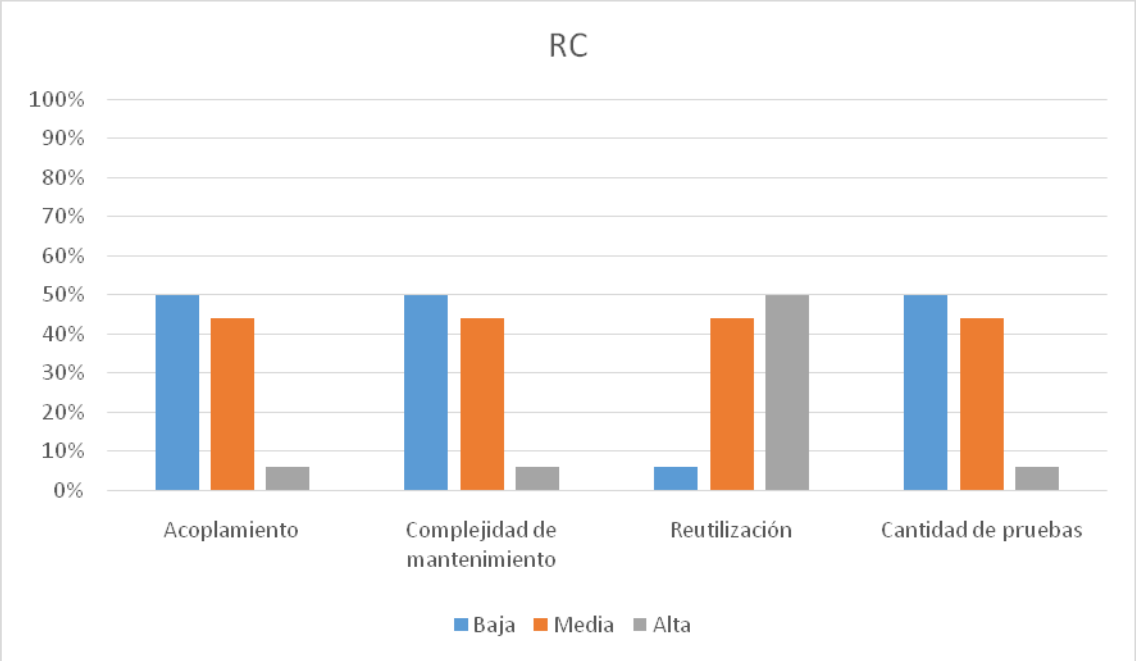

Figura 10. Resultados RC.

2.7 Conclusiones del capítulo

Luego de finalizado el capítulo 2 se concluye lo siguiente:

- Como parte del desarrollo de la fase planificación de la metodología XP, se realizó una descripción de cada uno de los artefactos generados en el transcurso de la misma, se realizaron un total de 30 HU que describen los requisitos funcionales que se obtuvieron para el sistema STX.
- Como parte de la fase de diseño de la metodología XP, se realizaron las tarjetas CRC para la descripción de las clases y se validó el diseño a partir de las métricas TOC y RC, las cuales evidenciaron resultados satisfactorios de acuerdo a los atributos de calidad evaluados.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS

3.1 Introducción

Uno de los pilares de la Programación Extrema es el proceso de pruebas. XP anima a probar constantemente tanto como sea posible. Esto permite aumentar la calidad de los sistemas reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección. También permite aumentar la seguridad de evitar efectos colaterales no deseados a la hora de realizar modificaciones y refactorizaciones. XP divide las pruebas del sistema en dos grupos: pruebas unitarias, encargadas de verificar el código y diseñada por los programadores para tener una máxima claridad sobre lo que se va a programar antes de hacerlo, y pruebas de aceptación destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida diseñadas por el cliente final.

3.2 Prueba

Una de las últimas fases del ciclo de vida antes de entregar un programa para su explotación, es la fase de pruebas. El desarrollo del software implica una serie de actividades de producción donde es común la posibilidad de que aparezca una falla humana. Los errores pueden presentarse desde el inicio del proceso con la especificación errónea de los requisitos; de igual forma en los posteriores pasos del diseño y desarrollo. Debido a la imposibilidad humana de trabajar y comunicarse de forma perfecta, el desarrollo del software ha de ir acompañado de una actividad que garantice la calidad. La prueba de software es un elemento crítico para la garantía de la calidad y representa una revisión final de las especificaciones del diseño y de la codificación.

3.3 Métodos de prueba

Las pruebas son acciones en las cuales el sistema es ejecutado bajo condiciones o requerimientos determinados. Los resultados son chequeados y registrados. Las pruebas verifican los resultados de la implementación del sistema. El modelo de prueba indica cómo han de ser probados aspectos determinados del producto o software. En lo que sigue se intentará una pequeña organización de acuerdo a sus características.

3.3.1 Tipos de pruebas:

Pruebas de cajas blancas o estructurales: son aquellas pruebas que se les hace a los métodos o procedimientos, los cuales deben ser unidades estructurales del programa encargados de una tarea específica. Se caracterizan por realizarse individualmente, o sea, una prueba hecha a una función es independiente a la realizada a otra.

Pruebas de cajas negras o funcionales: Las pruebas de caja negra son aquellas que se realizan sobre la interfaz del software. Además, no requieren el conocimiento de la estructura interna del programa para su puesta en marcha. Estas pruebas no son una alternativa a las técnicas de prueba de caja blanca, sino un enfoque complementario. Los casos de prueba intentan demostrar que la entrada se acepta de forma adecuada y que se produce una salida correcta. Las pruebas de caja negra intentan hallar errores tales como:

- Funciones incorrectas o inexistentes.
- Errores de interface.
- Errores de iniciación, comienzo o finalización.
- Errores de rendimiento. (Prezi, 2013).

3.4 Diseño y ejecución de las pruebas de software

La metodología XP plantea la necesidad de realizar pruebas unitarias y de aceptación para una correcta validación del sistema siendo estos los artefactos generados necesarios para comprobar y asegurar que la herramienta desarrollada cumple con las necesidades y expectativas del cliente.

A continuación se muestra el resultado de aplicar dichas pruebas.

Pruebas de caja blanca o pruebas estructurales

Las pruebas de caja blanca o pruebas estructurales son la forma de probar el correcto funcionamiento del código de un sistema informático. Seguidamente se describirá la realización de las pruebas, y las técnicas utilizadas: camino básico, cálculo de la complejidad ciclomática y extracción de los caminos independientes. Además se mostrarán los casos de pruebas elaborados y el análisis de los resultados obtenidos.

A continuación se muestra el código del método **run()** de la clase **ModuloCodificacion.java** al cual se le

aplico la técnica de complejidad ciclomática:

```
public boolean run() {
    boolean result = validarConfiguracion();
    if (result) {
        limpiarTablas();
        int a = JComboBox_TipoCodif.getSelectedIndex();
        switch (a) {
            case 0:
                crearModelos(false);
                codificacionEficiente();
                codificacionRedundante(false);
                break;
            case 1:
                codificacionEficiente();
                break;
            case 2:
                crearModelos(true);
                codificacionRedundante(true);
                break;
        }
    }
    return result;
}
```

Figura 11. Método run()

Para el cálculo de la complejidad ciclomática es necesario representar el grafo de flujo asociado al código antes presentado a través de nodos, aristas y regiones, quedando como se muestra en la Figura 12.

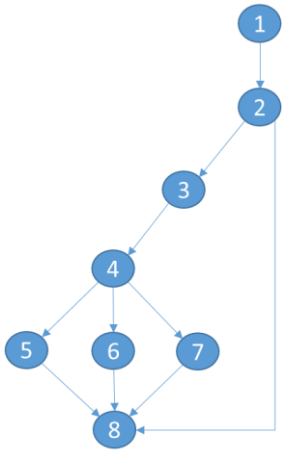

Figura 12 Grafo de flujo asociado al método

Una vez construido el grafo de flujo asociado al procedimiento anterior se determina la complejidad ciclomática, el cálculo es necesario efectuarlo mediante tres vías o fórmulas, se debe utilizar el mismo grafo en cada caso:

Fórmula 1 .V (G) = (A – N) + 2

- Siendo “A” la cantidad total de aristas y “N” la cantidad total de nodos.

Resultado.

V (G) = 10 – 8 + 2 = 4

Fórmula 2 .V (G) = P + 1

- Siendo “P” la cantidad total de nodos predicados (son los nodos de los cuales parten dos o más aristas).

Resultado.

V (G) = 1 + 3 = 4

Fórmula 3 .V (G) = R

- Siendo “R” la cantidad total de regiones, se incluye el área exterior del grafo, contando como una región más.

Resultado.

V (G) = 3 + 1 = 4

Seguidamente es necesario especificar los caminos básicos que puede tomar el algoritmo durante su ejecución. En estas representaciones se subrayan los elementos de cada camino que los hacen independientes a los demás.

Camino básico # 1: 1-2-3-4-5-8

Camino básico # 2: 1-2-3-4-6-8

Camino básico # 3: 1-2-3-4-7-8

Camino básico # 4: 1-2-8

Por su parte la técnica de camino básico desarrollada de forma automatizada con la ayuda del Junit a las clases Modulo_Codificacion.java y Modulo_Compresion.java arrojaron los siguientes resultados:

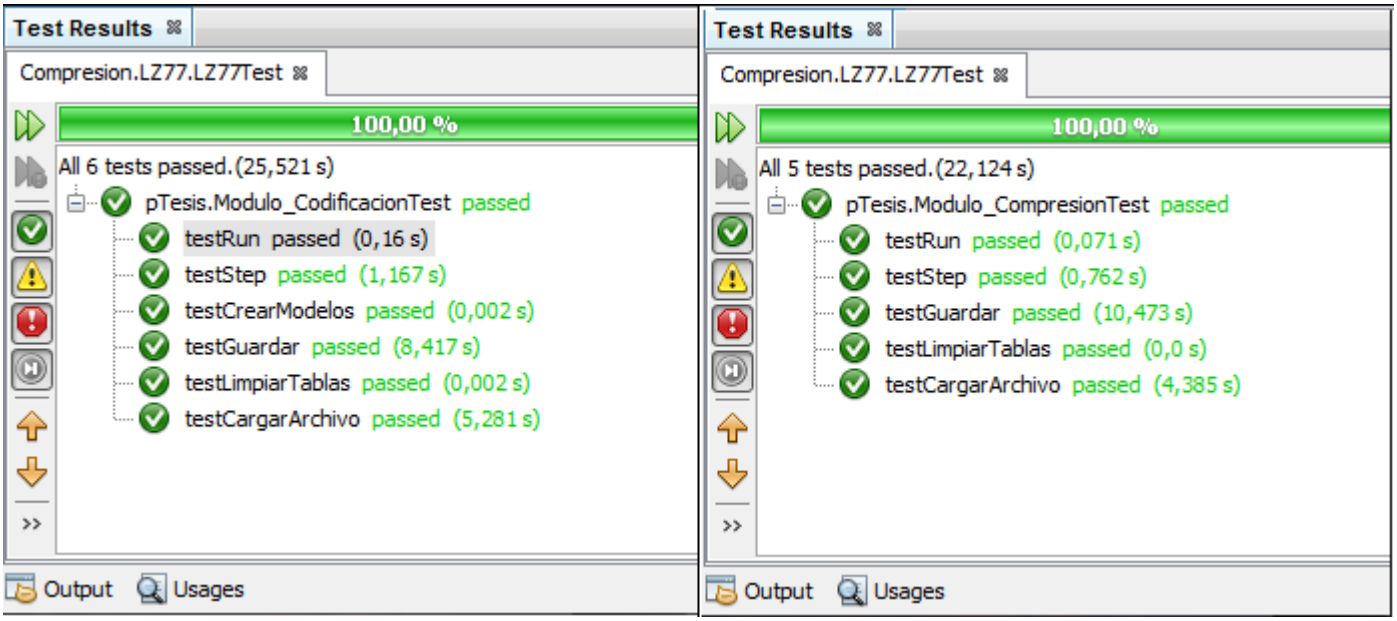

Figura 12 A la izquierda la clase Modulo_CodificacionTest y a la derecha Modulo_CompresionTest

Pruebas de caja negra o pruebas funcionales

Las pruebas de caja negra o estructurales son una forma de comprobar que las entradas al sistema estén lo más acorde posible en relación a lo que se espera. Para eso se realizaron casos de pruebas que intentan demostrar que la entrada se acepta de forma adecuada y que se produce una salida correcta.

Casos de prueba

La aplicación de casos de pruebas apropiados es esencial para la validación del sistema construido. El objetivo del diseño de casos de prueba es crear un conjunto de casos de prueba que sean efectivos descubriendo defectos en los programas y muestren que el sistema satisface sus requerimientos. Para diseñar un caso de prueba, se selecciona una característica del sistema o componente que ese está probando.

A continuación se muestran los casos de prueba aplicados a las historias de usuario: Entrar cadena y Codificación eficiente, el resto se especifican en el Anexo 3.

Caso de prueba funcional
Nombre: Entrar cadena

Descripción: Debe permitir insertar un texto en el área de texto que aparece en la ventana.
Condiciones de ejecución: Se debe teclear un texto en el área de texto que aparece en la ventana.
Resultados Esperados: El usuario puede seguir introduciendo caracteres.
Evaluación de la Prueba: Prueba satisfactoria
Caso de prueba funcional
Nombre: Codificación eficiente
Descripción: Debe permitir codificar eficientemente una cadena de símbolos.
Condiciones de ejecución: Debe permitir codificar eficientemente una cadena de símbolos.
Resultados Esperados: Se muestra el resultado de la codificación en la tabla que aparece en la parte inferior izquierda de la ventana y se debe activar el botón.
Evaluación de la Prueba: Prueba satisfactoria

Figura 13. Resultados de la generación de casos de prueba

Pruebas de aceptación

Las pruebas de aceptación se realizaron con el metodólogo del Departamento de Técnicas de Programación central. Ver carta de aceptación en el Anexo 4.

3.5 Conclusiones del capítulo

Luego de desarrollado el capitulo 3 se concluye lo siguiente:

- Durante el desarrollo de la aplicación se realizaron las pruebas necesarias para comprobar el cumplimiento y la calidad de sus funcionalidades planteadas por el cliente mediante las HU. Por cada HU se realizó un caso de prueba.
- Fueron detectadas un total de 7 no conformidades en las 3 iteraciones realizadas, recogido de la siguiente manera: en la primera iteración se detectaron 5 no conformidades, 2 significativas, 1 no significativas y 2 recomendaciones, en la segunda iteración se detectaron 1 no conformidad del tipo recomendación, y en la tercera iteración 1 no conformidad no significativa. Las no conformidades no significativas detectadas se centraron más en los errores ortográficos como son: comas, puntos, paréntesis, mayúsculas o minúsculas, ya que la documentación de esta aplicación se utilizará en el estudio de los estudiantes.
- El sistema quedó validado mediante la carta de aceptación emitida por el metodólogo de la asignatura a nivel central.

CONCLUSIONES GENERALES

Una vez terminado el presente trabajo se concluye lo siguiente:

- Fueron estudiaros diferentes conceptos referentes a los sistemas de transmisión de datos y sus procesos, que apoyaron la fundamentación teórica de la presente investigación.
- De la búsqueda realizada de sistemas que permitan representar el Esquema Ampliado del Transmisor de Shannon se obtuvo que no existe ningún sistema para tal fin y por ello se hace necesario desarrollar un sistema nuevo.
- Del análisis de sistemas existentes para la simulación de procesos similares, el VirtualNet desarrollado en la UCI, como herramienta educativa de apoyo en la impartición de los contenidos Direccionamiento y Enrutamiento de la asignatura Teleinformática fue el que más aportó.
- Del estudio de las herramientas y metodologías realizado se seleccionaron para el desarrollo del sistema: como metodología de desarrollo XP, como lenguaje de modelado UML, como herramienta case para el modelado Visual Paradigm for UML v8.0, como entorno de desarrollo el NetBeans v7.4 y como lenguaje de programación Java.
- Durante el desarrollo de la aplicación se realizaron un total de 30 HU, que fueron diseñadas y validadas a través de las métricas TOC y RC, las cuales evidenciaron resultados satisfactorios de acuerdo a los atributos de calidad evaluados.
- Para la validación del sistema se realizaron pruebas de caja blanca y pruebas de caja negra que arrojaron resultados satisfactorios.
- El sistema quedó validado mediante la carta de aceptación emitida por el metodólogo de la asignatura a nivel central.

RECOMENDACIONES

Una vez terminado el presente trabajo se recomienda lo siguiente:

- Desarrollar en futuras versiones del sistema STX los subsistemas Cifrado, Multiplexión y Modulación.
- Introducir en futuras versiones del sistema STX un funcionamiento en cadena, de manera que la salida de cada subsistema se convierta en la entrada del subsistema siguiente.
- Validar que el polinomio generador en la codificación redundante por el método de códigos cíclicos sea correcto.
- Representar en el sistema STX un entorno real de transmisión tal como el de la UCI con codificación de línea.