**INSTITUTO SUPERIOR POLITÉCNICO**

**“JOSÉ ANTONIO ECHEVERRÍA”**

logo

**Descripción: E:\CUJAE\logoCujae\logotipo.png**

**FACULTAD DE INGENIERÍA INFORMÁTICA**

**“Desarrollo de componentes para el filtrado digital de señales”**

**Trabajo de diploma para optar por el título de Ingeniería en Informática**

**Presentado por:**

Anailec Venero Perdomo (avenero@ceis.cujae.edu.cu)

**Tutores:**

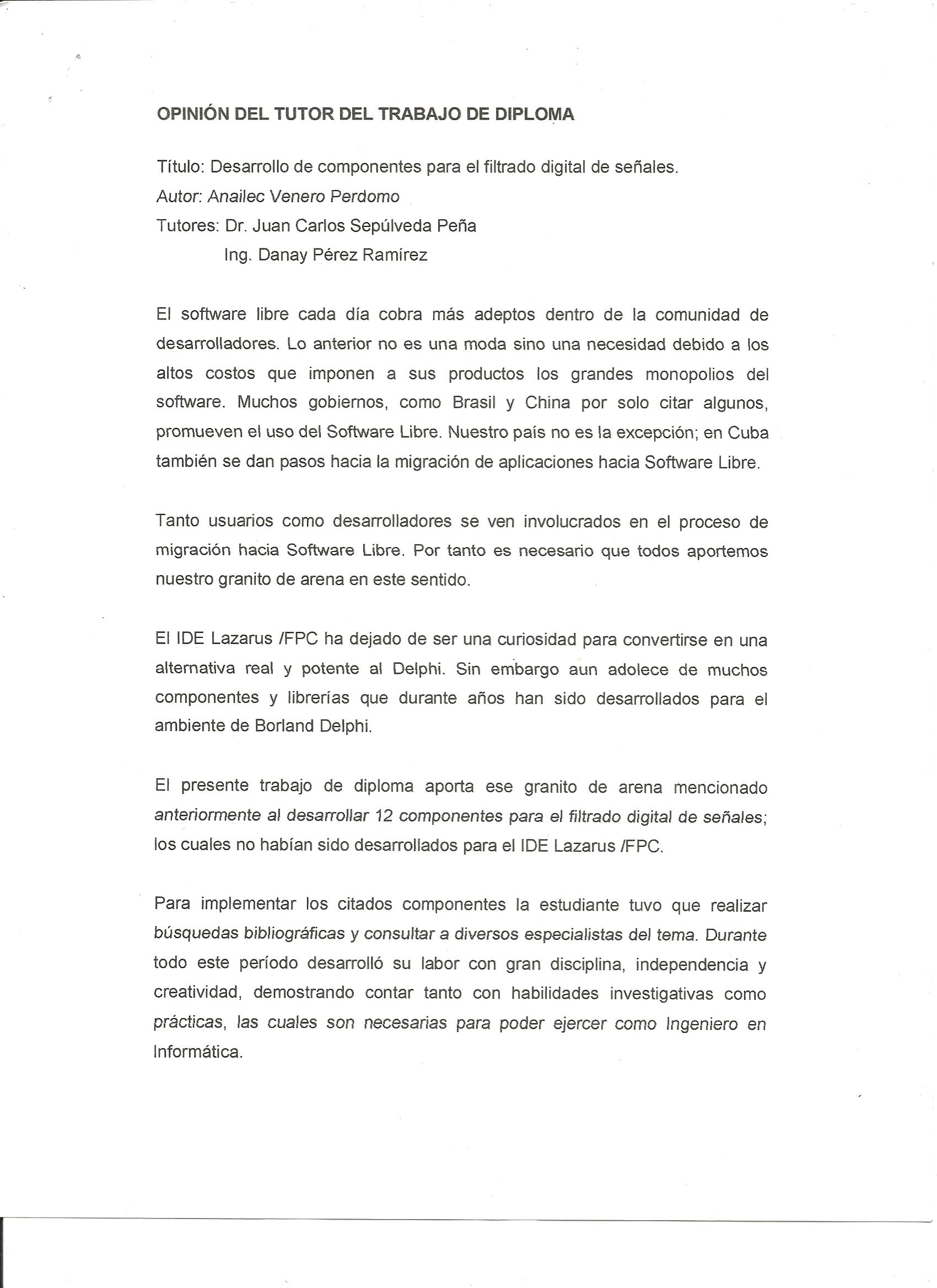
Dr. Juan Carlos Sepúlveda (jcarlos@ceis.cujae.edu.cu)

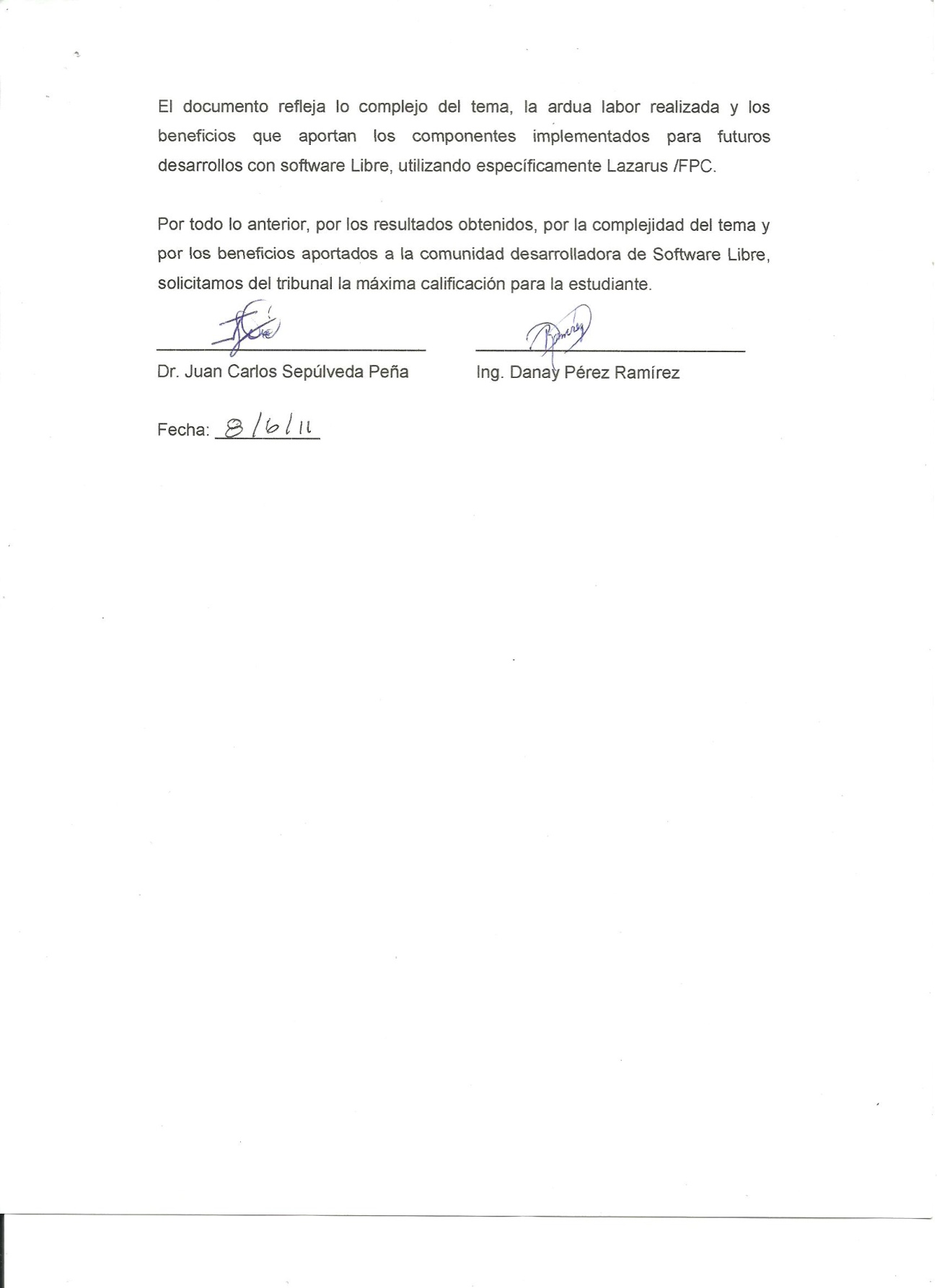
Ing. Danay Pérez Ramírez (dperezr@ceis.cujae.edu.cu)

**Ciudad de la Habana, Cuba**

**Junio del 2011**

****





**Agradecimientos**

**Mi más sincero agradecimiento:**

A todos los que de una forma u otra han contribuido a la realización de este trabajo especialmente a:

**Nuestra revolución y su partido:**

Por hacer posible la formación de nuevos profesionales al nivel que exige el desarrollo científico técnico de nuestro país.

**A mis padres:**

Que con su empeño y sacrificio han estado presentes en mi educación.

**A mi Abuela y hermana:**

Por ser una parte indisoluble de mi misma.

**A mi familia:**

Por su apoyo incondicional.

**A la Ing. Danay Pérez Ramírez y el Dr. Juan Carlos Sepúlveda:**

Tutores de este trabajo, que con su mano firme y acertadas orientaciones supieron guiarme a la culminación del presente.

**A todos los profesores**:

El respeto y agradecimiento merecido, por contribuir a mi formación integral como profesional.

**A mis compañeros de estudio y amigos**:

Por el apoyo brindado en este tiempo.

**A todos muchas gracias.**

**“A mis padres y familia por ser mi inspiración cada día para seguir adelante”**

**Resumen**

La tecnología basada en componentes presenta numerosas ventajas con respecto al paradigma basado en objetos, evitando muchos de sus problemas.

En el año 2006 el Centro Nacional para la producción de Animales de Laboratorio (CENPALAB), el Centro de Estudios de Ingeniería y Sistemas (CEIS), y una empresa cubano-brasileña TechAgro, se unen para desarrollar el “Sistema para la Gestión de Maquinarias Agrícolas” (SGMA) y un conjunto de componentes para usar en la agricultura de precisión. Este sistema junto con lo*s* componentes fueron desarrollados usando el Entorno de Desarrollo Integrado Borland Developer Studio 2006 de Delphi.

Actualmente surge la necesidad de desarrollar componentes con funcionalidades similares a los que se desarrollaron con Delphi, pero sobre plataformas de Software Libre debido a las conocidas ventajas que esta ofrece, en lo económico, político y tecnológico.

El presente trabajo de diploma tiene como objetivo principal crear doce componentes, que permitan el filtrado digital de señales, para ser utilizados en la migración del sistema (SGMA), y por cualquiera que los necesite.

Estos componentes se desarrollan sobre el IDE Lazarus/Free Pascal, ya que es altamente compatible con Delphi, y es además multiplataforma. Se utiliza la Metodología ágil Extreme Programming (XP) por contarse con requisitos cambiantes durante el desarrollo, un reducido número de desarrolladores y tener al cliente disponible todo el tiempo.

**Summary**

The technology based on components presents numerous advantages with regard to the paradigm based on objects, avoiding many of its problems.

In the year 2006 the National Center for the production of Animals of Laboratory (CENPALAB), the Center of Studies of Engineering and Systems (CEIS), and a Cuban-Brazilian company TechAgro, come together to develop the " Management System for Equipment Agricultural” (SGMA) and a group of components to use in the agriculture of precision. This system together with the components were developed using the Integrated Development Environment of Borland Developer Studio 2006 Delphi.

At present there is a need to develop components with similar functionalities to those developed with Delphi but on Free Software platforms due to the known advantages this offers, in the economic, political and technological.

Presently diploma main objective is to create twelve components that permit digital filtering of signals for use in the migration of system (SGMA), and for anyone who needs them.

These components are developed on the IDE Lazarus / Free Pascal, which is highly compatible with Delphi, and is also cross platform. Agile methodology is used Extreme Programming (XP) for having a changing requirements during the development, small number of developers and customers to have available all the time.

# Índice

[Introducción 1](#_Toc295310069)

[Capítulo 1 Fundamentos Teóricos 5](#_Toc295310070)

[1.1 Introducción 5](#_Toc295310071)

[1.2 Procesamiento de Señales 5](#_Toc295310072)

[1.2.1 Sistemas Analógicos y Digitales 5](#_Toc295310073)

[1.2.2 Procesamiento Digital de Señales 6](#_Toc295310074)

[1.3 Filtros 16](#_Toc295310075)

[1.3.1 Algunas clasificaciones de los Filtros digitales 19](#_Toc295310076)

[1.3.2 Aproximación de Butterworth 22](#_Toc295310077)

[1.3.3 Aproximación de Chebyshev 22](#_Toc295310078)

[1.3.4 Aproximación de Bessel 23](#_Toc295310079)

[1.4 Tendencias y Tecnologías Actuales 24](#_Toc295310080)

[1.4.1 Comparativa de Free Pascal frente a otros Lenguajes 25](#_Toc295310081)

[1.4.2 Lazarus 28](#_Toc295310082)

[1.4.3 Programación Basada en Componentes 30](#_Toc295310083)

[1.4.4 Necesidad e Importancia de crear Componente sobre Software Libre 33](#_Toc295310084)

[1.4.5 Herramienta Case de modelación utilizada 34](#_Toc295310085)

[1.4.6 Metodología utilizada: Extreme Programming (XP) 36](#_Toc295310086)

[1.5 Conclusiones 39](#_Toc295310087)

[Capítulo 2 Exploración y Planificación 41](#_Toc295310088)

[2.1 Introducción 41](#_Toc295310089)

[2.2 Elaboración de la metáfora 41](#_Toc295310090)

[2.3 Exploración 41](#_Toc295310091)

[2.3.1 Historias de usuarios 42](#_Toc295310092)

[2.3.2 Tareas de programación 44](#_Toc295310093)

[2.3.3 Casos de Prueba 52](#_Toc295310094)

[2.4 Planificación 54](#_Toc295310095)

[2.4.1 Estimaciones 54](#_Toc295310096)

[2.4.2 Estimar esfuerzo total 55](#_Toc295310097)

[2.4.3 Estimar velocidad del equipo 55](#_Toc295310098)

[2.4.4 Calcular tiempo real 56](#_Toc295310099)

[2.4.5 Calcular puntos de historia de usuario por entrega 56](#_Toc295310100)

[2.4.6 Distribución de historias de usuario por iteración 56](#_Toc295310101)

[2.4.7 Cálculo de puntos de historia de usuario por iteración 57](#_Toc295310102)

[2.5 Plan de Entrega 59](#_Toc295310103)

[2.6 Conclusiones 60](#_Toc295310104)

[Capítulo 3 Desarrollo de los componentes 62](#_Toc295310105)

[3.1 Introducción 62](#_Toc295310106)

[3.2 Iteración I 62](#_Toc295310107)

[3.2.1 Plan de Iteración 62](#_Toc295310108)

[3.2.2 Diseño Preliminar 63](#_Toc295310109)

[3.2.3 Casos de Prueba 64](#_Toc295310110)

[3.2.4 Diseño Final 64](#_Toc295310111)

[3.3 Iteración II 65](#_Toc295310112)

[3.3.1 Plan de Iteración 66](#_Toc295310113)

[3.3.2 Diseño Preliminar 67](#_Toc295310114)

[3.3.3 Casos de Prueba 68](#_Toc295310115)

[3.3.4 Diseño Final 69](#_Toc295310116)

[3.4 Iteración III 69](#_Toc295310117)

[3.4.1 Plan de Iteración 69](#_Toc295310118)

[3.4.2 Diseño Preliminar 70](#_Toc295310119)

[3.4.3 Casos de Prueba 71](#_Toc295310120)

[3.4.4 Diseño Final 72](#_Toc295310121)

[3.5 Iteración IV 72](#_Toc295310122)

[3.5.1 Plan de Iteración 72](#_Toc295310123)

[3.5.2 Diseño Preliminar 73](#_Toc295310124)

[3.5.3 Casos de Prueba 74](#_Toc295310125)

[3.5.4 Diseño Final 74](#_Toc295310126)

[3.6 Producción 75](#_Toc295310127)

[3.7 Mantenimiento 75](#_Toc295310128)

[3.8 Muerte 75](#_Toc295310129)

[3.9 Conclusiones 76](#_Toc295310130)

[Conclusiones 77](#_Toc295310131)

[Recomendaciones 78](#_Toc295310132)

[Referencias Bibliográficas 79](#_Toc295310133)

[Bibliografía 82](#_Toc295310134)

[Glosario de Términos 83](#_Toc295310135)

[Anexo 1 Tablas de Historias de Usuario 86](#_Toc295310136)

[Anexo 2 Tablas de Casos de Prueba 89](#_Toc295310137)

[Anexo 3 Vistas de algunos ejemplos 91](#_Toc295310138)

**Índice de Tablas**

[Tabla 1 Historias de usuario 42](#_Toc295295658)

[Tabla 2 Tareas de programación pertenecientes a la Historia de Usuario 1 “Especificar el orden del filtro” 45](#_Toc295295659)

[Tabla 3 Tareas de programación pertenecientes a la Historia de Usuario 2 “Especificar la frecuencia de corte del filtro” 45](#_Toc295295660)

[Tabla 4 Tareas de programación pertenecientes a la Historia de Usuario 3 “Diseñar el filtro FilterIIRLPButterworth” 45](#_Toc295295661)

[Tabla 5 Tareas de programación pertenecientes a la Historia de Usuario 4 “Filtrar señal a través del filtro FilterIIRLPButterworth” 45](#_Toc295295662)

[Tabla 6 Tareas de programación pertenecientes a la Historia de Usuario 5 “Diseñar el filtro FilterIIRLPChebyshev” 46](#_Toc295295663)

[Tabla 7 Tareas de programación pertenecientes a la Historia de Usuario 6 “Filtrar señal a través del filtro FilterIIRLPChebyshev” 46](#_Toc295295664)

[Tabla 8 Tareas de programación pertenecientes a la Historia de Usuario 7 “Diseñar el filtro FilterIIRLPBessel” 46](#_Toc295295665)

[Tabla 9 Tareas de programación pertenecientes a la Historia de Usuario 8 “Filtrar señal a través del filtro FilterIIRLPBessel” 46](#_Toc295295666)

[Tabla 10 Tareas de programación pertenecientes a la Historia de Usuario 9 “Diseñar el filtro FilterIIRHPButterworth” 47](#_Toc295295667)

[Tabla 11 Tareas de programación pertenecientes a la Historia de Usuario 10 “Filtrar señal a través del filtro FilterIIRHPButterworth” 47](#_Toc295295668)

[Tabla 12 Tareas de programación pertenecientes a la Historia de Usuario 11 “Diseñar el filtro FilterIIRHPChebyshev” 47](#_Toc295295669)

[Tabla 13 Tareas de programación pertenecientes a la Historia de Usuario 12 “Filtrar señal a través del filtro FilterIIRHPChebyshev” 48](#_Toc295295670)

[Tabla 14 Tareas de programación pertenecientes a la Historia de Usuario 13 “Diseñar el filtro FilterIIRHPBessel” 48](#_Toc295295671)

[Tabla 15 Tareas de programación pertenecientes a la Historia de Usuario 14 “Filtrar señal a través del filtro FilterIIRHPBessel” 48](#_Toc295295672)

[Tabla 16 Tareas de programación pertenecientes a la Historia de Usuario 15 “Diseñar el filtro FilterIIRBPButterworth” 48](#_Toc295295673)

[Tabla 17 Tareas de programación pertenecientes a la Historia de Usuario 16 “Filtrar señal a través del filtro FilterIIRBPButterworth” 49](#_Toc295295674)

[Tabla 18 Tareas de programación pertenecientes a la Historia de Usuario 17 “Diseñar el filtro FilterIIRBPChebyshev” 49](#_Toc295295675)

[Tabla 19 Tareas de programación pertenecientes a la Historia de Usuario 18 “Filtrar señal a través del filtro FilterIIRBPChebyshev” 49](#_Toc295295676)

[Tabla 20 Tareas de programación pertenecientes a la Historia de Usuario 19 “Diseñar el filtro FilterIIRBPBessel” 49](#_Toc295295677)

[Tabla 21 Tareas de programación pertenecientes a la Historia de Usuario 20 “Filtrar señal a través del filtro FilterIIRBPBessel” 50](#_Toc295295678)

[Tabla 22 Tareas de programación pertenecientes a la Historia de Usuario 21 “Diseñar el filtro FilterIIRBSButterworth” 50](#_Toc295295679)

[Tabla 23 Tareas de programación pertenecientes a la Historia de Usuario 22 “Filtrar señal a través del filtro FilterIIRBSButterworth” 50](#_Toc295295680)

[Tabla 24 Tareas de programación pertenecientes a la Historia de Usuario 23 “Diseñar el filtro FilterIIRBSChebyshev” 51](#_Toc295295681)

[Tabla 25 Tareas de programación pertenecientes a la Historia de Usuario 24 “Filtrar señal a través del filtro FilterIIRBSChebyshev” 51](#_Toc295295682)

[Tabla 26 Tareas de programación pertenecientes a la Historia de Usuario 25 “Diseñar el filtro FilterIIRBSBessel” 51](#_Toc295295683)

[Tabla 27 Tareas de programación pertenecientes a la Historia de Usuario 26 “Filtrar señal a través del filtro FilterIIRBSBessel” 51](#_Toc295295684)

[Tabla 28 Casos de Prueba para las Historias de Usuario 53](#_Toc295295685)

[Tabla 29 Plan de Entrega 59](#_Toc295295686)

[Tabla 30 Plan de Iteración I 63](#_Toc295295687)

[Tabla 31 Casos de Prueba Iteración I 64](#_Toc295295688)

[Tabla 32 Plan de Iteración II 66](#_Toc295295689)

[Tabla 33 Casos de Prueba Iteración II 68](#_Toc295295690)

[Tabla 34 Plan de Iteración III 70](#_Toc295295691)

[Tabla 35 Casos de Prueba Iteración III 71](#_Toc295295692)

[Tabla 36 Plan de Iteración IV 73](#_Toc295295693)

[Tabla 37 Casos de Prueba Iteración IV 74](#_Toc295295694)

[Tabla 38 Historia de Usuario 1 85](#_Toc295295695)

[Tabla 39 Historia de Usuario 2 85](#_Toc295295696)

[Tabla 40 Historia de Usuario 3 85](#_Toc295295697)

[Tabla 41 Historia de Usuario 4 86](#_Toc295295698)

[Tabla 42 Caso de Prueba 1 87](#_Toc295295699)

[Tabla 43 Caso de Prueba 2 88](#_Toc295295700)

[Tabla 44 Caso de Prueba 3 88](#_Toc295295701)

**Índice de Figuras**

Figura 1 Procesamiento de señales analógicas 8

Figura 2 Diagrama de bloques de un sistema de procesamiento digital de señales 9

Figura 3 Partes básicas de un conversor analógico-digital (ADC) 11

Figura 4 Conversión (interpolación) D/A ideal 14

Figura 6 Funcionamiento interno básico detallado de un filtro 18

Figura 5 Funcionamiento interno básico de un filtro 18

Figura 7 a-) Filtro pasa Bajo(LP) b-) Filtro pasa Alto(HP) 19

Figura 8 a-) Filtro pasa Banda (BP) b-) Filtro banda atenuada (BS) 20

Figura 9 Vista del Diseño Preliminar de la Iteración I 64

Figura 10 Vista del Diseño Final de la Iteración I 65

Figura 11 Vista de Diseño Preliminar de la Iteración II 68

Figura 12 Vista de Diseño Preliminar de la Iteración III 71

Figura 13 Vista de Diseño Preliminar de la Iteración IV 74

Figura 14 Vista 1 del uso de un filtro pasa bajo Butterworth 89

Figura 15 Vista 2 del uso de un filtro pasa bajo Butterworth 90

Figura 16 Vista 3 del uso de un filtro pasa bajo Butterworth 91

Figura 17 Vista 1 del uso de un filtro pasa alto Chebyshev 92

Figura 18 Vista 2 del uso de un filtro pasa alto Chebyshev 93

Figura 19 Vista 2 del uso de un filtro pasa alto Chebyshev 94

Figura 20 Vista 1 del uso de un filtro pasa banda Bessel 95

Figura 21 Vista 2 del uso de un filtro pasa banda Bessel 96

Figura 22 Vista 3 del uso de un filtro pasa banda Bessel 97

Figura 23 Vista 4 del uso de un filtro pasa banda Bessel 98

Figura 24 Vista 1 del uso de un filtro supresor de banda Butterworth 99

Figura 25 Vista 2 del uso de un filtro supresor de banda Butterworth 100

Figura 26 Vista 3 del uso de un filtro supresor de banda Butterworth 101

Figura 27 Vista 4 del uso de un filtro supresor de banda Butterworth 102

# Introducción

El modelo de software libre se plantea originalmente como un modelo más justo y equitativo de producción y uso de software, independientemente del ámbito en el que esas tareas se lleven a cabo.

Los valores que sustentan este modelo parten de una concepción más ética del software y su ciclo de vida, y llevan asociadas ideas que fomentan la libertad individual, el progreso científico y tecnológico, la solidaridad y la cooperación.

Por estas razones se presenta como una alternativa de gran interés para los países en desarrollo, pues permite un acceso más directo y fluido a las herramientas informáticas, y una mayor difusión de las mismas y de los conocimientos asociados a ellas.

Un software libre para una determinada tarea permitirá a aquellos potenciales usuarios tener un acceso adecuado a sus funcionalidades, y encontrar así, de forma óptima, una herramienta que aplicar en su trabajo, a diferencia de las limitaciones que establece uno de iguales prestaciones pero distribuido bajo el modelo clásico del software privativo.

Sin embargo, más allá de eso, el software libre puede ser modificado y permite que se ofrezcan servicios de diversa índole basados en él, abriendo así vías para la creación de actividades y negocios sostenibles que pueden tener un impacto positivo en el desarrollo de la economía del país [1].

En el año 2006 el Centro Nacional para la producción de Animales de Laboratorio (CENPALAB), el Centro de Estudios de Ingeniería y Sistemas (CEIS), y una empresa cubano-brasileña TechAgro, se unen para desarrollar el “Sistema para la Gestión de Maquinarias Agrícolas” (SGMA) y un conjunto de componentes para usar en la agricultura de precisión.

El Sistema (SGMA) se desarrolló sobre Borland Developer Studio 2006, en su realización, se utilizaron componentes propietarios desarrollados para el ambiente Delphi, sin embargo para la migración a software libre de dicho sistema, es necesario el uso de componentes con funcionalidades similares.

Se decide utilizar para el desarrollo de los componentes en código abierto, el entorno de desarrollo Lazarus/FPC, por ser altamente compatible con Delphi, y por las numerosas ventajas que brinda en lo económico, político y tecnológico.

Después de una profunda investigación a cerca de herramientas desarrolladas sobre Lazarus/FPC, se tiene como **situación problemática** el desconocimiento de la existencia enla comunidad de desarrolladores, que trabajan con Lazarus/FPC, de componentes que permitan el filtrado digital de señales.

Como consecuencia de esto el **problema a resolver** esla inexistencia de componentes que permitan el filtrado digital de señales en el ambiente de desarrollo Lazarus/FPC. De ahí parte la **actualidad y necesidad de este trabajo** **de diploma** puesto que los mismos no han sido desarrollados y son necesarios para migrar el SGMA hacia software libre.

El **Objeto de estudio** del presente trabajo está enmarcado en el área del procesamiento digital de señales y la creación de componentes en el Entorno de Desarrollo Integrado Lazarus/Free Pascal.

El **Campo de acción** se enmarca en el uso de filtros digitales para el Procesamiento Digital de Señales.

El **aporte práctico** que se espera del presente trabajo radica en la utilización de los componentes implementados, para facilitar la migración del sistema SGMA a software libre y posibilitar el filtrado digital de señales a sistemas implementados con otros fines, consiguiendo con esto la disminución del tiempo de desarrollo de aplicaciones en el entorno Lazarus / Free Pascal.

Como **hipótesis** del trabajo de diploma se tiene la siguiente:

Si se crean los componentes para el filtrado digital de señales usando el IDE Lazarus/Free Pascal, se facilitará la migración del SGMA a software libre y se contará con dichos componentes para ser reutilizados en otras aplicaciones.

El **objetivo general** de este trabajo es:

* Crear componentes para el filtrado digital de señales en lazarus/ FPC.

Para dar cumplimiento al objetivo general surgen los siguientes **objetivos específicos**:

* Determinar las características y funcionalidades de cada componente para el filtrado digital señales.
* Realizar aplicaciones ejemplos que demuestren el uso de los componentes implementados.

También, se plantean las siguientes **tareas** que dan cumplimiento a los Objetivos Específicos antes mencionados

* Realizar un estudio de características y funcionalidades de componentes que se utilizan para el filtrado digital de señales.
* Realizar un estudio de las técnicas que se utilizan para el filtrado digital de señales.
* Realizar las fases de exploración, planificación e iteraciones del ciclo de vida de la metodología XP en el desarrollo de los componentes sobre Lazarus/FreePascal.
* Implementar los componentes para el filtrado digital de señales sobre Lazarus/FreePascal.
* Poner a punto y prueba los componentes desarrollados, mediante ejemplos que muestren las funcionalidades de los mismos.
* Crear la documentación pertinente a cada componente.

El trabajo de diploma está constituido por 3 capítulos.

En el **Capítulo 1** se expone todo lo relacionado con el objeto de estudio de la solución propuesta. Además incluye un estudio del estado del arte de las tecnologías y herramientas para el desarrollo de los objetivos propuestos en este Trabajo de Diploma.

En el **Capítulo 2** se abordan temas relacionados con las dos primeras fases del ciclo de vida de la metodología ágil XP. Se describe que son las Historias de Usuario, las Tareas de Programación, se definen los Casos de Prueba y se realiza la planificación del producto de Software.

El **Capítulo 3** se dedica a la tercera fase del ciclo de vida de la metodología XP. Se aborda todo lo referente a las Iteraciones que se definen en el producto de software. Además se realiza un pequeño análisis de las 3 últimas fases del ciclo de vida de XP, Producción, Mantenimiento y Muerte, aplicadas en el desarrollo de los componentes.

# Capítulo 1 Fundamentos Teóricos

## 1.1 Introducción

En este capítulo se abordan temas de carácter teórico relacionados con el tema de investigación. Por otro lado se realiza un estudio de las tecnologías actuales comparando el entorno de desarrollo escogido con otros existentes. Se exponen las necesidades de crear componentes con funcionalidades destinadas a facilitar el filtrado digital de señales usando el Entorno de Desarrollo Integrado Lazarus/Free Pascal, profundizando en las bondades y ventajas que brinda el mismo, para facilitar migrar el Sistema SGMA a software libre.

Se define la herramienta case y la metodología a utilizar para la modelación de los componentes, así como conceptos importantes relacionados con el procesamiento digital de señales y filtrado de señales digitales.

## 1.2 Procesamiento de Señales

El Procesamiento de Señaleses un área de la Ingeniería Electrónica que se concentra en la representación, transformación y manipulación de señales, y de la información que ellas contienen.

El primer tipo de procesamiento electrónico que se desarrolló y se aplicó extensivamente fue el procesamiento analógico, el cual se lleva a cabo mediante circuitos compuestos por resistores, capacitores, inductores, amplificadores operacionales [2].

### 1.2.1 Sistemas Analógicos y Digitales

Los circuitos electrónicos se pueden dividir en dos amplias categorías: digitales y analógicos. La electrónica digital utiliza magnitudes con valores discretos, mientras que la electrónica analógica emplea magnitudes con valores continuos [3].

**Sistema Analógico**

Una señal analógica es un voltaje o corriente que varía suave y continuamente. Una onda sinusoidal es una señal analógica de una sola frecuencia. Los voltajes de la voz y del video son señales analógicas que varían de acuerdo con el sonido o variaciones de la luz que corresponden a la información que se está transmitiendo. Se dice que un sistema es analógico cuando las magnitudes de la señal se representan mediante variables continuas.

Un sistema analógico contiene dispositivos que manipulan cantidades físicas representadas en forma analógica. En un sistema de este tipo, las cantidades varían sobre un intervalo continuo de valores [3].

**Sistema Digital**

Las señales digitales, en contraste con las señales analógicas, no varían en forma continua, sino que cambian en pasos o en incrementos discretos. La mayoría de las señales digitales utilizan códigos binarios o de dos estados.

Un sistema digital es cualquier dispositivo destinado a la generación, transmisión, procesamiento o almacenamiento de señales digitales. También un sistema digital es una combinación de dispositivos diseñado para manipular cantidades físicas o información que estén representadas en forma digital; es decir, que solo puedan tomar valores discretos [3].

### 1.2.2 Procesamiento Digital de Señales

El procesamiento digital de señales (Digital Signal Processingo DSP por sus siglas en inglés) es un área de la ciencia y la ingeniería que se ha desarrollado rápidamente durante los últimos 30 años. Este rápido desarrollo es el resultado de los avances tecnológicos tanto en los ordenadores digitales como en la fabricación de circuitos integrados. Los ordenadores digitales y el hardware asociado hace tres décadas eran relativamente grandes y caros y, como consecuencia, su uso se limitaba a aplicaciones de propósito general en tiempo no real, tanto científicas como comerciales. El rápido desarrollo de la tecnología de circuitos integrados, empezando con la integración a media escala (MSI, medium-scale integration) y continuando la integración a gran escala (LSI, large-scale integration), y ahora, la integración a muy gran escala (VLSI, very-large-scale integration) de circuitos electrónicos integrados ha estimulado el desarrollo de ordenadores digitales más potentes , pequeños, rápidos y baratos y de hardware digital de propósito general [4].

Dado que el procesamiento digital de una señal solo requiere efectuar ciertos cálculos a partir de los datos disponibles, y que en muchas ocasiones dichos cálculos pueden ser efectuados en forma manual, se puede afirmar que el procesamiento digital de señales se practicó durante varios siglos, mucho antes de la aparición de los computadores, en situaciones tales como el análisis y la predicción del movimiento de cuerpos celestes, o en el análisis y la predicción de las mareas [2].

Una **señal** se define como una cantidad física que varía con el tiempo, el espacio o cualquier otra variable o variables independientes. Matemáticamente, se describe una señal, como una función de una o más variables independientes.

Un **sistema** se puede definir como un dispositivo físico que realiza una operación sobre una señal. Por ejemplo un filtro que se usa para reducir el ruido y las interferencias que corrompen la señal conteniendo la información deseada, se denomina sistema. Cuando se pasa una señal a través de un sistema, como en el caso del filtrado, se dice que se ha procesado la señal. En general, el sistema se caracteriza según el tipo de operación que realiza sobre la señal.

Es conveniente ampliar la definición de sistema para incluir no solo dispositivos físicos, sino también realizaciones software de operaciones sobre una señal. En el procesamiento digital de señales en un ordenador, las operaciones realizadas sobre una señal constan de varias operaciones matemáticas especificadas por un programa software. En este caso el programa representa una implementación del sistema en software [4].

**Procesamiento de Señales en Tiempo Discreto** (Discrete-Time Signal Processing por sus siglas en inglés) se refiere al procesamiento de señales discretas en el tiempo o en el espacio. Esto implica que solo se conoce el valor de la señal en instantes o en puntos específicos. Sin embargo, la amplitud de la señal es continua, es decir, puede tomar infinitos valores diferentes [2].

**Procesamiento Digital de Señales** añade a la característica anterior, manejar la amplitud en forma discreta, la cual es una condición necesaria para que la señal pueda ser procesada en un computador digital. La amplitud de la señal solo puede tener un número finito de valores diferentes [2].

**Elementos de un sistema de procesamiento digital de señales**

La mayor parte de las señales que aparecen en los ámbitos de la ciencia y la ingeniería son de naturaleza analógica, es decir, las señales son funciones de una variable continua, como el tiempo o el espacio y normalmente toman valores en un rango continuo. Tales señales pueden ser procesadas directamente por sistemas analógicos adecuados (como filtros o analizadores de frecuencia) o multiplicadores de frecuencia con el propósito de cambiar sus características o extraer cualquier información deseada. En tal caso, se dice que la señal ha sido procesada directamente en forma analógica, como se ilustra en la figura 1. Tanto la señal de entrada como la de salida están en forma analógica [4].

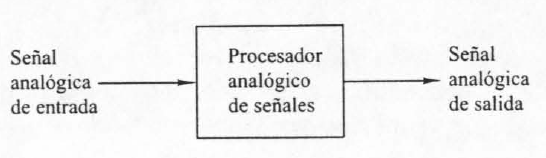


Figura Procesamiento de señales analógicas

El procesamiento digital de señales proporciona un método alternativo para procesar una señal analógica, como se ilustra en la Figura 2. Para realizar el procesamiento digital, se necesita una interfaz entre la señal analógica y el procesador digital. Esta interfaz se denomina conversor analógico digital (A/D).

En aplicaciones donde la salida del procesador digital de señales se requiere en forma analógica, como en las comunicaciones digitales, debe proporcionarse otra interfaz desde el dominio digital al analógico. Tal interfaz se denomina conversor digital analógico (D/A). De este modo, la señal se entrega al usuario en forma analógica, como se ilustra en el diagrama en bloques de la Figura 2 [4].

**Conversores Analógico-Digital y Conversores Digital-Analógico**

Figura Diagrama de bloques de un sistema de procesamiento digital de señales

La mayoría de los transductores ofrecen a su salida señales eléctricas del tipo analógico. Sin embargo, el procesamiento de dichas señales es, generalmente, llevado a cabo por equipos digitales: micro procesadores, micro controladores, u ordenadores. Por ello se hace necesaria una conversión del tipo señal analógica a señal digital, con lo que la información suministrada por el transductor podrá ser tratada digitalmente, con las ventajas que ello ofrece, económicas, como de potencial de procesamiento.

El resultado del procesamiento digital es un conjunto de palabras en binario, que deben ser aplicadas a los elementos actuadores correspondientes y que, por lo general, suelen responder a señales analógicas. De nuevo se tiene la necesidad de llevar a cabo una conversión; en este caso será contraria a la anterior: señal digital a señal analógica. Estos tipos de conversión se utilizan ampliamente en los sistemas de control electrónico: telemetría, adquisición de datos, control de procesos [5].

**Conversor Analógico Digital**

La misión de un conversor analógico digital (analogic to digital converter o ADC por sus siglas en inglés) es obtener una representación digital (conjunto de unos y ceros) de una magnitud analógica, véase Figura 3. Las operaciones necesarias para convertir una señal analógica en un conjunto de códigos digitales, son las siguientes [5]:

* **Muestreo:** Mediante esta operación se obtienen los valores instantáneosde la señal analógica. La frecuencia de muestreo debe ser de un valor tal, que en el proceso inverso, se asegure la completa reconstrucción de la señal original. Esta frecuencia de muestreo (fm) viene impuesta por el Teorema de Nyquist, que obliga a que sea, por lo menos, el doble de la máxima frecuencia (fmax) presente en la señal que va a ser digitalizada.

fm ≥ 2 fmax

* **Retención:** Esta operación es necesaria para que el valor instantáneo de la muestra se mantenga durante el tiempo empleado por el ADC para la conversión.
* **Cuantificación:** Como los valores de las muestras obtenidas mediante el muestreo de la señal analógica pueden ser infinitos, y el número de bits de salida del ADC es finito, es necesario realizar una correspondencia entre tramos de valores de la señal analógica y estadosdigitales posibles con “n” bits. Esta correspondencia se le conoce con el nombre de cuantificación.
* **Codificación:** Es el proceso en el cual se asigna un conjunto de bits (código digital) a cada uno de los N niveles de cuantificación. Si a la entrada del ADC aparecen valores solo positivos o solo negativos, se utilizarán para la salida digital los códigos unipolares(binario natural y BCD); si la entrada al ADC es una señal analógica con valores positivos y negativos se utilizan los códigos bipolares. En un ADC, el valor analógico equivalente al bit de menor peso (LSB) es:

1 LSB = q = FS / N

Siendo N el número de combinaciones posibles del código digital de salida. Si, por ejemplo, este fuese el binario natural de 4 bits, se tiene:

1 LSB = q = FS / 24 = FS /16

Siendo FS (Full Scale) el valor a fondo de escala de la señal analógica de entrada al ADC.

La figura 3 muestras las partes básicas de un conversor analógico digital [5].

[](http://es.wikipedia.org/wiki/Archivo:Conversor_AD.svg)

Bajo ciertas condiciones, una señal puede representarse y reconstruirse por completo partiendo del conocimiento de sus valores, o muestras, en puntos igualmente espaciados en el tiempo. Esta propiedad un tanto sorprendente se deriva de un resultado básico que se conoce como el teorema de muestreo [6].

Figura Partes básicas de un conversor analógico-digital (ADC)

**Teorema del Muestreo de Nyquist-Shannon**

Este teorema fue formulado en forma de conjetura por primera vez por Harry Nyquist en 1928, y fue demostrado formalmente por Claude E. Shannon en 1949.

Dada una señal analógica cualesquiera, ¿Cómo se debe elegir el periodo de muestreo ó, lo que lo mismo, la velocidad de muestreo ? Para contestar a esta cuestión es necesaria cierta información sobre la característica de la señal que va a ser muestreada [4].

En particular, se debe tener cierta información general sobre el contenido de la frecuencia de la señal. Generalmente, dicha información se encuentra disponible. Por ejemplo se sabe que la mayor frecuencia en señales de voz ronda los 3.000Hz. Por otra parte, las señales de televisión tienen componentes de frecuencia importante hasta los 5MHz. La información contenida en dichas señales se encuentra en las amplitudes, frecuencias y fases de las distintas componentes de frecuencia, pero antes de obtener dichas señales no se conoce sus características con detalle. De hecho, el propósito del procesamiento de señales es normalmente la extracción de dichas características. Sin embargo, si se conoce la máxima frecuencia de una determinada clase de señales (por ejemplo, señales de voz, de video, etc.), se puede especificar la velocidad de muestreo necesaria para convertir las señales analógicas en señales digitales [4].

Suponiendo que cualquier señal analógica se puede representar como una suma de sinusoides de diferentes amplitudes, frecuencias y fases, es decir:

(1.1)

Donde *N* indica el número de componentes de frecuencia. Todas las señales, como las de voz ó video se prestan a dicha representación en cualquier intervalo de tiempo pequeño. Normalmente, las amplitudes, fases y frecuencias varían lentamente de un intervalo de tiempo al siguiente. Suponiendo que las frecuencias de una determinada señal no exceden una frecuencia máxima conocida . Por ejemplo, = 3KHz, para señales de voz y = 5MHz, para señales de video. Dado que la máxima frecuencia puede variar ligeramente (por ejemplo, de un orador a otro) se puede querer asegurar que no sobrepase determinado valor y para ello se pasa la señal analógica por un filtro que atenúe fuertemente las componentes de frecuencia por encima de . Así, se tiene cierto grado de seguridad de que ninguna señal de la clase de interés tiene componentes de frecuencia (con amplitud o potencia significativa) por encima de .  En la práctica, este filtrado se realiza antes del muestreo. El conocimiento de permite seleccionar la velocidad de muestreo apropiada. Se sabe que la frecuencia más alta de la señal analógica que puede reconstruirse sin ambigüedad cuando la señal se muestrea a una velocidad de = 1/*T* es /2. Cualquier frecuencia por encima de Fs/2 o por debajo de – /2 produce muestras que son idénticas a las correspondientes a las frecuencias dentro del intervalo – /2 ≤ – ≤ /2. Para evitar las ambigüedades, que resultan del aliasing, se debe seleccionar una velocidad de muestreo lo suficientemente alta, es decir, escoger /2 mayor que . Por lo tanto para evitar el problema de aliasing, se selecciona según

(1.2)

Donde Fmax es la frecuencia más alta de la señal analógica. Con la velocidad de muestreo seleccionada de esta manera se tiene que cualquier componente de frecuencia, por ejemplo , de la señal analógica se corresponde en tiempo discreto con una sinusoide de frecuencia

(1.3)

O, equivalente,

(1.4)

Dado que es la frecuencia más alta (única) en una señal en tiempo discreto, la elección de la velocidad de muestreo según   
*𝐹𝑠>2𝐹𝑚𝑎𝑥* evita el problema del aliasing. En otras palabras, la condición garantiza que todas las componentes sinusoidales de la señal analógicas se correspondan con componentes en frecuencia de tiempo discreto en el intervalo fundamental. Por lo tanto, todas las componentes en frecuencias de la señal analógica están representadas sin ambigüedad en la forma muestreada de la señal, y así la señal analógica puede ser reconstruida sin distorsión a partir de la muestras usando un método de interpolación apropiado. La fórmula de interpolación ideal o apropiada se especifica mediante el teorema del muestreo [4].

**Teorema. “**Teorema del muestreo.Si la frecuencia más alta contenida en una señal analógica es y la señal se muestrea a una velocidad , entonces se puede recuperar totalmente a partir de sus muestras mediante la siguiente función de interpolación: **”** [4]

(1.5)

Así, se puede expresar como

(1.6)

donde son las muestras de [4].

Cuando el muestreo de se realiza a la tasa mínima de muestreo , la fórmula de reconstrucción se transforma en: [4]

(1.7)

La tasa de muestreo dada por , se denomina tasa de Nyquist. La figura 4 ilustra el proceso de Conversión Digital Analógica ideal que utiliza la función de interpolación [4].



Figura Conversión (interpolación) D/A ideal

**Comparación entre DSP y ASP, ventajas y limitaciones**

El Procesamiento Analógico de Señales (ASP por sus siglas en inglés) es generalmente más simple que el procesamiento digital, el cual requiere típicamente de un filtro análogo antialiasing, un conversor A/D, un procesador DSP, un conversor D/A y un filtro análogo para suavizar la salida. Sin embargo, el procesamiento analógico es incapaz de realizar muchas funciones que el procesamiento digital sí puede realizar [2].

Algunas ventajas del procesamiento digital con respecto al analógico son [2]:

* El envejecimiento de los componentes no afecta el resultado del proceso.
* Todos los dispositivos fabricados se comportan en forma idéntica, ya que la tolerancia de los componentes no influye en el procesamiento.
* Se puede reconfigurar un dispositivo modificando los valores de algunos coeficientes; no es necesario reemplazar componentes.
* El procesamiento analógico de señales de muy baja frecuencia se dificulta debido al requerimiento de capacitores de gran capacidad y muy baja corriente de fuga. En el caso del procesamiento digital no existen limitaciones; se pueden procesar señales con períodos de horas (tales como las mareas) e incluso de años (tales como las manchas solares).
* El procesamiento digital es capaz de realizar tareas muy complejas.

Algunas desventajas del procesamiento digital con respecto al analógico son [2]:

* Mayor limitación en frecuencias altas, ya que normalmente se requieren conversores A/D capaces de tomar muestras a una tasa varias veces mayor que la de la frecuencia de la señal analógica, y procesadores capaces de efectuar muchas operaciones por cada muestra recibida. Por ejemplo, un filtro digital FIR con una respuesta al impulso de 1000 coeficientes, operando a una tasa de 10·106 muestras por segundo, requeriría un procesador capaz de efectuar al menos 10·109 multiplicaciones y sumas por segundo.
* El diseño es generalmente más complejo, ya que incluye aspectos de hardwarey de software.
* El rango dinámico en la amplitud (razón entre la señal más grande y la más pequeña que pueden ser procesadas) es más limitado. Sin embargo, la disponibilidad actual de conversores A/D de alta resolución (18 a 24 bits) y de procesadores capaces de efectuar cálculos en punto flotante con un gran número de decimales, puede eliminar esta desventaja en muchos casos.
* El rango dinámico en la frecuencia también es más limitado. Por ejemplo, un filtro analógico sencillo podría procesar simultáneamente componentes de 1Hz y de 1MHz, a pesar de que las 2 frecuencias están separadas por 6 órdenes de magnitud. Para realizar la misma tarea con un filtro digital, sería necesario procesar un gran número de muestras.

## 1.3 Filtros

En su definición más general, un filtro se puede definir como todo procesamiento que altera la naturaleza de una señal sonora de una forma u otra. Un filtro es un proceso computacional o algoritmo mediante el cual una señal digital o secuencia de muestras, es transformada en una segunda secuencia de muestra o señal digital de salida [7].

Los filtros se utilizan mucho en todos los ámbitos del procesamiento de señales, y son una componente esencial en toda cadena de comunicación. Constituyen la base del procesamiento de señales, que puede aplicarse a señales de todo tipo (sonidos, imágenes, vídeo, vibraciones sísmicas, etc.) [7].

Existen dos tipos de filtros, analógicosy digitales. Ambos son muy distintos en su construcción y en la forma en que tratan la señal.

Los filtros digitales son una importante parte de un DSP. De hecho, su gran versatilidad es una de las razones claves por lo que los DSP son tan populares.

Muchos problemas pueden ser tratados por un filtro, tanto digital como analógico. ¿Por cuál decantarse? Los filtros analógicos son más baratos, rápidos y tienen un gran rango dinámico tanto en amplitud como en frecuencia. En cambio, los filtros digitales son enormemente superiores a los analógicos en cuanto a el nivel de cumplimiento [8].

**Filtros Analógicos**

Los filtros analógicos se utilizan por ejemplo en circuitos electrónicos que hacen uso de resistencias, inductores, condensadores y amplificadores para producir el efecto requerido en el filtrado.

Existen unas técnicas estándares bien establecidas para el diseño de un circuito de filtrado analógico. En todas las etapas, la señal que se filtra es un voltaje eléctrico o corriente como puede ser la analógica directa de una magnitud física (un sonido, una señal de video) compleja, etc. [8].

**Filtros Digitales**

Aprovechan los avances de la tecnología digital para emular sistemas analógicos. Deben cumplir los requisitos necesarios para procesar las señales analógicas (véase Teorema del muestreo de Nyquist-Shannon).

Un filtro digital requiere un procesador digital para realizar cálculos numéricos en los valores muestreados de la señal. El procesador puede ser un ordenador corriente, como una PC, un chip DSP ó un DSP especializado [8].

La parte analógica de la señal debe ser previamente muestreada y digitalizada por un convertidor A/D. Los números binarios resultantes de la conversión anterior, que representan valores sucesivos muestreados de la señal de entrada, son transferidos al procesador, que realiza unos cálculos numéricos sobre ellos. Estos cálculos suelen ser multiplicaciones de los valores de entrada por unas constantes y suma de las anteriores multiplicaciones. Si es necesario, los resultados de los cálculos, que representan los valores de una señal filtrada, son sacados a través de un D/A para convertir la señal a su forma analógica [8].

La figura 5 muestra la configuración básica de un filtrado digital: [8]

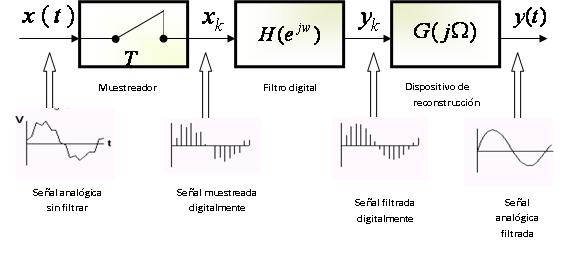


Figura Funcionamiento interno básico de un filtro

Que con más detalle queda de la forma que muestra la figura 6:

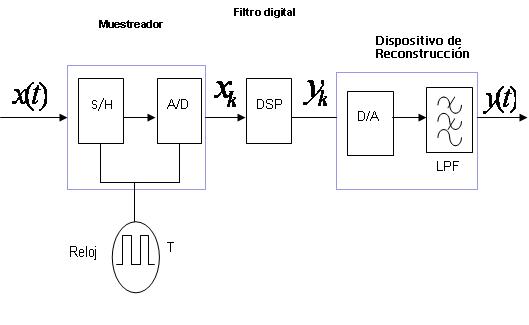


Figura Funcionamiento interno básico detallado de un filtro

**Ventajas de los filtros digitales**

Algunas ventajas de los filtros digitales son [8]:

* Un filtro digital es programable, su función está determinada por un programa almacenado en el procesador. Esto significa que el efecto del filtro puede ser cambiado fácilmente sin modificar su hardware. Un filtro analógico solo puede cambiar rediseñando el circuito de filtrado.
* Los filtros digitales son fácilmente diseñados, testados e implementados en un ordenador convencional o en una estación de trabajo.
* Las características de los circuitos de filtrado analógico (particularmente aquellos que contengan componentes activos) son susceptibles a las variaciones de velocidad y de temperatura. En cambio, los filtros digitales no sufren este problema, y son extremadamente establescon respecto al tiempo y la temperatura.
* A parte de sus correspondientes partes analógicas, los filtros digitales solo pueden tratar señales de bajafrecuenciacon gran exactitud. A medida que la velocidad de la tecnología DSP aumente, los filtros digitales pueden aplicarse en señales de altafrecuencia, por ejemploen el dominio de las frecuencias de radio, el cual fue un campo exclusivo reservado a la tecnología analógica.
* Los filtros digitales son mucho más versátiles en su capacidad de procesar señales de diferentes formas. Esto significa que algunos filtros digitales tienen la capacidad de adaptarse a los cambios en las características de la señal.
* Los procesadores DSP más rápidos pueden tratar combinaciones complejas de filtros en paralelo o en cascada, haciendo que los requerimientos de hardware sean relativamente simples y compactas en comparación con la circuitería analógica.
* Alta inmunidad al ruido.
* Alta precisión (limitada por los errores de redondeo en la aritmética empleada).
* Muy bajo coste.

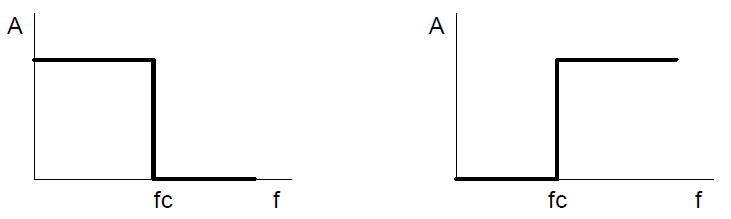
### 1.3.1 Algunas clasificaciones de los Filtros digitales

Existen varios tipos de filtros así como distintas clasificaciones para estos.

De acuerdo con la parte del espectro que dejan pasar y que atenúan se clasifican en [7]:

**Filtros pasa alto (High Pass, HP):** dejan pasar las frecuencias que están por encima de una determinada frecuencia (véase Figura 7).

**Filtros pasa bajo (Low Pass, LP):** dejan pasar las frecuencias que están por debajo de una determinada frecuencia (véase Figura 7).



a-) b-)

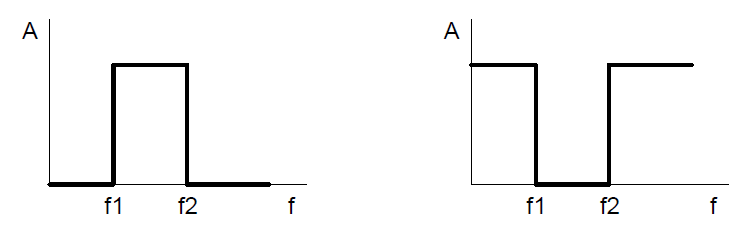
Figura a-) Filtro pasa Bajo (LP) b-) Filtro pasa Alto (HP)

Estos dos tipos de filtros están definidos por su frecuencia de corte, que es la frecuencia a la cual la amplitud de la señal se reduce a 0.707 o lo que es igual () de su valor máximo, es decir, sufre una atenuación de 3 dB.

**Filtros pasa banda (Band Pass, BP):** dejan pasar las frecuencias que están situadas en una determinada banda de frecuencia, es decir, entre dos determinadas frecuencias (véase Figura 8).

**Filtros rechazo de banda, banda atenuada o supresor de banda (Band Reject, Band stop o Notch, BS):** dejan pasar todas las frecuencias excepto las que están situadas en una determinada banda de frecuencia, es decir, entre dos determinadas frecuencias *f*1 y *f*2. Estas frecuencias son las frecuencias a las que la amplitud de la señal se reduce a 0.707 o lo que es igual () de su valor máximo, es decir, sufre una atenuación de 3 dB.

Estos dos tipos de filtros están definidos por su frecuencia central y su ancho de banda, que sería la diferencia entre las frecuencias de corte inferior y superior (véase Figura 8).



a-) b-)

Figura a-) Filtro pasa Banda (BP) b-) Filtro banda atenuada (BS)

De acuerdo con el tipo de respuesta ante entrada unitaria se clasifican en [7]:

**FIR (Respuesta de Impulso Finita o Finite Impulse Response):** una muestra

de la salida se puede definir como una combinación linear de muestras de la entrada presentes y pasadas.

**IIR (Respuesta de Impulso Infinita o Infinite Impulse Response):** se distinguen de los filtros FIR por la presencia de una recursividad; la señal de salida del filtro se re inyecta a la entrada del mismo, constituyendo un circuito recursivo o con feedback. Este método permite implementar filtros con respuesta más compleja y con menos datos. Como inyectamos constantemente energía en el circuito, la respuesta al impulso tiene una duración potencial infinita.

De acuerdo con su orden [7]:

El número de muestras anteriores a la actual que se utilizan en un filtro para generar una muestra de salida corresponde al orden del filtro. Mientras mayor sea el orden de un filtro, es decir, cuantos más retardos se utilicen en el circuito, el corte del filtro será más abrupto.

**FIR vs IIR**

Los filtros FIR ofrecen en general una respuesta de fase más lineal y no entran en oscilación, es decir, no se vuelven inestables, ya que no poseen realimentación. Por otro lado, requieren un gran número de términos en sus ecuaciones y eso les hace más costosos en cuanto a cálculo o carga computacional. Un filtro FIR con un corte muy abrupto, es decir, que tenga una banda de transición muy corta puede requerir hasta centenas de retardos.

En cuanto a los filtros IIR, son muy eficaces y pueden proporcionar pendientes de corte muy pronunciadas. Por otro lado, al poseer características de realimentación o feedback, tienen tendencia a entrar en oscilación y en resonancia [7].

Los filtros son sistemas que se implementan principalmente para eliminar ciertas componentes no deseadas de una señal. Generalmente estas componentes no deseadas se describen en función de sus contenidos en frecuencias. Un filtro ideal permite el paso de ciertas frecuencias sin modificarlas, y elimina completamente otras [9].

Resulta difícil implementar filtros ideales, debido a esto se han desarrollado aproximaciones matemáticas, como son los filtros de Butterworth, Chebyshev, Bessel, entre otros.

Los filtros Butterworth tienen una atenuación lo más plana posible en la banda de paso, a expensas de un corte poco abrupto, su pendiente de transición es pequeña, mientras que su respuesta transitoria resulta satisfactoria, por otro lado los filtros Chebyshev poseen un corte más abrupto, a expensas de poseer rizado en la banda de paso, su pendiente de transición es más alta, pero su respuesta transitoria resulta menos satisfactoria que la de los filtros Butterworth. Los filtros Bessel proporcionan una mínima deformación de la señal, ya que son de desfase lineal, y su pendiente de transición es más pequeña que la de los filtros Butterworth.

### 1.3.2 Aproximación de Butterworth

La aproximación de Butterworth se denomina también aproximación máximamente plana, ya que la atenuación en la mayor parte de la banda pasante es cero y disminuye gradualmente hasta el final de la banda pasante. Por debajo de la frecuencia de inflexión, la respuesta decae a un ritmo aproximado de 20n dB por década, donde n es el orden del filtro [10].

Pendiente = 20n dB/década

El equivalente en octavas es

Pendiente = 6n dB/octavas

Por ejemplo, un filtro de Butterworth de primer orden decae a un ritmo de 20 dB por década o 6 dB por octava; uno de cuarto orden decae 80 dB por década o 24 dB por octava; uno de noveno orden, 180 dB por década o 54 dB por octava, etc. [10].

### 1.3.3 Aproximación de Chebyshev

En algunas aplicaciones no es importante la existencia de una banda pasante con una respuesta plana. En este caso, la aproximación de Chebyshev puede ser la adecuada, ya que decae más rápidamente en la región de transición que la aproximación de Butterworth. El precio a pagar por esta rápida caída es el rizado que aparecerá en la banda pasante de la respuesta en frecuencia.

El número de rizados en la banda pasante de un filtro paso bajo de Chebyshev es igual a la mitad del orden del filtro: [10]

Num. rizados = n/2

Un filtro de orden 10, tendrá 5 rizados en la banda pasante; si es de orden 15 tendrá 7,5 rizados. Normalmente, un diseñador escogerá una amplitud de rizado entre 0.1 y 3 dB, dependiendo de las necesidades de la aplicación [10].

### 1.3.4 Aproximación de Bessel

La aproximación de Bessel tiene una banda pasante plana y una banda eliminada monótona parecida a la de la aproximación de Butterworth. Para un filtro del mismo orden, sin embargo, la caída en la zona de atenuación es mucho menor.

El filtro de Bessel tiene una banda pasante plana, una zona de atenuación con pendiente relativamente sueva y una banda pasante monótona. Tomando una serie de especificaciones de filtros complejos, la aproximación de Bessel produce siempre una zona de atenuación más ancha de todas las aproximaciones. Dicho de otra forma, tiene el mayor orden o mayor complejidad de todas las aproximaciones [10].

La aproximación de Bessel está optimizada para producir un desfase lineal con la frecuencia. En otras palabras, los filtros de Bessel sacrifican la pendiente en la atenuación por conseguir un desfase lineal. El desfase lineal para todas las frecuencias de la banda pasante significa que la frecuencia fundamental y los armónicos de una señal no sinusoidal en la entrada del filtro se desfasarán linealmente a la salida del mismo. Por ello, la forma de la señal de salida será la misma que la de la señal de entrada, si se aplica una tensión en la entrada del filtro y se observa su salida en un osciloscopio, se comprueba que tiene la mejor respuesta al escalón de todos los filtros [10].

## 1.4 Tendencias y Tecnologías Actuales

Para la creación de componentes que permitan el tratamiento de señales, usados en aplicaciones implementadas en el entorno de desarrollo Lazarus/Free Pascal; es necesario el estudio de las tecnologías que hacen posible su confección. En este acápite se realiza un estudio de dichas tecnologías.

**Pascal/Free Pascal**

El lenguaje de programación Pascal aparece por primera vez en 1971, de la mano de Niklaus Wirth. Wirth concibe este lenguaje como una forma de mejorar el Algol.

Pascal cobra auténtica vida a partir de mediados de los 80, popularizado por Turbo Pascal y Apple Pascal.

Pascal fue popular hasta mediados de los 90, ya que una gran parte de las aplicaciones desarrolladas para Mac y programas de MS-DOS fueron realizadas en Pascal. Borland [12], se encargó de la mayor parte de la evolución de este lenguaje hasta la aparición de Turbo Pascal 7.

Con la llegada a mediados de los 90, de Windows y el renacimiento de los sistemas Unix (entre ellos Linux), origina que Pascal pase a un segundo plano sustituido por C [11].

En ese momento, prácticamente solo Delphi (Object Pascal para Windows) consigue mantenerse.

Casi todo el mundo intenta pasarse a C++, pero este no consigue calar del todo por su extrema complejidad. Su sistema de objetos, las macros, los crípticos nombres de funciones, los namespaces tan difíciles de manejar, los templates, y otras características, hizo que rápidamente muchos desarrolladores pasen de querer usar esa herramienta tan potente, a buscar alternativas que fueran más simples y prácticas.

En el momento en que Borland pasa de Turbo Pascal a Delphi, aparece Free Pascal. Free Pascal es un intento de la comunidad de código libre de hacer un compilador compatible con Turbo Pascal para las plataformas más famosas (incluido Linux, OS X y Windows). Desde la versión 2.0, Free Pascal es además compatible con Delphi [11].

Free Pascal es, hoy en día, el sistema preferido para programar en Pascal multiplataforma. Su IDE principal, Lazarus, es un clon del Delphi, corre tanto en Windows, OS X, como en Linux.

Lazarus y Free Pascal son la gran esperanza del lenguaje de programación Pascal, y de los entornos de desarrollo multiplataforma en general. Su desarrollo es muy activo, y tiene un futuro prometedor [11].

### 1.4.1 Comparativa de Free Pascal frente a otros Lenguajes

Free Pascal presenta numerosas ventajas y desventajas en comparación con otros lenguajes. A continuación se muestran algunas ventajas y desventajas [13].

**Ventajas respecto a C**

* Limpieza, Facilidad y Rapidez de compilación (FreePascal es 9 veces más rápido que GCC compilando).
* Mejor gestión de librerías y sus dependencias.
* Un entorno de programación gráfica (Lazarus y Delphi) sencillo y efectivo.
* Mejor mantenimiento del código.
* Mejor gestión de los ficheros de código (no es necesario usar make incluso para proyectos grandes).
* Mejor portabilidad.

**Desventajas respecto a C**

* Código generado ligeramente más lento, en torno al 10%.
* Acceso más directo a las librerías del sistema. En FreePascal es fácil usar librerías del sistema, pero hay que declarar las funciones previamente.
* Aritmética de punteros (no es lo mismo que uso de punteros). Algo que todo programador debería evitar. Pascal tiene punteros, y es posible la aritmética en los mismos con algunos trucos, pero no se recomienda.

**Ventajas respecto a C++.**

Además de todo lo que se gana con C:

* Sistema de objetos mucho más claro.
* Propiedades.

**Desventajas respecto a C++**

* Plantillas. No son necesarias, incluso para aplicaciones complejas y son difíciles de entender, además de tener una sintaxis poco elegante.
* Herencia múltiple. Sin embargo, es posible usar interfaces, al estilo Java.

**Ventajas respecto a Java.**

* Mayor velocidad, legibilidad y ahorro en memoria.
* Mayor flexibilidad, al ser posible acceder a funciones del sistema directamente.
* Ejecutables nativos.
* Código compilado cuando el programa arranca. En Java, tiene que actuar solo en tiempo de compilación (por sus siglas en inglés JIT ¨Just in Time compiler¨), lo cual da sensación de lentitud al usuario.

**Desventajas respecto a Java**

* Recolector de basura: Java gestiona la memoria por el usuario (aun a cambio de ser muchísimo menos eficiente)
* Se evita el uso de punteros. Aunque en Pascal es posible usar objetos y clases sin preocuparse, siendo transparente su uso.
* Un mismo ejecutable (archivo jar) funciona en todas las plataformas. Esto no es un gran inconveniente, pues FreePascal y Lazarus se portan con mucha facilidad (generalmente es solo una re compilación).

**Ventajas respecto a Python, Ruby, Perl**

* Mucha mayor velocidad de ejecución.
* Mayor seguridad en aplicaciones complejas, debido a su estilo estático.
* Acceso mucho más fácil a librerías externas.

**Desventajas respecto a los lenguajes scripting**

* Recolector de basura.
* Velocidad de escritura del código, aunque hay que tener en cuenta que esto a veces tiene como consecuencia baja legibilidad (sobre todo en el caso de Perl).

**Ventajas respecto a Lisp**

Lisp es un lenguaje de programación muy antiguo y muy complejo. Lo único que tienen en común es que ambos son lenguajes de propósito general.

Sin embargo, ello también demuestra que ambos lenguajes son tan diferentes que no deberían ser comparados tan a la ligera. Así, siguiendo con la analogía, Pascal es un lenguaje fuerte y seguro que está pensado para hacer aplicaciones que duran milenios (como las pirámides), mientras que Lisp es un lenguaje dinámico y flexible pero más vulnerable para hacer aplicaciones que tengan que estar constantemente actualizándose y cambiando (como los organismos).Por supuesto, es una exageración. Ni Pascal es tan rígido como dice (solo hay que ver cómo evolucionan Lazarus, Delphi, y miles de aplicaciones creadas con ellos), ni Lisp la última panacea como sus fanáticos pretenden convencer al mundo (y de ahí que a la hora de la verdad haya tan pocas aplicaciones disponibles realizadas en Lisp).

Lisp tiene macros, es dinámico, también muy complejo y realmente accesible solamente para mentes brillantes, o, al menos, para mentes con mucho tiempo y paciencia para aprender.

**Otras ventajas:**

* Pascal es mucho más rápido.
* Pascal necesita mucha menos memoria.
* Accede más fácilmente a las librerías del sistema.
* Existen editores específicos (Lazarus y Delphi) con constructor de Interfaces y otras facilidades, más sencillas en su utilización y más completas que las existentes en Lisp (Slime y LispWorks básicamente).
* Más seguro a la hora de manejar tipos de datos, menos posibilidades de fallos.

**Desventajas respecto a Lisp**

* El REPL o consola de comandos Lisp, donde se puede probar el programa mientras se hace. En FreePascal no es un gran problema pues la velocidad de compilación es altísima.
* Macros, que son básicamente formas de escribir programas, que escriben programas. Sucede algo similar a las plantillas del C++, ni está demostrado que sean necesarias, ni queda claro que sus aparentes beneficios sean mejores que la complejidad y falta de uniformidad que suelen añadir al sistema.

### 1.4.2 Lazarus

Lazarus [14] puede ser el entorno de desarrollo que mejor se ejecuta sobre Linux, Windows y Mac OS X. Utilizado para crear aplicaciones nativas con plataformas de código independiente.

Lazarus es una biblioteca de código abierto de componentes visuales y un IDE fuerte para el desarrollo de plataformas. El IDE contiene todas las características de una suite de desarrollo moderna, incluyendo un depurador, completamiento de código, los diseñadores visuales, refactorización de herramientas, herramientas de traducción y documentación. El proyecto Lazarus comenzó hace diez años sobre Linux y ahora se ejecuta sobre las principales plataformas existentes, Linux, Windows y Mac OS X. El lema del proyecto Lazarus es “Escriba una vez compilen en cualquier lugar” (en inglés “Write once compile anywhere”) [14].

**Algunas de las características de Lazarus:**

* Un lenguaje fácil de aprender: Pascal.
* Un diseñador de formato visual.
* Produce los ejecutables de códigos nativos que ejecutan con velocidades comparables a lenguaje C / C++.
* Admite el acceso directo a bibliotecas del sistema.
* Soporte de código ensamblador.
* Maneja fácilmente proyectos grandes, con millones de líneas.
* Compatible con la biblioteca de componente visual de Delphi [14].

**IDE y Biblioteca de Componentes de Lazarus**

Lazarus da una cara al compilador Free Pascal (sus siglas en inglés FPC), proveyendo la biblioteca de componentes de Lazarus (sus siglas en inglés LCL), una biblioteca de componentes visuales, como botones, campos de edición, diálogos de archivo y mucho más. Estos componentes se pueden ejecutar sobre Linux, MS Windows, Mac OS X, FreeBSD y Solaris al utilizar artefactos nativos. Adicionalmente, sobre Linux, usted tiene la elección entre GTK o Qtcomo soporte final [14].

El IDE de Lazarus utiliza LCL y tiene un diseñador de formato visual integrado, que permite que usted edite las formas gráficamente. El diseñador de Lazarus trabaja directamente con la fuente de unidad de Pascal correspondiente [14].

**Free Pascal Compiler (FPC)**

Lazarus utiliza FPC, que comprende Object Pascal (un descendiente de Pascal). Free Pascal es un compilador profesional de Object Pascal de 32 y 64 bit. Está disponible para los siguientes sistemas operativos: Linux, FreeBSD, Mac OS X/Darwin, DOS, Win32, Win64, WinCE, OS/2, NetWare, MorphOS y para procesadores diferentes: Intel x86, AMD64/x86\_64, PowerPC, PowerPC64, SPARC y ARM.

FPC también compila rápido, normalmente más de 10000 líneas de código por segundo. Eso es porque en Object Pascal las declaraciones delanteras son más limitadas que en C/C++. Esto salva mucho tiempo, incluso para programas pequeños, y permite que usted sea más productivo [15].

### 1.4.3 Programación Basada en Componentes

A pesar del éxito de la programación orientada a objetos, surgen nuevos problemas o desafíos, como por ejemplo, la posibilidad de reutilización, objetivo importante de cualquier modelo de objetos.

En el contexto de la Ingeniería del software la reutilización es una idea tanto antigua como nueva. En la actualidad, los complejos sistemas de alta calidad basados en computadoras se deben construir en un tiempo muy corto y demanda un enfoque más organizado de la reutilización.

La ingeniería del software basada en componentes (ISBC por sus siglas en inglés) es un proceso que concede particular importancia al diseño y la construcción de sistemas basados en computadoras que utilizan componentes de software reutilizables [16].

**Definición**

“Componente: parte importante, casi independiente y reemplazable de un sistema que satisface una función clara en el contexto de una arquitectura bien definida.” [16]

Un componente de software se debe diseñar e implementar de forma que pueda utilizarse en muchos programas diferentes. Los componentes reutilizables modernos encapsulan tanto los datos como el proceso que se aplica a estos, lo que permite al ingeniero de software crear aplicaciones nuevas a partir de partes reutilizables [16].

**Significado de Componente para Delphi y Free Pascal**

Para Delphi y Free Pascal un componente es una clase de uso específico lista para usar, que puede ser configurada o utilizada de forma visual desde el entorno de desarrollo. La principal diferencia respecto a una clase normal es que la mayor parte del trabajo se puede hacer de forma visual, ajustando las opciones que se ofrece en el entorno.

En la programación orientada a objetos se codifican una serie de operaciones más o menos laboriosas para preparar los objetos para su uso. Programar estas operaciones requiere su tiempo, por su complejidad y origen de errores. Sin embargo, en la programación basada en componentes todas estas operaciones se realizan de forma visual, para así poder dedicar la atención al problema [17].

Aunque hay muchos tipos, se diferencian claramente dos grupos de componentes: los visuales y los no visuales.

Los componentes visuales son aquellos que, al utilizarlos muestran algún elemento o dibujo en la pantalla y es el usuario del programa el que interactúa con él. El componente es el principal responsable de dibujar en la pantalla lo que sea oportuno, dependiendo del estado, valor de sus atributos, etc. Hay muchos componentes de este tipo, como pueden ser los botones (TButton), etiquetas de texto (TLabel), formas (TShape). [18].

El componente visual se puede dividir en dos tipos:

* **Componentes interactivos:** permiten que el usuario final los manipule introduciendo datos, seleccionando elementos, etc. De forma que estos componentes pueden recibir el foco así como los eventos propios del teclado y del ratón. Normalmente, el propio sistema operativo es el encargado de dibujar el aspecto del componente, haciendo el componente las llamadas correspondientes para que este aspecto cambie.
* **Componentes gráficos:** el propio componente es el encargado de dibujar en la pantalla lo que crea oportuno, bien a través de las funciones básicas del API de Windows y ciertas en Linux con el objeto TCanvas o a través de otras librerías gráficas, como OpenGL, DirectX, etc. Estos componentes no suelen recibir eventos del usuario final aunque si eventos del propio programador, ya que su cometido no suele ir más allá de mostrar ciertos gráficos o imágenes en la pantalla.

**Los** **componentes no visuales:** son aquellos que no aparecen en la ventana y se insertan en un formulario para que el programador los utilice. Son más fáciles de programar que los componentes visuales ya que no tienen ningún tipo de interfaz gráfico. Ejemplos de componentes no visuales podrían ser un temporizador (TTimer), una tabla (TTable) o una conexión a base de datos (TConnection, TSQLConnection, etc.) [18].

Vale destacar que además un componente tiene propiedades y eventos.

**Las propiedades:** son datos públicos del componente muy parecidas a los atributos de una clase, aunque se accede a ellas a través de dos métodos: un método para leer su valor y otro para modificarlo. Existen propiedades de solo lectura en las que se pueden consultar pero no modificar su valor y propiedades de solo escritura. Por ejemplo, las propiedades “Alto” (Width) y “Ancho” (Height) de un botón permiten que un programador pueda cambiar las dimensiones del componente. Cuando el programador cambia alguna de ellas, el componente debe redibujarse en la pantalla para mostrar los nuevos cambios [18].

**Los eventos** son funciones del componente que se ejecutarán automáticamente cuando ocurra algo importante. Un programador puede poner el código que quiera en el evento, y así poder realizar una acción cuando ese algo importante ocurra. En realidad, los eventos son punteros a funciones [18].

Dado que un componente es una clase como cualquier otra, tiene además métodos y atributos.

**Los métodos:** son funciones que permiten realizar acciones. Normalmente, se utilizan métodos para dos tareas distintas: realizar algo importante como repintar en pantalla, cambiar el foco o algo así y para establecer el valor de los atributos internos, haciendo algún tipo de comprobación previa. Como las propiedades pueden ser leídas o escritas a través de métodos, a veces es equivalente la llamada a un método y el cambio de una propiedad [18].

Y por último, los componentes poseen **atributos,** los cuales tienen la misma misión que en programación orientada a objetos, es decir: almacenar datos internos al objeto o clase. En el mundo de los componentes los atributos siempre son internos, de uso privado y se utilizan las propiedades para que un programador pueda leer o establecer un dato. Luego de este análisis, se puede decir que la principal misión del programador de componentes es definir un grupo de propiedades, métodos y eventos para que otros programadores puedan utilizar el componente de forma sencilla y rápida [18].

### 1.4.4 Necesidad e Importancia de crear Componente sobre Software Libre

Debido a que en el sistema SGMA se utilizan componentes propietarios para el cálculo científico y la comunicación con el computador de a bordo, se decide que las futuras versiones del sistema se realizaran sobre software libre por las ventajas que tendría [19]:

* En lo político debido a la no utilización de productos informáticos que demanden la autorización de sus propietarios (licencias) para su explotación. El software libre representa la alternativa para los países pobres y es por concepción propiedad social, si se tiene en cuenta que una vez que comienza a circular rápidamente se encuentra disponible para todos los interesados sin costo alguno o en su defecto a muy bajo costo. Además es desarrollado de forma colectiva y cooperativa, tanto en su creación como en su desarrollo, así como cuantitativa y cualitativamente mostrando su carácter público y sus objetivos de beneficiar a toda la comunidad.
* En lo económico su utilización no implica gastos adicionales por concepto de cambio de plataforma de software, por cuanto es operable en el mismo soporte de hardware con que cuenta el país. La adquisición de cualquiera de sus distributivas puede hacerse de forma gratuita, descargándolas directamente de Internet o en algunos casos a muy bajos precios, se garantiza su explotación con un mínimo de recursos, por tanto no hay que pagar absolutamente nada por su utilización (no requiere de licencia de uso, las cuales son generalmente muy caras), distribución y/o modificación.
* En lo tecnológico permite su adaptación a los contextos de aplicación al contar con su código fuente, lo cual garantiza un mayor por ciento de efectividad, además la corrección de sus errores de programación y obtención de las actualizaciones y nuevas versiones.

Los componentes son completamente reutilizables y funcionales fuera del marco del proyecto previsto, es decir, se pueden emplear en la construcción de cualquier sistema que necesite las funcionalidades implementadas.

### 1.4.5 Herramienta Case de modelación utilizada

**Enterprise Architect**

Las bases de Enterprise Architect (EA, por sus siglas en inglés) [20] están construidas sobre la especificación de UML 2.0, y usa perfiles UML (Lenguaje Unificado de Modelado) para extender el dominio de modelado, mientras que la validación del modelo asegura integridad. Combina procesos de negocio, información y flujos de trabajo en un modelo, usando extensiones gratuitas para Business Process Management Notation (BPMN, por sus siglas en inglés) [21] y el perfil Eriksson-Penker [22].

EA es una herramienta comprensible de diseño y análisis UML, cubre el desarrollo de software, desde los requerimientos a través de las etapas del análisis, modelos de diseño, pruebas y mantenimiento. EA es una herramienta multiusuario, basada en Windows, diseñada para ayudar a construir software robusto y fácil de mantener. Ofrece salida de documentación flexible y de alta calidad. El manual de usuario está disponible en línea [20].

EA provee trazabilidad completa desde el análisis de requerimientos, hasta los artefactos de análisis y diseño, a través de la implementación y el despliegue. Combinados con la ubicación de recursos y tareas incorporadas, los equipos de Administradores de Proyectos y Calidad están equipados con la información que ellos necesitan para ayudarles a entregar proyectos en tiempo [20].

EA soporta generación e ingeniería inversa de código fuente para muchos lenguajes populares, incluyendo C++, C#, Java, Delphi, Visual Basic y PHP. Con un editor de código fuente con resaltador de sintaxis incorporado, EA le permite navegar y explorar su modelo de código fuente en el mismo ambiente. Las plantillas de generación de código le permiten personalizar el código fuente generado a las especificaciones que se desee [20].

**Características Principales**

* Diseño y construcción de UML.
* Casos de Uso, Modelos Lógico, Dinámico y Físico.
* Extensiones personalizadas para modelado de procesos.
* Documentación de alta calidad compatible con MS Word.
* Intuitivo y simple de usar.
* Modelado de Datos, Ingeniería directa de Base de Datos a DDL e ingeniería inversa de Base de Datos desde ODBC.
* Multiusuario.
* Ingeniería de Código Directa e Inversa.
* Soporte para ActionScript 2.0, Java, C#, C++, Delphi, Visual Basic, Python y PHP.
* Facilidad de Importación/Exportación XMI.
* Corrector Ortográfico.

Por las características antes expuestas, y luego de una investigación comparativa respecto a otras herramientas de modelación, se decide utilizar Enterprise Architect 6.0, ya que esta es una herramienta fácil de usar para el desarrollo de los componentes a implementar. Un factor decisivo para la selección de EA es la facilidad que brinda de generar código para el lenguaje Pascal, el cual es utilizado en la implementación de los componentes. La mayoría de las herramientas consultadas carecen de dicha funcionalidad, solamente Rational Rose permite la generación de código para el lenguaje Pascal presentando el inconveniente de necesitar un plugin el cual no es de fácil adquisición.

### 1.4.6 Metodología utilizada: Extreme Programming (XP)

Programación Extrema, más conocida por sus siglas en inglés XP, surge en 1996. Sus antecedentes se remontan a mediados de la década de los 80, cuando Kent Beck y Ward Cunningham probaron nuevas formas de desarrollar software, diferentes a las que se habían utilizado hasta el momento, trabajando en un grupo de investigación de Tektronix. Posteriormente, en los 90, Beck comenzó un proyecto en DaimlerChrysler, conocido como C3 (Chrysler Comprehensive Compensation). A este proyecto se le considera los cimientos de XP [23].

XP se desarrolla en torno a cuatro valores fundamentales: comunicación, simplicidad, retroalimentación y coraje. Su ciclo de vida consta de 6 etapas: exploración, planificación, iteraciones, producción, mantenimiento y muerte. En cada una de estas etapas intervienen distintos roles y se llevan a cabo prácticas como la programación en parejas, pruebas y el cliente presente, entre otras.

Kent Beck, autor de XP, la define como una metodología ágil para el desarrollo de software destinada, a ser utilizada por equipos de desarrollo, pequeños y medianos (de 2 a 10 miembros) que se enfrenten a proyectos con requerimientos imprecisos o cambiantes. Las relaciones desarrollador-desarrollador y desarrolladores - cliente son pilares fundamentales en esta metodología [11]. La clave del éxito radica en adoptar al cliente, como miembro del equipo de desarrollo.

Elementos que distinguen a XP de las restantes metodologías para el desarrollo de software [11]:

* Concreta, rápida y continua retroalimentación mediante ciclos cortos.
* Su creciente estrategia de programación, que comienza con un plan global, y se espera que evolucione a lo largo de la vida del proyecto.
* Su flexibilidad ante los cambios en los requerimientos o el negocio.
* Su seguridad y confianza en la comunicación oral, el código y las pruebas.
* Su diseño evolutivo a lo largo de la vida del proyecto.
* Estrecha colaboración entre desarrolladores y el cliente.

XP, propone varios roles, como toda metodología de desarrollo de software. Cada rol, conlleva ciertas responsabilidades; algunas más sencillas que otras.

XP brinda siete roles: programador, cliente, encargado de pruebas, encargado de seguimiento, entrenador, consultor y gestor. Una persona puede asumir varios roles en el mismo proyecto [11].

Con el objetivo de lograr el éxito, se definen una serie de buenas prácticas, para la adopción de XP como metodología para el desarrollo. Las 12 prácticas que propone XP no son más que la unión de viejas con nuevas prácticas y algunas modificaciones a otras [11]. El mérito de XP es integrarlas de una forma efectiva y complementarlas con otras ideas desde la perspectiva del negocio, los valores humanos y el trabajo en equipo [24].

**Guía de Aplicación de XP**

Para desarrollar utilizando XP se utiliza la siguiente guía, que explica paso a paso que hacer en cada fase [15]:

**Exploración**

1. Los desarrolladores y el cliente elaboran la metáfora.
2. El cliente redacta las historias de usuario.
3. Los desarrolladores dividen las historias de usuario en tareas de programación y calculan los puntos estimados de cada historia de usuario
4. Los desarrolladores estudian las tecnologías a utilizar.
5. Los desarrolladores construyen uno o varios prototipos del sistema.
6. Los desarrolladores realizan una propuesta inicial de la arquitectura del sistema.
7. El cliente redacta los casos de prueba ayudado por el encargado de pruebas.

**Planificación**

1. El cliente asigna prioridad a las historias de usuario.
2. Los desarrolladores estiman el esfuerzo total.
3. Los desarrolladores estiman la velocidad del equipo.
4. El cliente y los desarrolladores elaboran el plan de entrega.
5. Actualizar documentación.

**Iteraciones**

1. El cliente reasigna prioridad a las historias de usuario.
2. El cliente y los desarrolladores elaboran el plan de iteración.
3. Los desarrolladores definen el diseño preliminar.
4. El cliente y los desarrolladores mejoran los casos de prueba.
5. Los programadores comienzan la implementación.
6. Los desarrolladores mejoran el diseño.
7. El cliente y los desarrolladores completan y actualizan el plan de iteración.
8. Actualizar documentación

**Producción**

1. Completar y actualizar el plan de entrega
2. El cliente y los desarrolladores se reúnen diariamente para informar en que continúa trabajando cada cual.
3. El cliente y los desarrolladores realizan continuas pruebas al sistema.

**Mantenimiento**

1. El cliente y los desarrolladores comienzan una nueva entrega, con lo que se regresa a la fase de exploración.

**Muerte**

1. Muerte del proyecto.

Para el desarrollo de este sistema se utilizó la metodología XP teniendo en cuenta que tiene gran capacidad de respuesta a los cambios, lo cual se espera para el desarrollo de los componentes. Permite una interacción entre cliente y equipo de desarrollo, y se utiliza para el desarrollo de sistemas con poca envergadura.

## 1.5 Conclusiones

En este capítulo se realiza un estudio de temas como el procesamiento de señales el cual se concentra en la representación, transformación y manipulación de señales, y de la información que ellas contienen. Dentro de procesamiento de señales tiene gran importancia el procesamiento digital de señales, ya que este ha sido aplicado en la últimas tres décadas en el desarrollo de los ordenadores digitales y la fabricación de circuitos integrados, obteniéndose grandes avances tecnológicos. Los filtros digitales son unos de los más usados dentro del procesamiento digital de señales, debido a que estos son una componente esencial en toda cadena de comunicación, y constituyen la base del procesamiento de señales, ya que pueden aplicarse a señales de todo tipo (sonidos, imágenes, vídeo, vibraciones sísmicas, etc.).

Se hizo un estudio del IDE de desarrollo Lazarus/FPC el cual se caracteriza por ser Multiplataforma y no requerir licencia para desarrollar ya que es un software libre y ser altamente compatible con Delphi. La herramienta Enterprise Architect se escoge para la modelación, entre otras cosas por la facilidad que brinda de generar código para el lenguaje Pascal. Se decide utilizar la metodología XP teniendo en cuenta que tiene gran capacidad de respuesta a los cambios, lo cual se espera para el desarrollo de los componentes. Permite una interacción entre cliente y equipo de desarrollo, y se utiliza para el desarrollo de sistemas con poca envergadura.

# Capítulo 2 Exploración y Planificación

## Introducción

En este capítulo se documentan las dos primeras fases del ciclo de vida de la metodología XP, aplicadas en el desarrollo de los componentes. Estas fases son Exploración y Planificación, las cuales constan de pasos sumamente importantes para el surgimiento de cualquier proyecto desarrollado utilizando la metodología XP.

## Elaboración de la metáfora

Cada proyecto XP es guiado por una metáfora. En ocasiones está es bien sencilla como firmar un acuerdo entre desarrolladores y clientes; en otras la metáfora requiere de una explicación. La metáfora ayuda a todos, clientes y desarrolladores, a entender los elementos básicos del sistema y sus relaciones. Su objetivo es guiar el desarrollo del sistema a través de una idea, compartida entre clientes y desarrolladores, sobre cómo funcionará el futuro sistema.

La metáfora en XP reemplaza una gran parte de la arquitectura en etapas tempranas del desarrollo; debido a que la arquitectura se asume como evolutiva por lo riesgoso que resultaría para el sistema definirla totalmente desde un inicio. La metáfora es como la arquitectura del sistema, lo que más fácil de comunicar y de elaborar [15].

Para el desarrollo de este trabajo la metáfora elaborada es:

Implementar una serie de componentes en el entorno de desarrollo Lazarus /FPC. El objetivo de estos componentes es filtrar una señal digital, para el procesamiento digital de señales.

## 2.3 Exploración

En la fase de exploración ocurren una serie de acontecimientos que preparan al equipo de desarrollo y al cliente para comenzar la próxima fase. Es donde se gana confianza en las herramientas que se utilizarán. También los miembros del equipo de desarrollo se conocen y aprenden a confiar unos en otros. Durante la exploración los programadores prueban cada pieza de tecnología que se supone utilicen en el sistema [15].

En esta fase, se realiza un estudio y análisis profundo de las tecnologías que se van a utilizar para dar cumplimiento a los objetivos trazados, descrito en el capítulo 1.

Esta fase comienza el 18 de octubre del 2010 y finaliza el 7 de enero del 2011.

### 2.3.1 Historias de usuarios

Las historias de usuario son tarjetas, donde se especifican los requisitos del software a implementar. En estas se describen brevemente las características que debe poseer el software, ya sean requisitos funcionales o no funcionales. Las historias de usuario deben ser redactadas utilizando un lenguaje común a todos, claro y sencillo, véase anexo 1.

Las historias de usuario se estiman para un tiempo ideal de programación, donde se trabaja sin interrupciones ni distracciones.

Beck y Fowler realizan una definición de las “Semanas ideales” como el número de semanas que le toma a una pareja de programadores, dedicada 100% a la labor de implementar una historia de usuario, lograr implementarla [25].

En la tabla 1 se muestran las Historias de Usuario diseñadas para el desarrollo de cada uno de los componentes.

Tabla Historias de usuario

|  |  |  |  |
| --- | --- | --- | --- |
| 1. **Historias de Usuario** | | | |
| **Número** | **Nombre** | **Prioridad** | **Descripción** |
| **1** | Especificar el orden del filtro | Alta | El usuario puede especificar el valor del orden del filtro que desea. |
| **2** | Especificar la frecuencia de corte del filtro | Alta | El usuario puede especificar el valor de la frecuencia de corte que desea. |
| **3** | Diseñar el filtro FilterIIRLPButterworth | Alta | Se diseña el filtro para que responda según las características de un filtro pasa bajo Butterworth. |
| **4** | Filtrar señal a través del filtro FilterIIRLPButterworth | Alta | Se filtra una señal a través del filtro FilterIIRLPButterworth. |
| **5** | Diseñar el filtro FilterIIRLPChebyshev | Alta | Se diseña el filtro para que responda según las características de un filtro pasa bajo Chebyshev. |
| **6** | Filtrar señal a través del filtro FilterIIRLPChebyshev | Alta | Se filtra una señal a través del filtro FilterIIRLPChebyshev. |
| **7** | Diseñar el filtro FilterIIRLPBessel | Alta | Se diseña el filtro para que responda según las características de un filtro pasa bajo Bessel. |
| **8** | Filtrar señal a través del filtro FilterIIRLPBessel | Alta | Se filtra una señal a través del filtro FilterIIRLPBessel. |
| **9** | Diseñar el filtro FilterIIRHPButterworth | Alta | Se diseña el filtro para que responda según las características de un filtro pasa alto Butterworth. |
| **10** | Filtrar señal a través del filtro FilterIIRHPButterworth | Alta | Se filtra una señal a través del filtro FilterIIRHPButterworth. |
| **11** | Diseñar el filtro FilterIIRHPChebyshev | Alta | Se diseña el filtro para que responda según las características de un filtro pasa alto Chebyshev. |
| **12** | Filtrar señal a través del filtro FilterIIRHPChebyshev | Alta | Se filtra una señal a través del filtro FilterIIRHPChebyshev. |
| **13** | Diseñar el filtro FilterIIRHPBessel | Alta | Se diseña el filtro para que responda según las características de un filtro pasa alto Bessel. |
| **14** | Filtrar señal a través del filtro FilterIIRHPBessel | Alta | Se filtra una señal a través del filtro FilterIIRHPBessel. |
| **15** | Diseñar el filtro FilterIIRBPButterworth | Alta | Se diseña el filtro para que responda según las características de un filtro pasa banda Butterworth. |
| **16** | Filtrar señal a través del filtro FilterIIRBPButterworth | Alta | Se filtra una señal a través del filtro FilterIIRBPButterworth. |
| **17** | Diseñar el filtro FilterIIRBPChebyshev | Alta | Se diseña el filtro para que responda según las características de un filtro pasa banda Chebyshev. |
| **18** | Filtrar señal a través del filtro FilterIIRBPChebyshev | Alta | Se filtra una señal a través del filtro FilterIIRBPChebyshev. |
| **19** | Diseñar el filtro FilterIIRBPBessel | Alta | Se diseña el filtro para que responda según las características de un filtro pasa banda Bessel. |
| **20** | Filtrar señal a través del filtro FilterIIRPBandBessel | Alta | Se filtra una señal a través del filtro FilterIIRPBandBessel. |
| **21** | Diseñar el filtro FilterIIRBSButterworth | Alta | Se diseña el filtro para que responda según las características de un filtro supresor de banda Butterworth. |
| **22** | Filtrar señal a través del filtro FilterIIRBSButterworth | Alta | Se filtra una señal a través del filtro FilterIIRBSButterworth. |
| **23** | Diseñar el filtro FilterIIRBSChebyshev | Alta | Se diseña el filtro para que responda según las características de un filtro supresor de banda Chebyshev. |
| **24** | Filtrar señal a través del filtro FilterIIRBSChebyshev | Alta | Se filtra una señal a través del filtro FilterIIRBSChebyshev. |
| **25** | Diseñar el filtro FilterIIRBSBessel | Alta | Se diseña el filtro para que responda según las características de un filtro supresor de banda Bessel. |
| **26** | Filtrar señal a través del filtro FilterIIRBSBessel | Alta | Se filtra una señal a través del filtro FilterIIRBSBessel. |

### 2.3.2 Tareas de programación

Las tareas de programación son funcionalidades que los programadores conocen que el sistema debe hacer. Deben ser estimables, y poder ser implementadas entre uno y tres días ideales. La mayoría de las tareas de programación se derivan directamente de las historias de usuario [15].

**Tarjetas de tareas de programación**

Las tareas de programación, según propone Beck deben escribirse en las tarjetas de historia [11].

**Campos de la tabla tareas de programación**

**Número tarea:** Índice de la tarea de programación. Es un número único que se le asigna a cada tarea de programación que pertenece a una historia de usuario determinada con el fin de lograr una mejor organización de éstas [15].

**Nombre tarea:** Nombre de la tarea de programación. Debe ser descriptivo, en la medida de las posibilidades, de lo que se realizará y no muy extenso [15].

**Descripción:** Se describe qué es lo que se desea realizar. La descripción debe ser corta y precisa [15].

**Puntos estimados:** Tiempo estimado que demorará la implementación de la tarea de programación. Tiempo ideal en que se estima se implementará la tarea de programación [15].

Un punto de estima equivale a un día ideal de trabajo de 8 horas.

A continuación se muestran las tablas que contienen las tareas de programación asignadas a cada historia de usuario.

Tabla Tareas de programación pertenecientes a la Historia de Usuario 1 “Especificar el orden del filtro”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Obtener un algoritmo que permita especificar el orden del filtro | Obtener un algoritmo que permita al usuario especificar el valor del orden del filtro. | 0.5 |
| **2** | Implementar el algoritmo | Implementar el algoritmo obtenido para especificar el orden del filtro, en Lazarus/FPC. | 1 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 2 “Especificar la frecuencia de corte del filtro”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Obtener un algoritmo que permita especificar el valor de la frecuencia de corte del filtro | Obtener un algoritmo que permita al usuario especificar el valor de la frecuencia de corte del filtro. | 1 |
| **2** | Implementar el algoritmo | Implementar el algoritmo obtenido para especificar el valor de la frecuencia de corte del filtro, en Lazarus/FPC. | 1.5 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 3 “Diseñar el filtro FilterIIRLPButterworth”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Crear el componente  FilterIIRLPButterworth | Crear el componente en el entorno Lazarus/FPC. | 0.1 |
| **2** | Establecer las funcionalidades del filtro FilterIIRLPButterworth. | Se determinan las funcionalidades de un filtro pasa bajo Butterworth. | 1.3 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 4 “Filtrar señal a través del filtro FilterIIRLPButterworth”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Obtener un algoritmo que permita filtrar una señal, siguiendo las características y funcionalidades del filtro FilterIIRLPButterworth. | Obtener un algoritmo que permita filtrar una señal siguiendo las características del filtro pasa bajo Butterworth. | 2 |
| **2** | Implementar el algoritmo | Implementar el algoritmo obtenido para filtrar una señal, en Lazarus/FPC. | 3 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 5 “Diseñar el filtro FilterIIRLPChebyshev”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Crear el componente FilterIIRLPChebyshev | Crear el componente en el entorno Lazarus/FPC. | 0.1 |
| **2** | Establecer las funcionalidades del filtro FilterIIRLPChebyshev. | Se determinan las funcionalidades de un filtro pasa bajo Chebyshev. | 1.4 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 6 “Filtrar señal a través del filtro FilterIIRLPChebyshev”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Obtener un algoritmo que permita filtrar una señal, siguiendo las características y funcionalidades del filtro FilterIIRLPChebyshev. | Obtener un algoritmo que permita filtrar una señal siguiendo las características del filtro pasa bajo Chebyshev. | 1.8 |
| **2** | Implementar el algoritmo | Implementar el algoritmo obtenido para filtrar una señal, en Lazarus/FPC. | 2.7 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 7 “Diseñar el filtro FilterIIRLPBessel”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Crear el componente FilterIIRLPBessel | Crear el componente en el entorno Lazarus/FPC. | 0.1 |
| **2** | Establecer las funcionalidades del filtro FilterIIRLPBessel. | Se determinan las funcionalidades de un filtro pasa bajo Bessel. | 1.5 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 8 “Filtrar señal a través del filtro FilterIIRLPBessel”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Obtener un algoritmo que permita filtrar una señal, siguiendo las características y funcionalidades del filtro FilterIIRLPBessel. | Obtener un algoritmo que permita filtrar una señal siguiendo las características del filtro pasa bajo Bessel. | 1.7 |
| **2** | Implementar el algoritmo | Implementar el algoritmo obtenido para filtrar una señal, en Lazarus/FPC. | 2.6 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 9 “Diseñar el filtro FilterIIRHPButterworth”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Crear el componente  FilterIIRHPButterworth | Crear el componente en el entorno Lazarus/FPC. | 0.1 |
| **2** | Establecer las funcionalidades del filtro FilterIIRHPButterworth. | Se determinan las funcionalidades de un filtro pasa alto Butterworth. | 1 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 10 “Filtrar señal a través del filtro FilterIIRHPButterworth”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Obtener un algoritmo que permita filtrar una señal, siguiendo las características y funcionalidades del filtro FilterIIRHPButterworth. | Obtener un algoritmo que permita filtrar una señal siguiendo las características del filtro pasa alto Butterworth. | 0.5 |
| **2** | Implementar el algoritmo | Implementar el algoritmo obtenido para filtrar una señal, en Lazarus/FPC. | 1 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 11 “Diseñar el filtro FilterIIRHPChebyshev”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Crear el componente FilterIIRHPChebyshev | Crear el componente en el entorno Lazarus/FPC. | 0.1 |
| **2** | Establecer las funcionalidades del filtro FilterIIRHPChebyshev. | Se determinan las funcionalidades de un filtro pasa alto Chebyshev. | 0.7 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 12 “Filtrar señal a través del filtro FilterIIRHPChebyshev”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Obtener un algoritmo que permita filtrar una señal, siguiendo las características y funcionalidades del filtro FilterIIRHPChebyshev. | Obtener un algoritmo que permita filtrar una señal siguiendo las características del filtro pasa alto Chebyshev. | 0.5 |
| **2** | Implementar el algoritmo | Implementar el algoritmo obtenido para filtrar una señal, en Lazarus/FPC. | 1 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 13 “Diseñar el filtro FilterIIRHPBessel”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Crear el componente FilterIIRHPBessel | Crear el componente en el entorno Lazarus/FPC. | 0.1 |
| **2** | Establecer las funcionalidades del filtro FilterIIRHPBessel. | Se determinan las funcionalidades de un filtro pasa alto Bessel. | 0.6 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 14 “Filtrar señal a través del filtro FilterIIRHPBessel”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Obtener un algoritmo que permita filtrar una señal, siguiendo las características y funcionalidades del filtro FilterIIRHPBessel. | Obtener un algoritmo que permita filtrar una señal siguiendo las características del filtro pasa alto Bessel. | 0.5 |
| **2** | Implementar el algoritmo | Implementar el algoritmo obtenido para filtrar una señal, en Lazarus/FPC. | 1 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 15 “Diseñar el filtro FilterIIRBPButterworth”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Crear el componente  FilterIIRBPButterworth | Crear el componente en el entorno Lazarus/FPC. | 0.1 |
| **2** | Establecer las funcionalidades del filtro FilterIIRBPButterworth. | Se determinan las funcionalidades de un filtro pasa banda Butterworth. | 1.1 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 16 “Filtrar señal a través del filtro FilterIIRBPButterworth”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Obtener un algoritmo que permita filtrar una señal, siguiendo las características y funcionalidades del filtro FilterIIRBPButterworth. | Obtener un algoritmo que permita filtrar una señal siguiendo las características del filtro pasa banda Butterworth. | 0.5 |
| **2** | Implementar el algoritmo | Implementar el algoritmo obtenido para filtrar una señal, en Lazarus/FPC. | 0.8 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 17 “Diseñar el filtro FilterIIRBPChebyshev”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Crear el componente FilterIIRBPChebyshev | Crear el componente en el entorno Lazarus/FPC. | 0.1 |
| **2** | Establecer las funcionalidades del filtro FilterIIRBPChebyshev. | Se determinan las funcionalidades de un filtro pasa banda Chebyshev. | 0.5 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 18 “Filtrar señal a través del filtro FilterIIRBPChebyshev”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Obtener un algoritmo que permita filtrar una señal, siguiendo las características y funcionalidades del filtro FilterIIRBPChebyshev. | Obtener un algoritmo que permita filtrar una señal siguiendo las características del filtro pasa banda Chebyshev. | 0.5 |
| **2** | Implementar el algoritmo | Implementar el algoritmo obtenido para filtrar una señal, en Lazarus/FPC. | 0.8 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 19 “Diseñar el filtro FilterIIRBPBessel”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Crear el componente FilterIIRBPBessel | Crear el componente en el entorno Lazarus/FPC. | 0.1 |
| **2** | Establecer las funcionalidades del filtro FilterIIRBPBessel. | Se determinan las funcionalidades de un filtro pasa banda Bessel. | 0.4 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 20 “Filtrar señal a través del filtro FilterIIRBPBessel”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Obtener un algoritmo que permita filtrar una señal, siguiendo las características y funcionalidades del filtro FilterIIRBPBessel. | Obtener un algoritmo que permita filtrar una señal siguiendo las características del filtro pasa banda Bessel. | 0.5 |
| **2** | Implementar el algoritmo | Implementar el algoritmo obtenido para filtrar una señal, en Lazarus/FPC. | 0.8 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 21 “Diseñar el filtro FilterIIRBSButterworth”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Crear el componente  FilterIIRBSButterworth | Crear el componente en el entorno Lazarus/FPC. | 0.1 |
| **2** | Establecer las funcionalidades del filtro FilterIIRBSButterworth. | Se determinan las funcionalidades de un filtro supresor de banda Butterworth. | 1 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 22 “Filtrar señal a través del filtro FilterIIRBSButterworth”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Obtener un algoritmo que permita filtrar una señal, siguiendo las características y funcionalidades del filtro FilterIIRBSButterworth. | Obtener un algoritmo que permita filtrar una señal siguiendo las características del filtro supresor de banda Butterworth. | 0.4 |
| **2** | Implementar el algoritmo | Implementar el algoritmo obtenido para filtrar una señal, en Lazarus/FPC. | 0.7 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 23 “Diseñar el filtro FilterIIRBSChebyshev”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Crear el componente FilterIIRBSChebyshev | Crear el componente en el entorno Lazarus/FPC. | 0.1 |
| **2** | Establecer las funcionalidades del filtro FilterIIRBSChebyshev. | Se determinan las funcionalidades de un filtro supresor de banda Chebyshev. | 0.4 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 24 “Filtrar señal a través del filtro FilterIIRBSChebyshev”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Obtener un algoritmo que permita filtrar una señal, siguiendo las características y funcionalidades del filtro FilterIIRBSChebyshev. | Obtener un algoritmo que permita filtrar una señal siguiendo las características del filtro supresor de banda Chebyshev. | 0.4 |
| **2** | Implementar el algoritmo | Implementar el algoritmo obtenido para filtrar una señal, en Lazarus/FPC. | 0.7 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 25 “Diseñar el filtro FilterIIRBSBessel”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Crear el componente FilterIIRBSBessel | Crear el componente en el entorno Lazarus/FPC. | 0.1 |
| **2** | Establecer las funcionalidades del filtro FilterIIRBSBessel. | Se determinan las funcionalidades de un filtro supresor de banda Bessel. | 0.4 |

Tabla Tareas de programación pertenecientes a la Historia de Usuario 26 “Filtrar señal a través del filtro FilterIIRBSBessel”

|  |  |  |  |
| --- | --- | --- | --- |
| **Número**  **tarea** | **Nombre**  **tarea** | **Descripción** | **Puntos estimados** |
| **1** | Obtener un algoritmo que permita filtrar una señal, siguiendo las características y funcionalidades del filtro FilterIIRBSBessel. | Obtener un algoritmo que permita filtrar una señal siguiendo las características del filtro supresor de banda Bessel. | 0.4 |
| **2** | Implementar el algoritmo | Implementar el algoritmo obtenido para filtrar una señal, en Lazarus/FPC. | 0.7 |

### 2.3.3 Casos de Prueba

Los casos de prueba son pruebas funcionales o unitarias que se realizan al sistema para comprobar su funcionamiento. Las pruebas funcionales no son más que pruebas escritas desde la perspectiva del cliente, y las pruebas unitarias son pruebas escritas desde la perspectiva del programador [15].

“Las pruebas unitarias y funcionales son el corazón de XP” [11], mientras un código no haya sido probado no existe. Estos pueden ser adicionados o eliminados en cualquier momento. El objetivo es tener una forma de que el cliente conozca cuando una historia de usuario está lista [15].

Existen tres razones, por las que, los casos de prueba deben ser escritos [14]:

1. Capturar muchos de los detalles que se obtienen a partir de las conversaciones entre clientes y desarrolladores.
2. Capturar suposiciones hechas por el cliente.
3. Proveer un criterio que puede ser utilizado para determinar si una historia de usuario está totalmente implementada.

Los casos de prueba se deben escribir antes de comenzar la implementación, pero siempre que sea necesario se puede incluir uno nuevo. No existe restricción de cantidad para estos, se deben escribir casos de prueba hasta que quede claro el objetivo de la historia de usuario y se verifique que cumpla con todos los requerimientos [15].

Se hacen tarjetas individuales para los casos de prueba de cada Historia de Usuario, véase Anexo 2.

**Campos de la tabla Caso de Prueba** [15]

**Número de historia de usuario:** Número de la historia de usuario a la que corresponde. Índice de la historia de usuario a la que se le desea comprobar este aspecto.

**Número:** Índice del caso de prueba. Es un número único que se le asigna a cada caso de prueba que pertenece a una historia de usuario determinada con el fin de lograr una mejor organización de éstos.

**Nombre:** Nombre del caso de prueba. Debe ser descriptivo, en la medida de las posibilidades, de lo que se comprobará y no muy extenso.

**Descripción:** Se describe qué es lo que se desea probar. La descripción debe ser corta y precisa.

En la tabla 28 se muestran los Casos de Prueba pensados para las Historias de Usuario.

Tabla Casos de Prueba para las Historias de Usuario

|  |  |  |  |
| --- | --- | --- | --- |
| **Número Historia de Usuario** | **Número Caso de Prueba** | **Nombre** | **Descripción** |
| **4,6,8,10,12,**  **14,16,18,20,**  **22,24,26** | **1** | Especificar el orden del filtro | El usuario puede especificar el orden del filtro desde la interfaz visual. |
| **4,6,8,10,12,**  **14** | **1** | Especificar una frecuencia de corte | El usuario puede especificar la frecuencia de corte del filtro desde la interfaz visual, para los filtros pasa bajo y pasa alto. |
| **16,18,20,**  **22,24,26** | **1** | Especificar dos frecuencias de corte | El usuario puede especificar las frecuencias de corte 1 y 2 del filtro desde la interfaz visual, para los filtros pasa banda y supresores de banda. |
| **4** | **1** | Filtrar señal a través del filtro FilterIIRLPButterworth | El usuario puede observar la señal filtrada por el componente FilterIIRLPButterworth a través del componente visual TChart. |
| **6** | **1** | Filtrar señal a través del filtro FilterIIRLPChebyshev | El usuario puede observar la señal filtrada por el componente FilterIIRLPChebyshev a través del componente visual TChart. |
| **8** | **1** | Filtrar señal a través del filtro FilterIIRLPBessel | El usuario puede observar la señal filtrada por el componente FilterIIRLPBessel a través del componente visual TChart. |
| **10** | **1** | Filtrar señal a través del filtro FilterIIRHPButterworth | El usuario puede observar la señal filtrada por el componente FilterIIRHPButterworth a través del componente visual TChart. |
| **12** | **1** | Filtrar señal a través del filtro FilterIIRHPChebyshev | El usuario puede observar la señal filtrada por el componente FilterIIRHPChebyshev a través del componente visual TChart. |
| **14** | **1** | Filtrar señal a través del filtro FilterIIRHPBessel | El usuario puede observar la señal filtrada por el componente FilterIIRHPBessel a través del componente visual TChart. |
| **16** | **1** | Filtrar señal a través del filtro FilterIIRBPButterworth | El usuario puede observar la señal filtrada por el componente FilterIIRBPButterworth a través del componente visual TChart. |
| **18** | **1** | Filtrar señal a través del filtro FilterIIRBPChebyshev | El usuario puede observar la señal filtrada por el componente FilterIIRBPChebyshev a través del componente visual TChart. |
| **20** | **1** | Filtrar señal a través del filtro FilterIIRBPChebyshev | El usuario puede observar la señal filtrada por el componente FilterIIRBPBessel a través del componente visual TChart. |
| **22** | **1** | Filtrar señal a través del filtro FilterIIRBSButterworth | El usuario puede observar la señal filtrada por el componente FilterIIRBSButterworth a través del componente visual TChart. |
| **24** | **1** | Filtrar señal a través del filtro FilterIIRBSChebyshev | El usuario puede observar la señal filtrada por el componente FilterIIRBSChebyshev a través del componente visual TChart. |
| **26** | **1** | Filtrar señal a través del filtro FilterIIRBSBessel | El usuario puede observar la señal filtrada por el componente FilterIIRBSBessel a través del componente visual TChart. |

## 

## 2.4 Planificación

Una vez que se culmina la fase de exploración el 11 de febrero del 2011, se comienza la fase de planificación el 14 de febrero, la que termina el 18 de febrero.

Para determinar si es posible terminar en tiempo el desarrollo de los componentes, en esta fase el cliente debe asignarle prioridad a las historias de usuario, es necesario conocer los puntos de historia de usuario por entrega (**PHUe**) que se es capaz de implementar y el esfuerzo total (**Et**) requerido.

### 2.4.1 Estimaciones

Para realizar una buena estimación se debe tener confianza en que los puntos estimados son correctos. En las estimaciones deben participar todos los desarrolladores. El procedimiento es el siguiente: se selecciona una historia de usuario, de ésta se analizan las tareas de programación que la componen y se estima para cada tarea por separado el tiempo ideal que tomará implementarla.

**Calcular puntos estimados**

Historia de usuario 1 = 1.5 puntos estimados

Historia de usuario 2 = 2.5 puntos estimados

Historia de usuario 3 = 1.4 puntos estimados

Historia de usuario 4 = 5 puntos estimados

Historia de usuario 5 = 1.5 puntos estimados

Historia de usuario 6 = 4.5 puntos estimados

Historia de usuario 7 = 1.6 puntos estimados

Historia de usuario 8 = 4.3 puntos estimados

Historia de usuario 9 = 1.1 puntos estimados

Historia de usuario 10 = 1.5 puntos estimados

Historia de usuario 11 = 0.9 puntos estimados

Historia de usuario 12 = 1.5 puntos estimados

Historia de usuario 13 = 0.7 puntos estimados

Historia de usuario 14 = 1.5 puntos estimados

Historia de usuario 15 = 1.2 puntos estimados

Historia de usuario 16 = 1.3 puntos estimados

Historia de usuario 17 = 0.6 puntos estimados

Historia de usuario 18 = 1.3 puntos estimados

Historia de usuario 19 = 0.5 puntos estimados

Historia de usuario 20 = 1.3 puntos estimados

Historia de usuario 21 = 1.1 puntos estimados

Historia de usuario 22 = 1.1 puntos estimados

Historia de usuario 23 = 0.5 puntos estimados

Historia de usuario 24 = 1.1 puntos estimados

Historia de usuario 25 = 0.5 puntos estimados

Historia de usuario 26 = 1.1 puntos estimados

### 2.4.2 Estimar esfuerzo total

El esfuerzo total (**Et**) es la suma de los esfuerzos de cada una de las historias de usuario que componen el sistema.

Dónde:

**Et = Σ PEHU** (2.1)

Dónde: **PEHU** es el punto estimado de la Historia de Usuario.

**Et** = (1.5+2.5+1.4+5+1.5+4.5+1.6+4.3+1.1+1.5+0.9+1.5+0.7+1.5+1.2+1.3+0.6

+1.3+0.5+1.3+1.1+1.1+0.5+1.1+0.5+1.1)

**Et = 41.1 ≈ 42 puntos estimados**

### 2.4.3 Estimar velocidad del equipo

Velocidad del equipo se denomina al número de semanas ideales que un equipo de desarrollo puede realizar en un tiempo determinado [11]. La velocidad es utilizada para determinar cuánto trabajo se realizará en cada iteración y puede ser estimada en cualquier momento del proceso de desarrollo.

El equipo está conformado por un solo programador y como no se conoce su velocidad por valores históricos ni se puede llevar a cabo una iteración inicial para determinarla, por razones de disponer poco tiempo, se le asigna el valor propuesto por Jeffries 1/3.

### 2.4.4 Calcular tiempo real

Para completar el plan de entrega se necesita calcular a cuantos días equivalen 42 puntos estimados, para conocer la fecha exacta de la entrega de los componentes. Se cuenta con la fórmula:

**Tr = Et / Ve** (2.2)

**Tr** = Tiempo real

**Et** = esfuerzo total

**Ve** = velocidad del equipo

**Tr = 42/ 1/3 ≈ 126 días**

### 2.4.5 Calcular puntos de historia de usuario por entrega

Una vez conocida la velocidad del equipo de desarrollo, se puede calcular los puntos de historia de usuario por entrega (**PHUe**). Con éstos se puede saber la cantidad de historias de usuario que se pueden implementar para la entrega. Como no existen limitaciones en cuanto al tiempo de desarrollo, y este resulta menor de 6 meses es posible implementar todas las historias de usuario en una primera entrega.

### 2.4.6 Distribución de historias de usuario por iteración

Para la primera iteración se seleccionó las 8 primeras historias de usuario, “Especificar el orden del filtro”, “Especificar la frecuencia de corte del filtro”, “Diseñar el filtro FilterIIRLPButterworth”, “Filtrar señal a través del filtro FilterIIRLPButterworth”, “Diseñar el filtro FilterIIRLPChebyshev”, “Filtrar señal a través del filtro FilterIIRLPChebyshev”, “Diseñar el filtro FilterIIRLPBessel”, “Filtrar señal a través del filtro FilterIIRLPBessel” . El esfuerzo total de éstas es 22.3 puntos estimados.

Para la segunda iteración se seleccionó las historias de usuario “Diseñar el filtro FilterIIRHPButterworth”, “Filtrar señal a través del filtro FilterIIRHPButterworth”, “Diseñar el filtro FilterIIRHPChebyshev”, “Filtrar señal a través del filtro FilterIIRHPChebyshev”, “Diseñar el filtro FilterIIRHPBessel”, “Filtrar señal a través del filtro FilterIIRHPBessel”. El esfuerzo total de éstas es 7.2 puntos estimados.

Para la tercera iteración se seleccionó las historias de usuario “Diseñar el filtro FilterIIRBPButterworth”, “Filtrar señal a través del filtro FilterIIRBPButterworth”, “Diseñar el filtro FilterIIRBPChebyshev”, “Filtrar señal a través del filtro FilterIIRBPChebyshev”, “Diseñar el filtro FilterIIRBPBessel”, “Filtrar señal a través del filtro FilterIIRBPBessel”. El esfuerzo total de éstas es 6.2 puntos estimados.

Para la cuarta iteración se seleccionó las historias de usuario “Diseñar el filtro FilterIIRBSButterworth”, “Filtrar señal a través del filtro FilterIIRBSButterworth”, “Diseñar el filtro FilterIIRBSChebyshev”, “Filtrar señal a través del filtro FilterIIRBSChebyshev”, “Diseñar el filtro FilterIIRBSBessel”, “Filtrar señal a través del filtro FilterIIRBSBessel”. El esfuerzo total de éstas es 5.4 puntos estimados.

### 2.4.7 Cálculo de puntos de historia de usuario por iteración

Conociendo que son 12 componentes, se decide realizar 4 iteraciones, implementando 3 componentes en cada iteración. En el epígrafe 2.4.6 se detallan las historias de usuario a implementar en cada iteración, teniendo en cuenta la prioridad de cada una.

**Primera Iteración**

Si se conoce que la velocidad es 1/3, y que en la primera iteración se deben implementar los componentes pasa bajo FilterIIRLPButterworth, FilterIIRLPChebyshev y FilterIIRLPBessel, los que requieren un esfuerzo total de 22.3 puntos estimados, entonces se puede calcular el tiempo necesario:

**c = PHUi / Ve** (2.3)

**PHUi = punto de historia de usuario por iteración.**

**Ve = velocidad del equipo**

**c = cantidad de días de la iteración.**

**c = 22.3 / 1/3 = 66.9 ≈ 67 días**

**Segunda Iteración**

Para el caso de la segunda iteración si se mantiene la velocidad de programación de 1/3 y se deben implementar los componentes pasa alto FilterIIRHPButterworth, FilterIIRHPChebyshev y FilterIIRHPBessel, los que requieren un esfuerzo total de 7.2 puntos estimados, entonces se puede calcular el tiempo necesario:

**c = 7.2 / 1/3 = 21.6 ≈ 22 días**

**Tercera Iteración**

Para el caso de la tercera iteración si se mantiene la velocidad de programación de 1/3 se deben implementar los componentes pasa banda FilterIIRBPButterworth, FilterIIRBPChebyshev y FilterIIRBPBessel, los que requieren un esfuerzo total de 6.2 puntos estimados, entonces se puede calcular el tiempo necesario:

**c = 6.2 / 1/3 = 18.6 ≈ 19 días**

**Cuarta Iteración**

Para el caso de la tercera iteración si se mantiene la velocidad de programación de 1/3 se deben implementar los componentes supresores de banda FilterIIRBSButterworth, FilterIIRBSChebyshev y FilterIIRBSBessel, los que requieren un esfuerzo total de 5.4 puntos estimados, entonces se puede calcular el tiempo necesario:

**c = 5.4 / 1/3 = 16.2 ≈ 17 días**

Después de los cálculos realizados la primera iteración demorará 67 días, la segunda 22 días, la tercera 19 días y la cuarta 17 días. Para un total de 126 días.

## 2.5 Plan de Entrega

Con los resultados obtenidos anteriormente se puede elaborar el plan de entrega, el que puede modificarse a lo largo de desarrollo de los componentes, visto que la planificación no se realiza en un momento fijo, sino se lleva a cabo a lo largo del proceso de desarrollo de XP.

En la tabla 29, se define el plan de entrega.

Tabla Plan de Entrega

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **# Entrega** | **# Iteración** | **# H Usuario** | **Estado** | **Descripción** |
| 1 | 1 | 1 | P | Especificar el orden del filtro |
| 2 | P | Especificar la frecuencia de corte del filtro |
| 3 | P | Diseñar el filtro FilterIIRLPButterworth |
| 4 | P | Filtrar señal a través del filtro FilterIIRLPButterworth |
| 5 | P | Diseñar el filtro FilterIIRLPChebyshev |
| 6 | P | Filtrar señal a través del filtro FilterIIRLPChebyshev |
| 7 | P | Diseñar el filtro FilterIIRLPBessel |
| 8 | P | Filtrar señal a través del filtro FilterIIRLPBessel |
| 2 | 9 | P | Diseñar el filtro FilterIIRHPButterworth |
| 10 | P | Filtrar señal a través del filtro FilterIIRHPButterworth |
| 11 | P | Diseñar el filtro FilterIIRHPChebyshev |
| 12 | P | Filtrar señal a través del filtro FilterIIRHPChebyshev |
| 13 | P | Diseñar el filtro FilterIIRHPBessel |
| 14 | P | Filtrar señal a través del filtro FilterIIRHPBessel |
| 3 | 15 | P | Diseñar el filtro FilterIIRBPButterworth |
| 16 | P | Filtrar señal a través del filtro FilterIIRBPButterworth |
| 17 | P | Diseñar el filtro FilterIIRBPChebyshev |
| 18 | P | Filtrar señal a través del filtro FilterIIRBPChebyshev |
| 19 | P | Diseñar el filtro FilterIIRBPBessel |
|  | 20 | P | Filtrar señal a través del filtro FilterIIRBPBessel |
| 4 | 21 | P | Diseñar el filtro FilterIIRBSButterworth |
| 22 | P | Filtrar señal a través del filtro FilterIIRBSButterworth |
| 23 | P | Diseñar el filtro FilterIIRBSChebyshev |
| 24 | P | Filtrar señal a través del filtro FilterIIRBSChebyshev |
| 25 | P | Diseñar el filtro FilterIIRBSBessel |
| 26 | P | Filtrar señal a través del filtro FilterIIRBSBessel |

Desde este momento en lo adelante se comienza a trabajar los 7 días de la semana, haciendo caso omiso a la buena práctica de XP que recomienda trabajar solo 40 horas a la semana. Se reconoce el riesgo de disminuir el rendimiento pero se decide tomarlo.

La entrega de las herramientas se planifica para 126 días después de que comience la primera iteración.

La primera iteración debe comenzar el 11 de enero del 2011 y finalizar el 18 de marzo de 2011. La segunda iteración debe comenzar el día 19 de marzo y terminar el 9 de abril de 2011. La tercera iteración debe comenzar el día 10 de abril y terminar el 28 de abril de 2011 y la cuarta iteración debe comenzar el día 29 de abril y terminar el 15 de mayo del 2011.

## 2.6 Conclusiones

En este capítulo se aplican las dos primeras fases del ciclo de vida de la metodología XP en el desarrollo de los componentes, la Exploración y Planificación.

Dentro de la fase de Exploración se definen un total de 26 historias de usuario, las cuales especifican los requisitos del software a implementar. Para cada historia de usuario se definen las tareas de programación, según las funcionalidades que debe tener el software. Además se definen 15 casos de pruebas que se realizaran con el objetivo de comprobar el funcionamiento del software. Se decide realizar cuatro iteraciones, tres componentes en cada iteración. Para cada iteración se hace una distribución de las historias de usuario que se implementaran en esta, donde para la primera iteración se decide implementar 8 historias de usuario, en la segunda 6, en la tercera 6 y en la cuarta 6, para el total de las 26 historias de usuario definidas anteriormente.

Mediante cálculos se estima que implementar todas las historias de usuario para el desarrollo de los componentes, requiere un esfuerzo total de 42 puntos estimados, equivalente a 126 días en tiempo real. Se estima además, el tiempo que demora implementar cada iteración, este resultado indica que se pueden implementar todas las historias de usuario en el tiempo estimado, posibilitando así, la entrega del proyecto.

# Capítulo 3 Desarrollo de los componentes

## 3.1 Introducción

En este capítulo se documenta la tercera etapa del ciclo de vida de la metodología XP, aplicada en el desarrollo de los componentes. Esta fase denominada iteraciones, es la fase final antes de que esté funcionando la primera versión del software. Se compone de cuatro iteraciones, y en cada una de estas el cliente y/o el programador puede realizar cualquier cambio que considere pertinente para lograr una calidad superior en el software. Se cuenta con un plan de entrega el cual se divide en diferentes planes de iteración. Para cada iteración se crean un grupo de pruebas que correspondan con las historias de usuario estimadas anteriormente. No se culmina la iteración hasta que cada prueba se realice satisfactoriamente.

Además se documentan las tres últimas etapas del ciclo de vida de la metodología XP, aplicadas en el desarrollo de dichos componentes. Estas fases son Producción, Mantenimiento y Muerte. Se realiza una breve explicación de cómo se comportan estas etapas en los componentes que se documentan en este trabajo.

## 3.2 Iteración I

Esta iteración comienza el 11 de enero del 2011. Se realiza el plan de iteración y los casos de prueba. Se parte con certeza de una arquitectura del sistema confiable que será utilizada en el desarrollo de los componentes estimados para esta iteración y durante el desarrollo de las restantes iteraciones.

### 3.2.1 Plan de Iteración

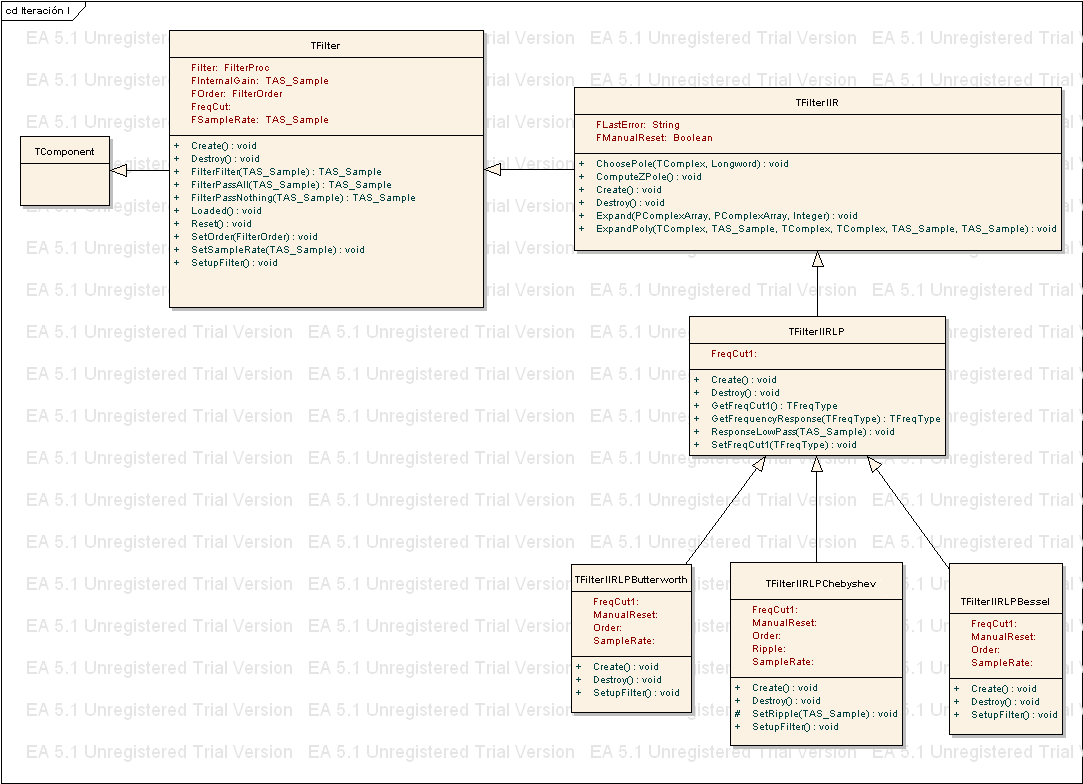
Conociendo las prioridades de las historias de usuario y el plan de entrega, se elabora el plan de iteración I que se muestra en la tabla 30.

Tabla Plan de Iteración I

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **# Entrega** | **# Iteración** | **# H Usuario** | **# Tarea** | **Fecha Inicio Estimada** | **Fecha Inicio Real** | **Fecha Fin Estimada** | **Fecha Fin Real** | **Estado** |
| **1** | **1** | **1** | **1** | **11/01** | **11/01** | **13/01** | **12/01** | **P** |
|  | **2** | **13/01** | **12/01** | **16/01** | **16/01** | **P** |
| **2** | **1** | **16/01** | **17/01** | **19/01** | **18/01** | **P** |
| **2** | **19/01** | **19/01** | **23/01** | **22/01** | **P** |
| **3** | **1** | **23/01** | **22/01** | **23/01** | **22/01** | **p** |
| **2** | **23/01** | **22/01** | **27/01** | **26/01** | **p** |
| **4** | **1** | **27/01** | **26/01** | **02/02** | **02/02** | **p** |
| **2** | **02/02** | **02/02** | **11/02** | **9/02** | **p** |
| **5** | **1** | **11/02** | **9/02** | **11/02** | **9/02** | **p** |
| **2** | **11/02** | **10/02** | **15/02** | **13/02** | **p** |
| **6** | **1** | **15/02** | **13/02** | **21/02** | **20/02** | **p** |
| **2** | **21/02** | **20/02** | **01/03** | **27/02** | **p** |
| **7** | **1** | **01/03** | **27/02** | **01/03** | **27/02** | **p** |
| **2** | **01/03** | **28/02** | **05/03** | **04/03** | **p** |
| **8** | **1** | **05/03** | **04/03** | **10/03** | **08/03** | **p** |
| **2** | **10/03** | **08/03** | **18/03** | **14/03** | **p** |

### 3.2.2 Diseño Preliminar

Al comienzo de esta iteración se realiza un diseño preliminar. Con este se busca conocer de forma general cuales son los objetos involucrados.



**Figura 9 Vista del Diseño Preliminar de la Iteración I**

### 3.2.3 Casos de Prueba

A lo largo de esta iteración se ejecutan los casos de prueba previstos en el epígrafe 2.3.3, que corresponden al desarrollo de los componentes FilterIIRLPButterworth, FilterIIRLPChebyshev y FilterIIRLPBessel.

En la tabla 31 se muestra un nuevo caso de prueba, añadido en esta iteración para mostrar la ganancia de los filtros.

Tabla Casos de Prueba Iteración I

|  |  |  |  |
| --- | --- | --- | --- |
| **#Historia de Usuario** | **#Caso de Prueba** | **Nombre** | **Descripción** |
| **4,6,8** | **1** | Mostrar la ganancia del filtro | El usuario puede observar la ganancia del filtro. |

### 3.2.4 Diseño Final

Después de implementar las historias de usuario y realizar las pruebas pertenecientes a la primera iteración, se tiene un diseño más completo, pues se

adicionaron nuevos atributos y funciones para un mejor funcionamiento de los

componentes FilterIIRLPButterworth, FilterIIRLPChebyshev y FilterIIRLPBessel.

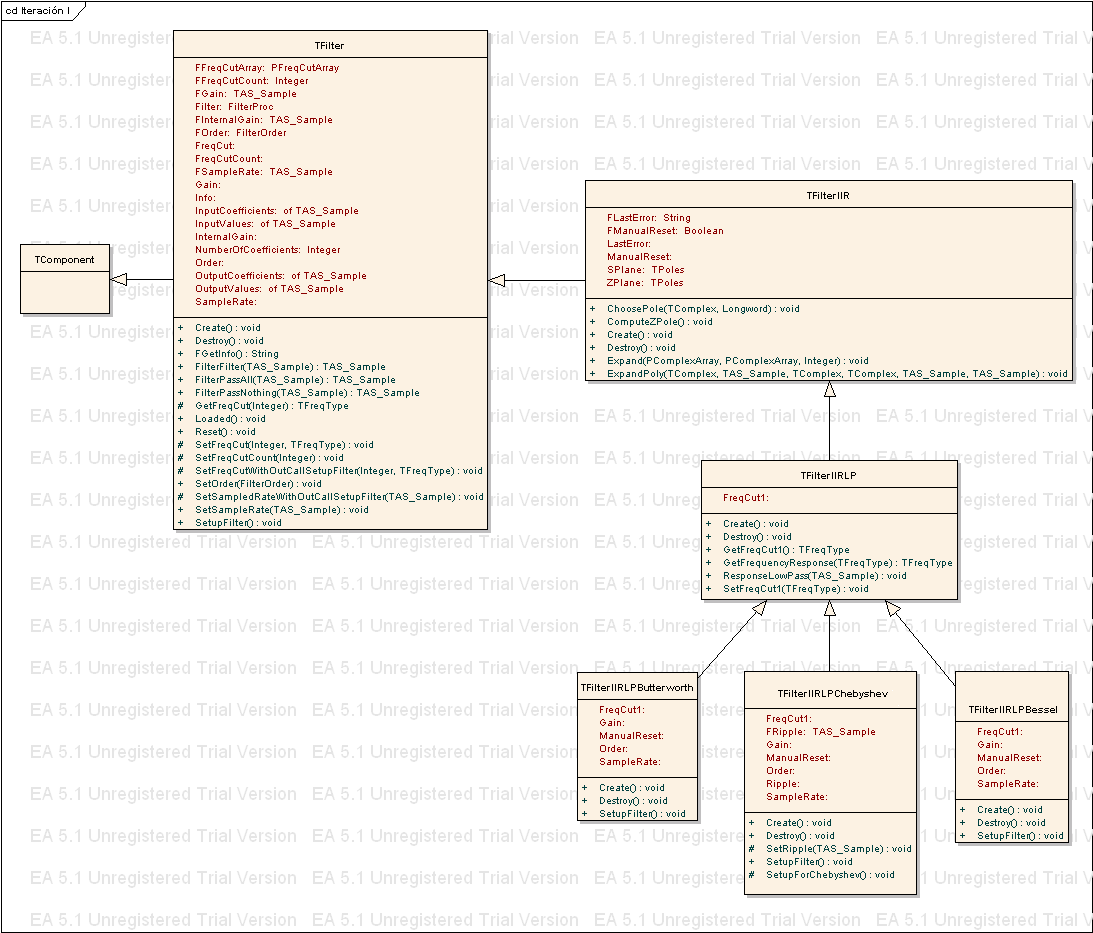


Figura Vista del Diseño Final de la Iteración I

## 3.3 Iteración II

Esta iteración comienza el 15 de marzo del 2011. Se realiza el plan de iteración y los casos de prueba. El cliente revisa la asignación que se le dio a cada historia de usuario. Este se cerciora de la prioridad que tiene cada una por si hay que realizar algún cambio. En este caso se queda igual a como se concibe en la planificación.

### 3.3.1 Plan de Iteración

Conociendo las prioridades de las historias de usuario y el plan de entrega, se pasa a re calcular la velocidad del equipo, ya que la primera iteración termina antes del tiempo estimado; con el fin de elaborar el plan de iteración II.

**Re cálculo de la velocidad del equipo**

**Ve = PHUi / c** (3.1)

**Ve = velocidad del equipo**

**PHUi = punto de historia de usuario para iteración I.**

**c = cantidad de días de la iteración I.**

**Ve = 22.3 / 63 = 0.35**

**Re cálculo del tiempo real para la iteración II**

**c = PHUi / Ve** (3.2)

**c = cantidad de días de la iteración II.**

**PHUi = punto de historia de usuario para iteración II.**

**Ve = velocidad del equipo re calculada**

**c = 7.2 / 0.35 = 20.3 ≈ 21**

Con los resultados obtenidos se aprecia que aumenta la velocidad del equipo, lo que justifica que se halla terminado con antelación la iteración I. Se procede a elaborar el plan de iteración II a partir de la nueva fecha.

Tabla Plan de Iteración II

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **# Entrega** | **# Iteración** | **# H Usuario** | **# Tarea** | **Fecha Inicio Estimada** | **Fecha Inicio Real** | **Fecha Fin Estimada** | **Fecha Fin Real** | **Estado** |
| **1** | **II** | **9** | **1** | **15/03** | **15/03** | **15/03** | **15/03** | **P** |
|  | **2** | **15/03** | **15/03** | **19/03** | **18/03** | **P** |
| **10** | **1** | **19/03** | **18/3** | **20/03** | **19/03** | **P** |
| **2** | **20/03** | **19/03** | **22/03** | **21/03** | **P** |
| **11** | **1** | **22/03** | **21/03** | **22/03** | **21/03** | **p** |
| **2** | **22/03** | **21/03** | **24/03** | **22/03** | **p** |
| **12** | **1** | **24/03** | **22/03** | **26/03** | **24/03** | **p** |
| **2** | **26/03** | **25/03** | **29/03** | **27/03** | **p** |
| **13** | **1** | **29/03** | **27/03** | **29/03** | **27/03** | **p** |
| **2** | **29/03** | **27/03** | **31/04** | **29/03** | **p** |
| **14** | **1** | **31/04** | **30/03** | **02/04** | **01/04** | **p** |
| **2** | **02/04** | **01/04** | **04/04** | **03/04** | **p** |

### 3.3.2 Diseño Preliminar

Al comienzo de esta iteración se realiza un diseño preliminar. Con este se busca conocer de forma general cuales son los objetos involucrados.

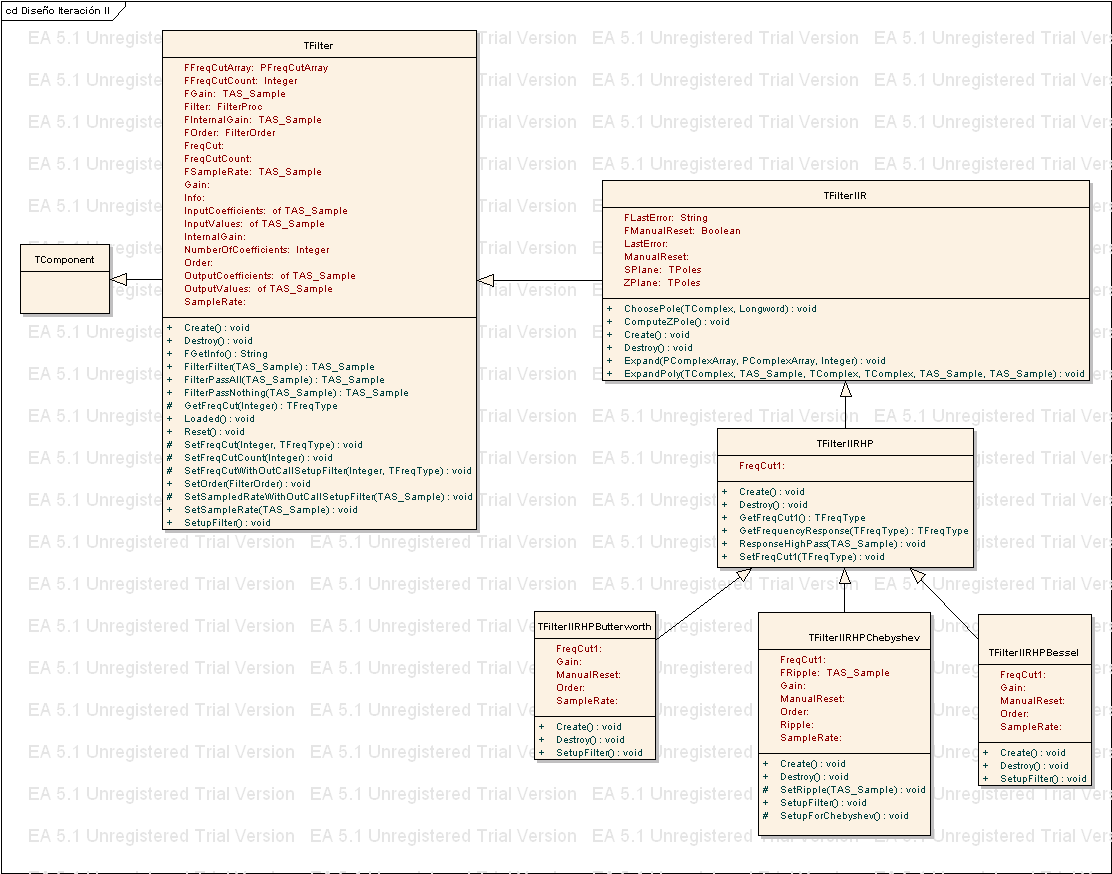


Figura Vista de Diseño Preliminar de la Iteración II

### 3.3.3 Casos de Prueba

A lo largo de esta iteración se ejecutan los casos de prueba previstos en el epígrafe 2.3.3, que corresponden al desarrollo de los componentes FilterIIRHPButterworth, FilterIIRHPChebyshev, FilterIIRHPBessel.

En la tabla 33 se muestra el caso de prueba añadido en esta iteración para mostrar la ganancia de los filtros.

Tabla Casos de Prueba Iteración II

|  |  |  |  |
| --- | --- | --- | --- |
| **#Historia de Usuario** | **#Caso de Prueba** | **Nombre** | **Descripción** |
| **10,12,14** | **1** | Mostrar la ganancia del filtro | El usuario puede observar la ganancia del filtro. |

### 3.3.4 Diseño Final

El diseño final en esta iteración se mantiene igual al diseño preliminar expuesto

en el epígrafe 3.3.2, ya que no hubo ningún cambio en la arquitectura de sistema para la implementación de los componentes FilterIIRHPButterworth, FilterIIRHPChebyshev y FilterIIRHPBessel.

## 

## 3.4 Iteración III

Esta iteración comienza el 4 de abril del 2011. Se realiza el plan de iteración y los casos de prueba. El cliente nuevamente revisa la asignación que se le dio a cada historia de usuario. Este se cerciora de la prioridad que tiene cada una por si hay que realizar algún cambio. En este caso se queda igual a como se concibe en la planificación.

### 3.4.1 Plan de Iteración

Conociendo las prioridades de las historias de usuario y el plan de entrega, se pasa a re calcular la velocidad del equipo, ya que la segunda iteración termina antes del tiempo estimado; con el fin de elaborar el plan de iteración III.

**Re cálculo de la velocidad del equipo**

**Ve = PHUi / c** (3.1)

**Ve = velocidad del equipo**

**PHUi = punto de historia de usuario para iteración II.**

**c = cantidad de días de la iteración II.**

**Ve = 7.2 / 20 = 0.36**

**Re cálculo del tiempo real para la iteración III**

**c = PHUi / Ve** (3.2)

**c = cantidad de días de la iteración III.**

**PHUi = punto de historia de usuario para iteración III.**

**Ve = velocidad del equipo re calculada**

**c = 6.2 / 0.36 = 17.2 ≈ 18 días estimados**

Con los resultados obtenidos se aprecia que aumentó la velocidad del equipo, lo que justifica que se halla terminado con antelación la iteración II. Se procede a elaborar el plan de iteración III a partir de la nueva fecha.

Tabla Plan de Iteración III

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **# Entrega** | **# Iteración** | **# H Usuario** | **# Tarea** | **Fecha Inicio Estimada** | **Fecha Inicio Real** | **Fecha Fin Estimada** | **Fecha Fin Real** | **Estado** |
| **1** | **III** | **15** | **1** | **05/04** | **05/04** | **05/04** | **05/04** | **P** |
|  | **2** | **05/04** | **05/04** | **09/04** | **08/04** | **P** |
| **16** | **1** | **09/04** | **08/04** | **10/04** | **09/04** | **P** |
| **2** | **10/04** | **09/04** | **12/04** | **11/04** | **P** |
| **17** | **1** | **12/04** | **11/04** | **12/04** | **11/04** | **p** |
| **2** | **12/04** | **11/04** | **14/04** | **12/04** | **p** |
| **18** | **1** | **14/04** | **12/04** | **15/04** | **13/04** | **p** |
| **2** | **15/04** | **14/04** | **18/04** | **16/04** | **p** |
| **19** | **1** | **18/04** | **16/04** | **18/04** | **16/04** | **p** |
| **2** | **18/04** | **16/04** | **19/04** | **17/04** | **p** |
| **20** | **1** | **19/04** | **17/04** | **20/04** | **19/04** | **p** |
| **2** | **20/04** | **19/04** | **22/04** | **20/04** | **p** |

### 3.4.2 Diseño Preliminar

Al comienzo de esta iteración se realiza un diseño preliminar. Con este se busca conocer de forma general cuales son los objetos involucrados.

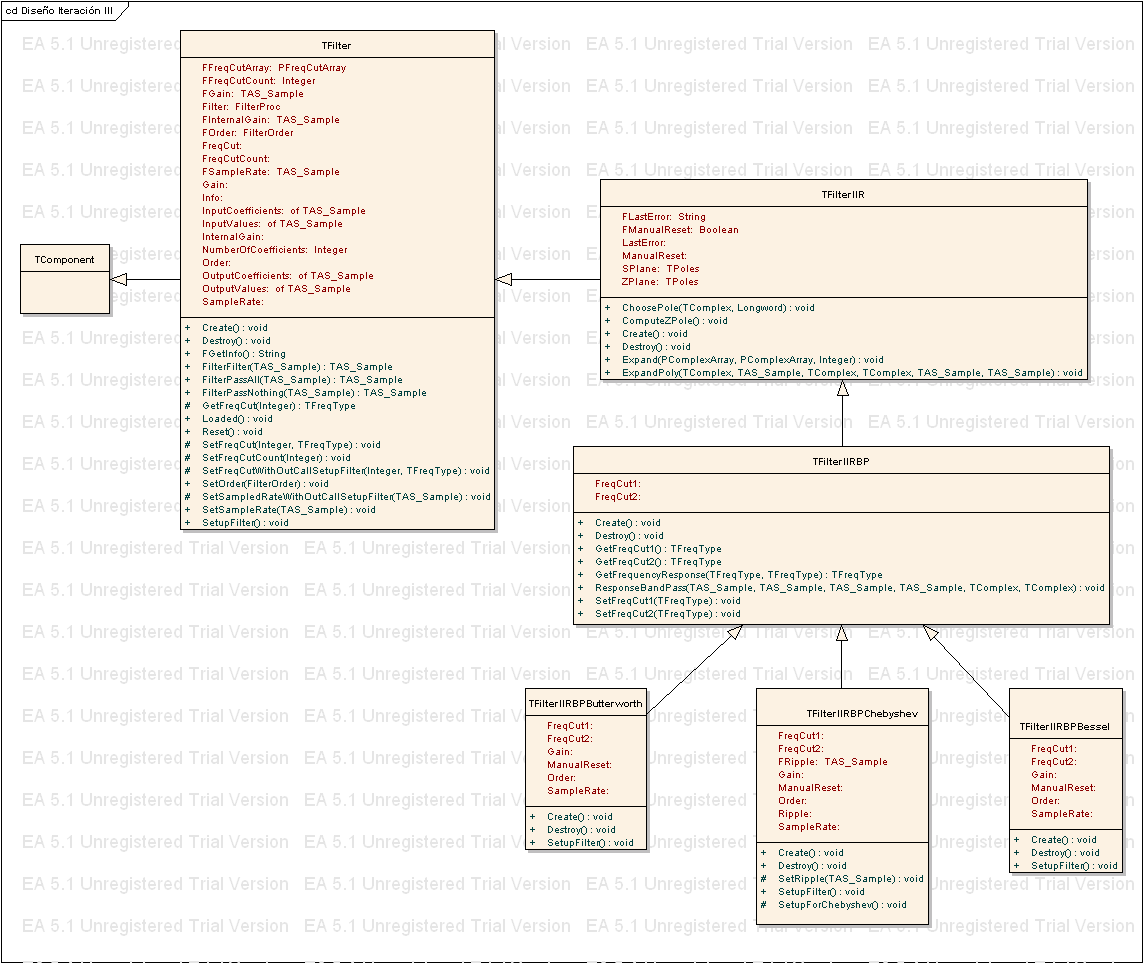


Figura Vista de Diseño Preliminar de la Iteración III

### 3.4.3 Casos de Prueba

A lo largo de esta iteración se ejecutan los casos de prueba previstos en el epígrafe 2.3.3, que corresponden al desarrollo de los componentes FilterIIRBPButterworth, FilterIIRBPChebyshev y FilterIIRBPBessel.

En la tabla 35 se muestra el caso de prueba, añadido en esta iteración para mostrar la ganancia de los filtros.

Tabla Casos de Prueba Iteración III

|  |  |  |  |
| --- | --- | --- | --- |
| **#Historia de Usuario** | **#Caso de Prueba** | **Nombre** | **Descripción** |
| **16,18,20** | **1** | Mostrar la ganancia del filtro | El usuario puede observar la ganancia del filtro. |

### 3.4.4 Diseño Final

El diseño final en esta iteración se mantiene igual al diseño preliminar expuesto

en el epígrafe 3.4.2, ya que no hubo ningún cambio en la arquitectura del sistema para la implementación de los componentes FilterIIRBPButterworth, FilterIIRBPChebyshev y FilterIIRBPBessel.

## 3.5 Iteración IV

Esta iteración comienza el 21 de abril del 2011. Se realiza el plan de iteración y los casos de prueba. El cliente nuevamente revisa la asignación que se le dio

a cada historia de usuario. Este se cerciora de la prioridad que tiene cada una por si hay que realizar algún cambio. En este caso se queda igual a como se concibe en la planificación.

### 3.5.1 Plan de Iteración

Conociendo las prioridades de las historias de usuario y el plan de entrega, se pasa a re calcular la velocidad del equipo, ya que la segunda iteración termina antes del tiempo estimado; con el fin de elaborar el plan de iteración IV.

**Re cálculo de la velocidad del equipo**

**Ve = PHUi / c** (3.1)

**Ve = velocidad del equipo**

**PHUi = punto de historia de usuario para iteración III.**

**c = cantidad de días de la iteración III.**

**Ve = 6.2 / 16 = 0.3875 ≈0.39**

**Re cálculo del tiempo real para la iteración**

**c = PHUi / Ve** (3.2)

**c = cantidad de días de la iteración IV.**

**PHUi = punto de historia de usuario para iteración IV.**

**Ve = velocidad del equipo re calculada**

**c = 5.4 / 0.39 = 13.8 ≈ 14 días estimados**

Con los resultados obtenidos se aprecia que aumentó la velocidad del equipo, lo que justifica que se halla terminado con antelación la iteración III. Se procede a elaborar el plan de iteración IV a partir de la nueva fecha.

Tabla Plan de Iteración IV

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **# Entrega** | **# Iteración** | **# H Usuario** | **# Tarea** | **Fecha Inicio Estimada** | **Fecha Inicio Real** | **Fecha Fin Estimada** | **Fecha Fin Real** | **Estado** |
| **1** | **IV** | **21** | **1** | **21/04** | **21/04** | **21/04** | **21/04** | **P** |
|  | **2** | **21/04** | **21/04** | **23/04** | **22/04** | **P** |
| **22** | **1** | **23/04** | **22/04** | **24/04** | **23/04** | **P** |
| **2** | **24/04** | **23/04** | **26/04** | **25/04** | **P** |
| **23** | **1** | **26/04** | **25/04** | **26/04** | **25/04** | **p** |
| **2** | **27/04** | **26/04** | **28/04** | **27/04** | **p** |
| **24** | **1** | **28/04** | **27/04** | **29/04** | **28/04** | **p** |
| **2** | **29/04** | **28/04** | **01/05** | **30/04** | **p** |
| **25** | **1** | **01/05** | **30/04** | **01/05** | **30/04** | **p** |
| **2** | **01/05** | **30/04** | **02/05** | **02/05** | **p** |
| **26** | **1** | **02/05** | **02/05** | **02/05** | **02/05** | **p** |
| **2** | **03/05** | **03/05** | **04/05** | **04/05** | **p** |

### 3.5.2 Diseño Preliminar

Al comienzo de esta iteración se realiza un diseño preliminar. Con este se busca conocer de forma general cuales son los objetos involucrados.

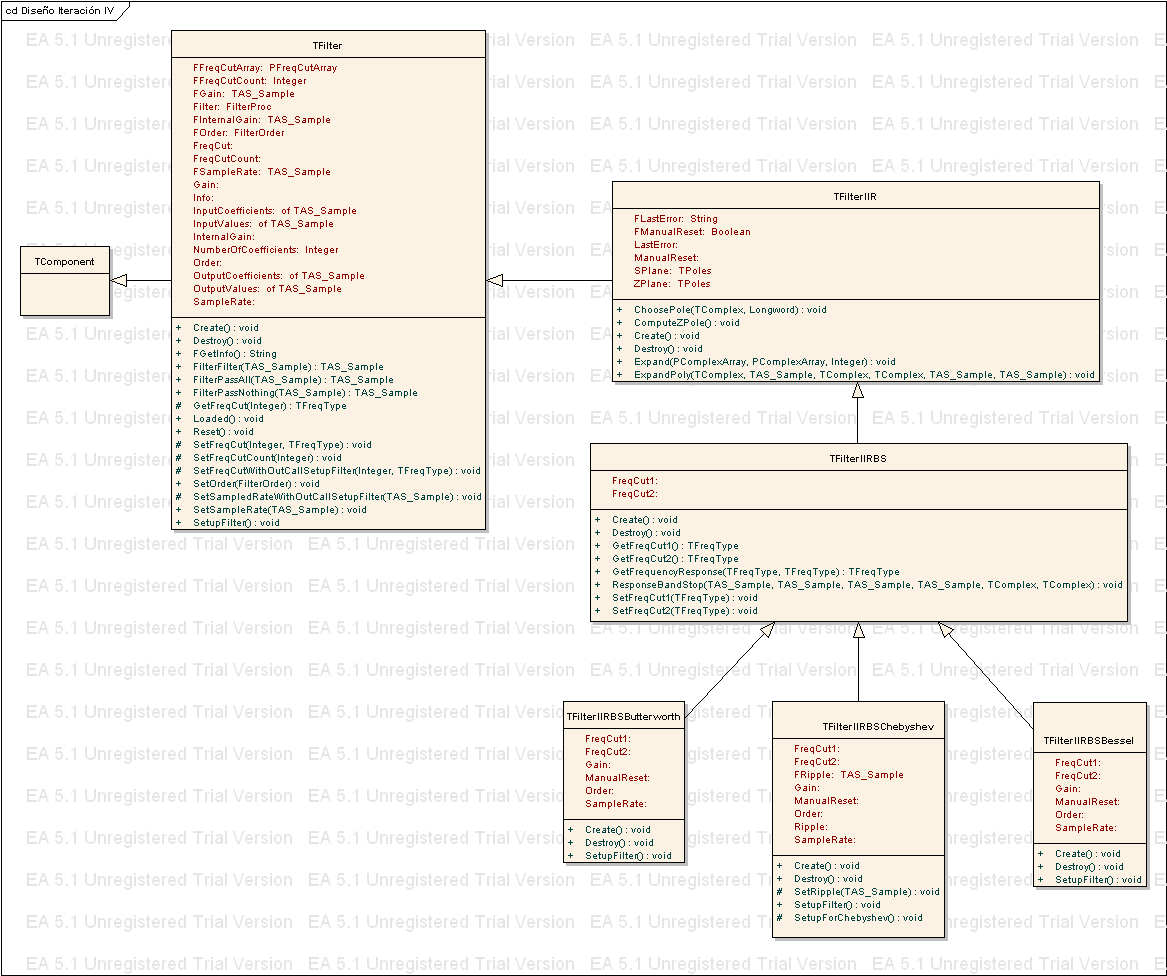


Figura Vista de Diseño Preliminar de la Iteración IV

### 3.5.3 Casos de Prueba

A lo largo de esta iteración se ejecutan los casos de prueba previstos en el epígrafe 2.3.3, que corresponden al desarrollo de los componentes FilterIIRBSButterworth, FilterIIRBSChebyshev y FilterIIRBSBessel.

En la tabla 37 se muestra el caso de prueba, añadido en esta iteración para mostrar la ganancia de los filtros.

Tabla Casos de Prueba Iteración IV

|  |  |  |  |
| --- | --- | --- | --- |
| **#Historia de Usuario** | **#Caso de Prueba** | **Nombre** | **Descripción** |
| **22,24,26** | **1** | Mostrar la ganancia del filtro | El usuario puede observar la ganancia del filtro. |

### 3.5.4 Diseño Final

El diseño final en esta iteración se mantiene igual al diseño preliminar expuesto

en el epígrafe 3.5.2, ya que no hubo ningún cambio en la arquitectura del sistema para la implementación de los componentes FilterIIRBSButterworth, FilterIIRBSChebyshev y FilterIIRBSBessel.

## 3.6 Producción

Las cuatro iteraciones planificadas concluyen el 4 de mayo del 2011 y el software entra en producción el 5 de mayo del 2011. A la paleta de componentes no fue necesario realizarle ninguna modificación, pues el cliente se mostró satisfecho con el trabajo realizado.

**Costo Total del Software**

**C = Cant \* S \* D** (3.3)

**Cant: Cantidad de Programadores = 1**

**S: Salario Diario = $25**

**D: Duración del software = 114 días.**

**C: Costo Total**

**C = 1 \* 25 \* 114**

**C = 2850 $**

## 3.7 Mantenimiento

El mantenimiento es el estado normal de un sistema desarrollado con XP. En esta etapa, simultáneamente, se tienen que implementar nuevas funcionalidades, mantener funcionando el sistema que se encuentra en producción, incorporar nuevos miembros al equipo de desarrollo y despedirse de otros [15].

En este trabajo no se desarrolla la etapa de mantenimiento pues no fue necesario realizar ningún cambio a lo que se hizo. Se deja abierta para otros programadores que deseen continuar desarrollando.

## 3.8 Muerte

En XP que un proyecto culmine exitosamente es tan importante como que así

haya sido su desarrollo. Un proyecto puede finalizar por varias razones. Algunos finalizan por una buena causa y otros por causas no deseadas. Una buena razón para concluir un proyecto es que el cliente se encuentre satisfecho con el trabajo realizado y que no sea capaz de escribir ninguna otra historia de usuario. Si esto ocurriese se debe proceder a redactar un documento acerca del sistema; se aconseja que el documento a redactar tenga las características del tipo de documento que se desearía tener cuando haya que realizar cambios al sistema 5 años más tarde [15].

En este caso el cliente se encuentra satisfecho con el trabajo realizado, lo que no significa que el proyecto llega a su fin (muerte). Este proyecto continuará para que en un futuro se puedan añadir otras funcionalidades.

## 3.9 Conclusiones

En este capítulo se aplica la fase de iteraciones del ciclo de vida de la metodología XP en el desarrollo de los componentes.

En cada una de las cuatro iteraciones previstas anteriormente se define un plan de iteración, y se crea un diseño preliminar, que ayuda a conocer que objetos están involucrados, y cuales son sus relaciones. Este diseño a medida que se avanza en el desarrollo sufre cambios, hasta lograr un diseño definitivo que logre cumplir las demandas del cliente.

Se realizan todas las pruebas definidas anteriormente con el objetivo de comprobar que software funcione correctamente, se añade un nuevo caso de prueba destinado a mostrar la ganancia del filtro.

Una vez terminada la fase de Iteraciones, se entra en la fase de Producción, donde se calcula el costo total al que asciende el software desarrollado siendo este de 2850 pesos. Para estos componentes que se desarrollaron no es objeto de estudio las fases Mantenimiento y Muerte, pero ya que estas son de gran importancia para la realización de sistemas siguiendo una buena práctica de XP, se muestra una breve explicación de ellas.

# Conclusiones

Durante el presente trabajo se desarrollaron 12 componentes para el filtrado digital de señales. Los filtros implementados son de respuesta infinita al impulso (IIR por sus siglas en inglés), de los tipos pasa bajo, pasa alto, pasa banda y supresor de banda, y que se usaron las aproximaciones matemáticas de Butterworth, Chebyshev y Bessel para cada uno de ellos.

Para realizar el diseño e implementación de dichos componentes se realiza un estudio del tema y se determinan las características y funcionalidades necesarias para cada uno de los componentes.

Se realizaron un total de 12 aplicaciones de ejemplos para demostrar el funcionamiento de los componentes implementados.

Los componentes desarrollados podrán ser utilizados por cualquiera que los necesite.

Teniendo en cuenta la necesidad de eficiencia que demanda la comunidad de desarrolladores de hoy en día, los componentes que se desarrollan en este trabajo de diploma, constituyen un serio aporte a todos los programadores que utilizan el IDE Lazarus/Free Pascal, puesto que los mismos contribuyen a mejorar su trabajo de forma significativa.

Los componentes se desarrollaron utilizando el IDE Lazarus/Free Pascal, y la metodología ágil Extreme Programming (XP), estudiados a fondo, para lograr un mejor desempeño en la tarea. A través de este trabajo de diploma se realiza un estudio de otras tecnologías vinculadas con el objeto de estudio y campo de acción del trabajo y se cumplen de forma cabal los objetivos trazados.

Las aplicaciones ejemplos desarrolladas para mostrar el uso y funcionamiento de los componentes implementados, son cómodas, fáciles de entender y los beneficios que reportan los componentes evidencian la factibilidad de su desarrollo.

# Recomendaciones

Se recomienda a un nuevo equipo de desarrollo:

* Crear un conjunto de nuevos componentes para el filtrado digital de señales del tipo de respuesta finita al impulso (FIR).

# Referencias Bibliográficas

1. *El Software Libre y su Importancia para Países en Desarrollo*. [cited 2011 26 de mayo]; Available from: <http://www.construmatica.com/construpedia>.

2. Barchiesi, J.V., *Introducción al Procesamiento digital de Señales*, I. N°173.752, Editor. 2008, Ediciones Universitarias de Valparaíso: Chile.

3. Martín del Campo Becerra and Gustavo Daniel. *Sistema Digital y Sistema Analógico: concepto, ventajas y ejemplos*. [cited 2010 noviembre 22 ];

:[Available from: <http://www.monografias.com/computación/general>.

4. John G. Proakis and Dimitris G. Manolakis, *Tratamiento digital de señales, Principios, algoritmos y aplicaciones*. tercera edición ed. 1998: Prentice Hall.

5. Sira Parazuelos and Manuel Ureña, *CONVERSORES ANALÓGICOS-DIGITALES Y DIGITALES-ANALÓGICOS.* Universidad de Alcalá, 2009(Tecnología de Computadores,

Ingeniería en Informática,).

6. Alan V. Oppenheim and Alan S. Willisky, *Señales y Sistemas*. segunda edición ed: Prentice Hall.

7. Emilia Gómez Gutiérres, *Introducción al filtrado digital*. 2009, Departamento de Sonología, Escola Superior de Musica de Catalunya.

8. *Filtros digitales.*

9. Mary Carmen Martínez Barrera, Dr. Oscar G. Ibarra Manzano, and Ing. Mario A. Ibarra Manzano, *DISEÑO DE FILTROS DIGITALES FIR MEDIANTE LA TECNICA DE VENTANAS.* Facultad de Ingeniería Mecánica Eléctrica y Electrónica

Universidad de Guanajuato.

10. Juan Carlos Ramírez Nuño, *Clasificación de los Filtros por su Respuesta*. 2003, Universidad de la Salle Bajío.

11. Beck K, *Extreme Programming Explained. Embrace Change.* Addison-Wesley, 1999.

12. *Borland*. [cited; Available from: <http://www.borland.com>.

13. Pascal. *Comparativa de Free Pascal Frente a otros Lenguajes*. 2009 [cited 2010 mayo 27]; Available from: <http://www.freepascal.com>.

14. Cohn M, *User Stories Applied: For Agile Software Development.* Addison-Wesley, 2004.

15. Danay Pérez Ramírez, *INVESTIGACIÓN DE LA METODOLOGÍA ÁGIL EXTREME PROGRAMMING Y SU APLICACIÓN EN UN CASO DE ESTUDIO*. 2008, ISPJAE.

16. Pressman, R.S., *Ingeniería del Software, un enfoque práctico*. Sexta Edición ed.

17. Manuel J. *Wikilearning*. [cited 2010 julio 2]; Available from: <http://www.lawebdejm.com>.

18. Manuel J. *Creación de Componentes VCL*. [cited 2010 julio 2]; Available from: <http://www.lawebdejm.com>.

19. RSL. *Reflecciones sobre el uso de Software Libre en Cuba*. [cited 2010 julio 2]; Available from: <http://www.monografias.com/trabajos40/software-cuba>.

20. Sparxsystems. [cited 2010 abril 20]; Available from: <http://www.sparxsystems.com.ar/products/ea.html>.

21. Emagister. *Modelado de Negocios Con BPMN y UML en Distrito Federal*. [cited 2010 mayo 26]; Available from: <http://www.emagister.com.mx/curso_modelado_negocios_con_bpmn_y_uml_distrito_federal-cinst-37479.htm>.

22. Sparxsystems. *Eriksson-Penker Extensions*. 2010 [cited 2010 mayo 28]; Available from: <http://www.sparxsystems.com/uml_tool_guide/extending_models/eriksson_penker_extensions.htm>.

23. Reynoso C, *Métodos Heterodoxos en Desarrollo de Software.* 2004.

24. Patricio Letelier and Maria del Carmen Penadés, *Metodologías ágiles para el desarrollo de software:Extreme Programming (XP).*

25. Beck K and Martin Fowler, *Planning Extreme Programing. First Edition.* 2000.

# Bibliografía

1. Letelier, P., J. Canós, and E. Sánchez, *An Experiment Working with RUP and XP.*Valencia, 2003.
2. Sánchez, M.A.M., *Metodologías De Desarrollo De Software.*Perú, 2004.
3. Rawsthorne, D., *Comparing/Combining RUP, XP, and Scrum – mixing the Process Cocktail.*2003.
4. Sánchez, P.A.E. and J.H.C. Letelier, *Mejorando la gestión de historias de*

*usuario en eXtreme Programming.* Valencia, 2004.

1. Sutherland, J., *Baufest Scrum Conference 2006.* 2005.
2. Shore, J., *Phased Releases.* 2004.
3. Quirino and E. Rámirez, *Programación Extrema(XP)*
4. Leone and N. Passerini, *Metodologías Iterativas de Desarrollo.*Buenos Aires, 2005.
5. Fowler, M., *After the Program After the Program Runs.* 1998.
6. Y Kuranuki and K. Hiranabe, *AntiPractices: AntiPatterns for XP Practices.*
7. Canneyt, M.V. and F. Klämpﬂ, *Free Pascal :User’s Guide.* 2009.
8. Canneyt, M.V., *Free Pascal Programmers’ manual.*2007.
9. Danay Perez Ramirez.”Investigación de la Metodología Ágil Extreme Programming y su Aplicación en un Caso de Estudio”
10. Kent Beck, Martin Fowler, and Robert Martin. “Programming Extreme Programming”. 2000.
11. John G. Proakis and Dimitris G. Manolakis, Tratamiento digital de señales, Principios, algoritmos y aplicaciones. tercera edición ed. 1998: Prentice Hall.
12. Alan V. Oppenheim and Alan S. Willisky, Señales y Sistemas. segunda edición ed: Prentice Hall.

17. Pressman, R.S., *Ingeniería del Software, un enfoque práctico*. Sexta Edición ed.

# Glosario de Términos

**Agricultura de precisión:** Es el uso de las Tecnologías de la Información y las Comunicaciones (TIC) para el manejo del sitio agrícola, como manera de alcanzar una mayor eficiencia productiva y una disminución del impacto ambiental respecto de los llamados manejos uniformes que no consideran la variabilidad espacial ni temporal en los cultivos.

**Aliasing:** El aliasing es el efecto que causa que las señales continuas distintas se tornen indistinguibles cuando se les muestra digitalmente. Cuando esto sucede, la señal original no puede ser reconstruida de forma unívoca a partir de la señal digital.

**Antialiasing:** En el área del procesamiento digital de señales se le llama antialiasing a los procesos que permiten minimizar el aliasing cuando se desea representar una señal de alta resolución en un sustrato de más baja resolución. Consiste en la eliminación de la información de frecuencia demasiado elevada para poder ser representada.

**Borland Delphi:** Entorno de desarrollo de software diseñado para la programación de propósito general con énfasis en la programación visual. En Delphi se utiliza como lenguaje de programación una versión moderna de Pascal llamada Object Pascal. Es producido comercialmente por la empresa estadounidense CodeGear, adquirida en mayo de 2008 por Embarcadero Technologies. En sus diferentes variantes, permite producir archivos ejecutables para Windows, GNU/Linux y la plataforma .Net.

**BorlandDeveloper Studio 2006 (BDS):** Lanzado en diciembre de 2006. Es un conjunto de lenguajes de programación con un ambiente elegante, un buen editor de código, diseñador visual y depurador. Incluye soporte completo de RAD para C++ y C#, además de Delphi para Win32 y Delphi para.NET lenguajes de programación.

**Eriksson-Penker:** Extensión desarrollada por H. E. Eriksson y M. Penker. Suministra una base para las extensiones de modelo de procesamiento UML, provee unos medios únicos y fuertes de visualizar y comunicar procesos de la empresa y la circulación necesaria de la información dentro de una organización.

**Fase:** Cada uno de los distintos estados sucesivos por los que puede pasar un software en su ciclo de vida.

**Feedback:** Proviene del inglés y podría ser traducido correctamente al castellano como ‘retroalimentación’, aunque de todos modos suele usarse en idioma inglés en la mayoría de los países de habla hispana. El feedback o retroalimentación es el proceso mediante el cual se realiza un [intercambio](http://www.definicionabc.com/general/intercambio.php)  de datos, informaciones, hipótesis o teorías entre dos puntas diferentes. Este término puede, así, aplicarse tanto a situaciones sociales como también a situaciones científicas, tanto biológicas como [tecnológicas](file:///F:\feedback.php.htm).

**Free Pascal:** Compilador compatible con Turbo Pascal para todas las plataformas más famosas (incluido Linux, OS X y Windows). Desde la versión 2.0, Free Pascal es además compatible con Delphi.

**IDE:** Un entorno de desarrollo integrado o en inglés Integrated Development Environment ('IDE') es un programa compuesto por un conjunto de herramientas para un programador.

**Iteración:** Período de 1 a 4 semanas en el que se implementa una o varias historias de usuarios.

**Lazarus:** Lazarus es una biblioteca de código abierto de componentes visuales y un IDE fuerte para el desarrollo de plataformas. El IDE contiene todas las características de una suite de desarrollo moderna, incluyendo un depurador, completamiento de código, diseñadores visuales, refactorización de herramientas, herramientas de traducción y documentación.

**Metáfora:** Acuerdo entre desarrolladores y clientes sobre cómo funcionará el futuro sistema que guía el desarrollo del sistema y reemplaza una gran parte de la arquitectura en etapas tempranas del desarrollo.

**Resonancia:** La resonancia es un estado de operación en el que una frecuencia de excitación se encuentra cerca de una frecuencia natural de la estructura de la máquina. Cuando ocurre la resonancia, los niveles de vibración que resultan pueden ser muy altos y pueden causar daños muy rápidamente.

**UML:** Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software.

**XP:** Extreme Programming o Programación Extrema. Metodología ágil para el desarrollo de software, destinada a ser utilizada por equipos de desarrollo, pequeños y medianos (de 2 a 10 miembros) que se enfrenten a proyectos con requerimientos imprecisos o cambiantes.

# Anexo 1 Tablas de Historias de Usuario

**Campos de tabla Historia de Usuario** [15]

**Número:** Índice de la historia de usuario. Es un número único que se le asigna a cada historia de usuario a fin de lograr una mejor organización de éstas.

**Nombre historia:** Nombre de la historia de usuario. Debe ser descriptivo, en la medida de las posibilidades, de lo que se implementará y no muy extenso.

**Usuario:** Rol que algo o alguien juega cuando interactúa con el sistema. Puede ser cualquier individuo, grupo, organización o máquina que interactúa con el sistema.

**Programador responsable:** Programador responsable de implementar la historia de usuario.

**Descripción:** Se describe el requerimiento que da origen a la historia de usuario. La descripción debe ser corta, precisa y dejar claro qué es lo que se desea hacer.

**Observaciones:** Aspectos que se deben tener en cuenta para implementar la historia de usuario. Precondiciones o pos condiciones que se deben tener presentes al implementar la historia de usuario.

**Puntos estimados:** Tiempo estimado que demorará la implementación de la historia de usuario. Tiempo ideal en que se estima se implementará la historia de usuario.

**Prioridad en negocio:** Prioridad que se le asigna a la historia de usuario en el negocio. Si es importante y debe ser implementada lo antes posible, entonces la prioridad debe ser alta, si no media o baja.

**Riesgo en desarrollo:** Riesgo que representa para el desarrollo la historia de usuario. En dependencia del riesgo que represente se le asigna tres valores, alto, medio y bajo.

* **Alto:** historias de usuario que no pueden ser estimadas con suficiente precisión o que implementarlas en iteraciones posteriores puede representar cambios sustanciales en el código base del proyecto.
* **Medio:** historias de usuario que son estimadas razonablemente bien.
* **Bajo:** historias de usuario que son estimadas sin problema alguno.

**Iteración asignada:** Iteración en la que se implementará la historia de usuario.

**Algunas tarjetas de Historias de Usuario**

Tabla Historia de Usuario 1

|  |  |
| --- | --- |
| **Historia de Usuario** | |
| **Número:** 1 | **Usuario:** Usuario Final |
| **Nombre de Historia:** Especificar el orden del filtro. | |
| **Prioridad en Negocio:** Alta | **Riesgo en Desarrollo:** Alto |
| **Puntos Estimados:** 1.5 | **Iteración Asignada:** 1 |
| **Programador Responsable:** Anailec Venero Perdomo | |
| **Descripción:** El usuario puede especificar el valor del orden del filtro que desea. | |
| **Observaciones:** | |

Tabla Historia de Usuario 2

|  |  |
| --- | --- |
| **Historia de Usuario** | |
| **Número:** 2 | **Usuario:** Usuario Final |
| **Nombre de Historia:** Especificar la frecuencia de corte del filtro | |
| **Prioridad en Negocio:** Alta | **Riesgo en Desarrollo:** Alto |
| **Puntos Estimados:** 2.5 | **Iteración Asignada:** 1 |
| **Programador Responsable:** Anailec Venero Perdomo | |
| **Descripción:** El usuario puede especificar el valor de la frecuencia de corte que desea. | |
| **Observaciones:** | |

Tabla Historia de Usuario 3

|  |  |
| --- | --- |
| **Historia de Usuario** | |
| **Número:** 3 | **Usuario:** Programador |
| **Nombre de Historia:** Diseñar el filtro FilterIIRLPButterworth | |
| **Prioridad en Negocio:** Alta | **Riesgo en Desarrollo:** Alto |
| **Puntos Estimados:** 1.4 | **Iteración Asignada:** 1 |
| **Programador Responsable:** Anailec Venero Perdomo | |
| **Descripción:** Se diseña el filtro para que responda según las características de un filtro pasa bajo Butterworth. | |
| **Observaciones:** | |

Tabla Historia de Usuario 4

|  |  |
| --- | --- |
| **Historia de Usuario** | |
| **Número:** 4 | **Usuario:** Sistema |
| **Nombre de Historia:** Filtrar señal a través del filtro FilterIIRLPButterworth | |
| **Prioridad en Negocio:** Alta | **Riesgo en Desarrollo:** Alto |
| **Puntos Estimados:** 5 | **Iteración Asignada:** 1 |
| **Programador Responsable:** Anailec Venero Perdomo | |
| **Descripción:** Se filtra una señal a través del filtro FilterIIRLPButterworth. | |
| **Observaciones:** | |

# Anexo 2 Tablas de Casos de Prueba

**Campos de la tabla Caso de Prueba**  [15]

**Fecha:** Fecha en que se realiza el caso de prueba. En caso de realizarse varias veces siempre se actualiza con la última.

**Tipo de prueba:** Informa si la prueba que se realiza es funcional o unitaria.

**Condiciones de ejecución:** Condiciones especiales que deben tenerse en cuenta para ejecutar el caso de prueba.

**Entradas:** Entradas al caso de prueba en caso de necesitarlas.

**Resultado esperado:** Resultado que se desea tenga el caso de prueba. Descripción breve de lo que debe suceder.

**Evaluación:** Se evalúa si el caso de prueba tuvo éxito o no. En caso de ser exitoso se asigna un resultado de satisfactorio, en caso contrario insatisfactorio.

**Algunas tarjetas de Casos de Prueba**

Tabla Caso de Prueba 1

|  |  |
| --- | --- |
| **Caso de Prueba** | |
| **Número: 1** | **Número Historia de Usuario: (\*)** |
| **Nombre:** Especificar el orden del filtro. | **Fecha: 2/05/11** |
| **Tipo de Prueba:** Unitaria | |
| **Descripción:** El usuario especifica el orden del filtro desde la interfaz visual. | |
| **Condiciones de Ejecución:** | |
| **Entradas:** | |
| **Resultado Esperado:** El filtro responde según el valor del orden especificado. | |
| **Evaluación:** Satisfactoria | |

Nota: (\*) se le aplica a todas las historias de usuario.

Tabla Caso de Prueba 2

|  |  |
| --- | --- |
| **Caso de Prueba** | |
| **Número:** 2 | **Número Historia de Usuario:** **4,6,8,10,12,14** |
| **Nombre:** Especificar una frecuencia de corte. | **Fecha: 2/04/11** |
| **Tipo de Prueba:** Unitaria | |
| **Descripción:** El usuario inserta la frecuencia de corte del filtro desde la interfaz visual, o directamente en la propiedad FreqCut1 para los filtros pasa bajo y pasa alto. | |
| **Condiciones de Ejecución:** | |
| **Entradas:** | |
| **Resultado Esperado:** El filtro responde según el valor de la frecuencia de corte especificada. | |
| **Evaluación:** Satisfactoria | |

Tabla Caso de Prueba 3

|  |  |
| --- | --- |
| **Caso de Prueba** | |
| **Número:** 3 | **Número Historia de Usuario:** **16,18,20,22,24,26** |
| **Nombre:** Especificar dos frecuencias de corte | **Fecha: 2/05/11** |
| **Tipo de Prueba:** Unitaria | |
| **Descripción:** El usuario inserta la frecuencia de corte del filtro desde la interfaz visual, o directamente en las propiedades FreqCut1 y FreqCut2 para los filtros pasa banda y supresor de banda. | |
| **Condiciones de Ejecución:** | |
| **Entradas:** | |
| **Resultado Esperado:** El filtro responde según el valor de las frecuencias de corte especificadas. | |
| **Evaluación:** Satisfactoria | |

# Anexo 3 Vistas de algunos ejemplos

**Ejemplo 1**

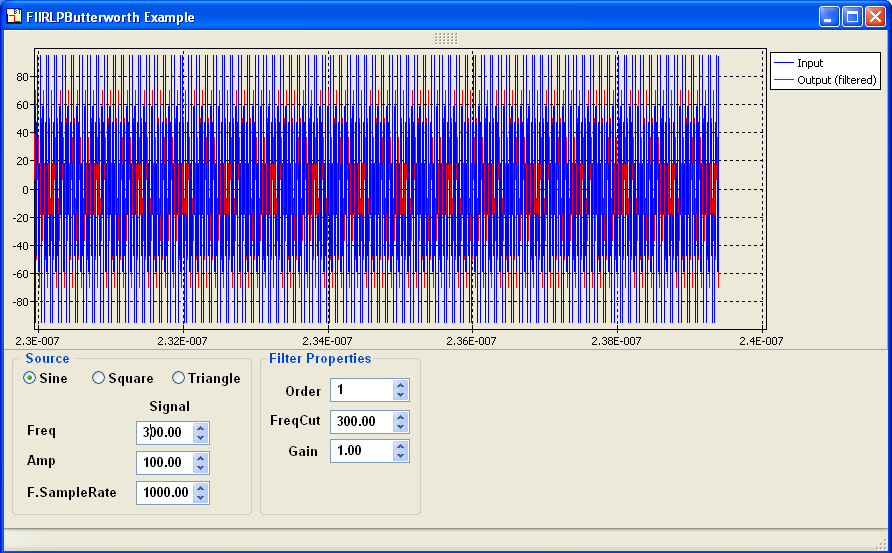
La figura 14 muestra una señal de entrada sinusoidal con frecuencia de 300Hz y amplitud de 100, y su correspondiente señal de salida después de aplicar un filtro pasa bajo Butterworth con frecuencia de corte de 300Hz, se puede observar que su amplitud se reduce aproximadamente al 70% de su valor máximo, es decir, sufre una atenuación de 3 dB.

Figura Vista 1 del uso de un filtro pasa bajo Butterworth

Si esta misma señal de entrada se muestrea a una frecuencia de 200Hz y se aplica el filtro pasa bajo de Butterworth con frecuencia de corte de 300Hz, entonces como se trata de un filtro pasa bajo, pasarían las componentes de la señal que están por debajo de la frecuencia de corte, como se muestra en la figura 15.

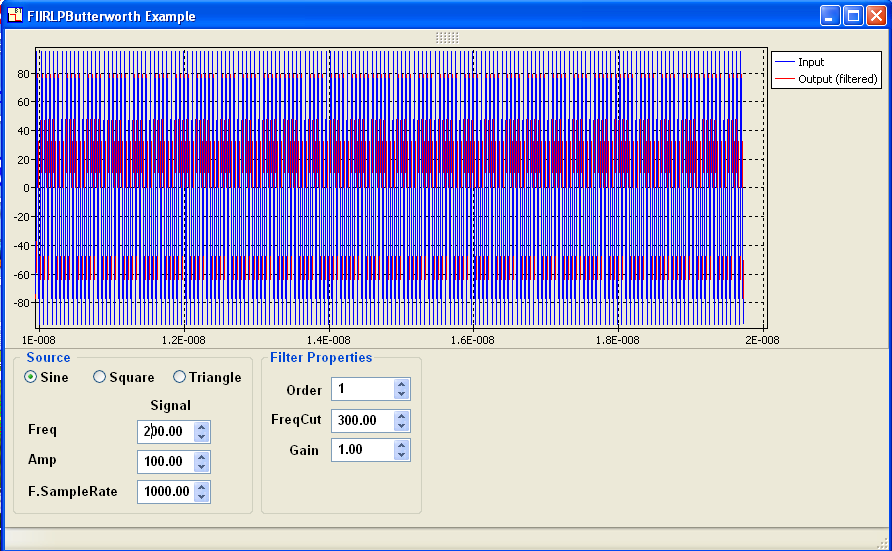


Figura Vista 2 del uso de un filtro pasa bajo Butterworth

Si esta señal de entrada se muestrea a una frecuencia de 500Hz y se aplica el filtro pasa bajo de Butterworth con frecuencia de corte de 300Hz, entonces como se trata de un filtro pasa bajo, no pasaría ninguna componente de la señal, como se muestra en la figura 16.

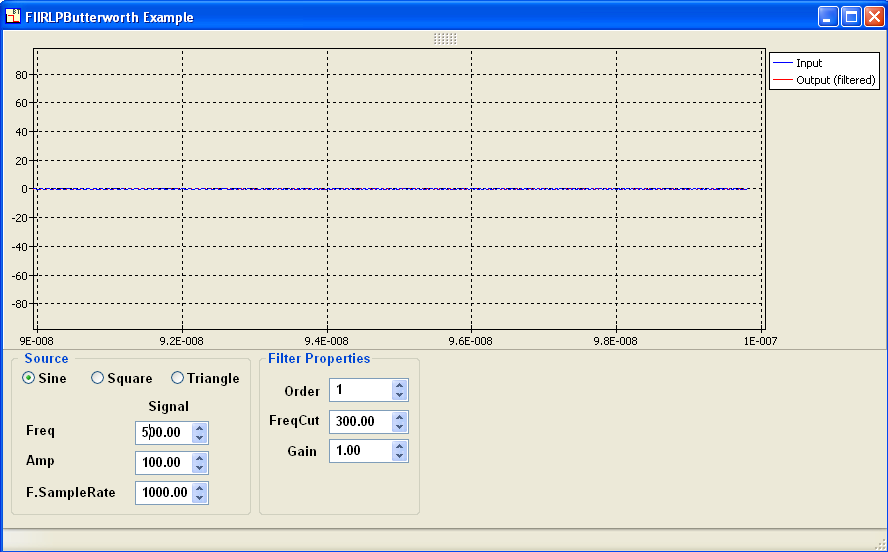


Figura Vista 3 del uso de un filtro pasa bajo Butterworth

**Ejemplo 2**

La figura 17 muestra una señal de entrada sinusoidal con frecuencia de 300Hz y amplitud de 100, y su correspondiente señal de salida después aplicar un filtro pasa alto Chebyshev con frecuencia de corte de 300Hz, se puede observar que su amplitud se reduce aproximadamente al 70% de su valor máximo, es decir, sufre una atenuación de 3 dB.

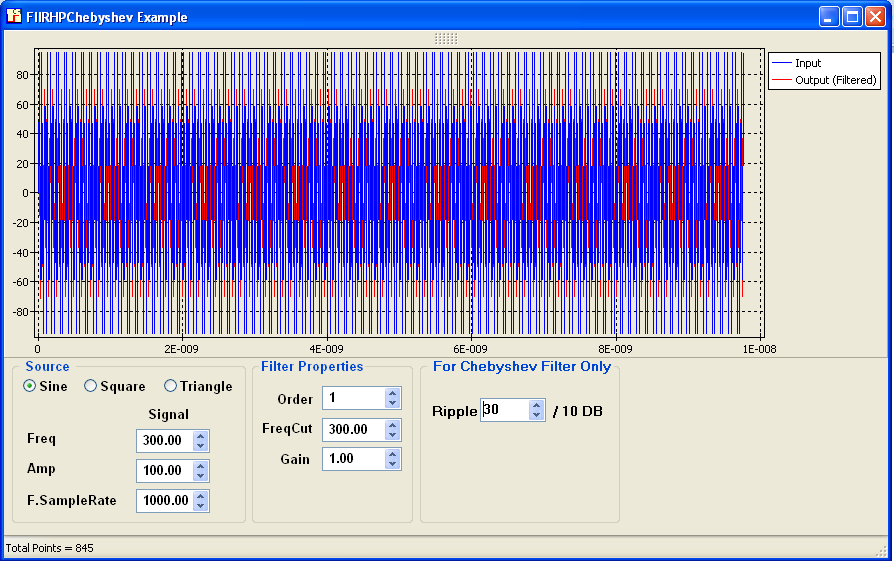
****

Figura Vista 1 del uso de un filtro pasa alto Chebyshev

Si esta misma señal de entrada se muestrea a una frecuencia de 600Hz y se aplica el filtro pasa bajo de Chebyshev con frecuencia de corte de 300Hz, entonces como se trata de un filtro pasa alto, pasarían las componentes de la señal que están por encima de la frecuencia de corte, como se muestra en la figura 18.

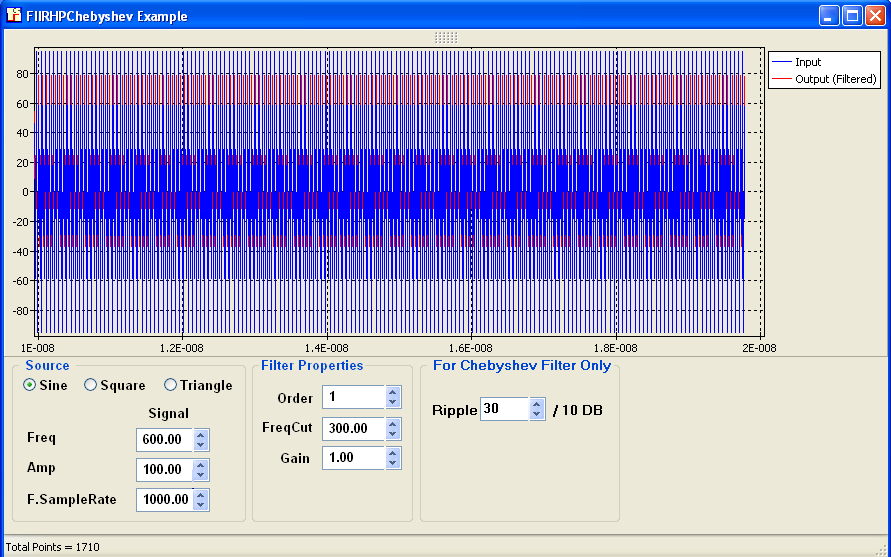


Figura Vista 2 del uso de un filtro pasa alto Chebyshev

Si esta señal de entrada se muestrea a una frecuencia de 5Hz y se aplica el filtro pasa alto de Chebyshev con frecuencia de corte de 300Hz, entonces como se trata de un filtro pasa alto, no pasaría ninguna componente de la señal, como se muestra en la figura 19.

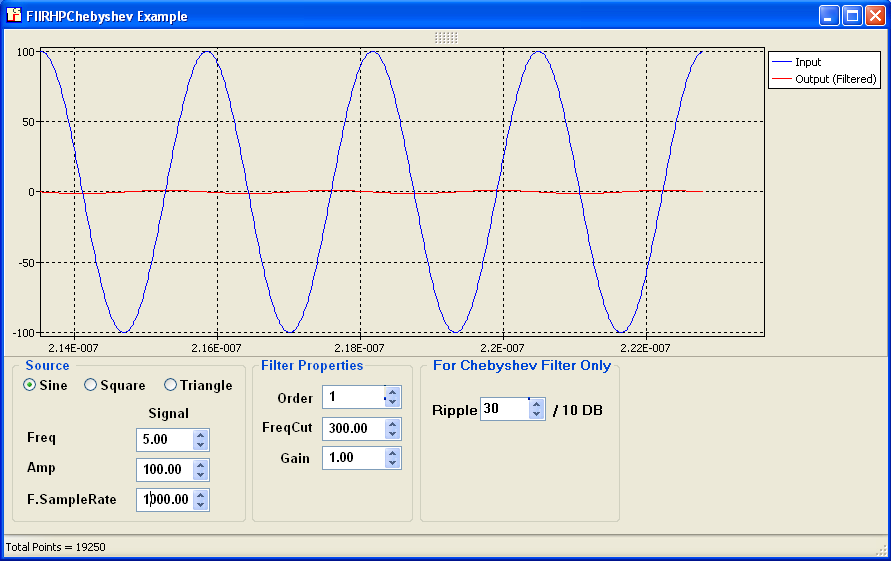


Figura Vista 2 del uso de un filtro pasa alto Chebyshev

**Ejemplo 3**

La figura 20 muestra una señal de entrada sinusoidal con frecuencia de 300Hz y amplitud de 100, después aplicar un filtro pasa banda Bessel con frecuencia de corte uno de 200Hz y frecuencia de corte dos de 300Hz, se puede observar que su amplitud se reduce aproximadamente al 70% de su valor máximo, es decir, sufre una atenuación de 3 dB.

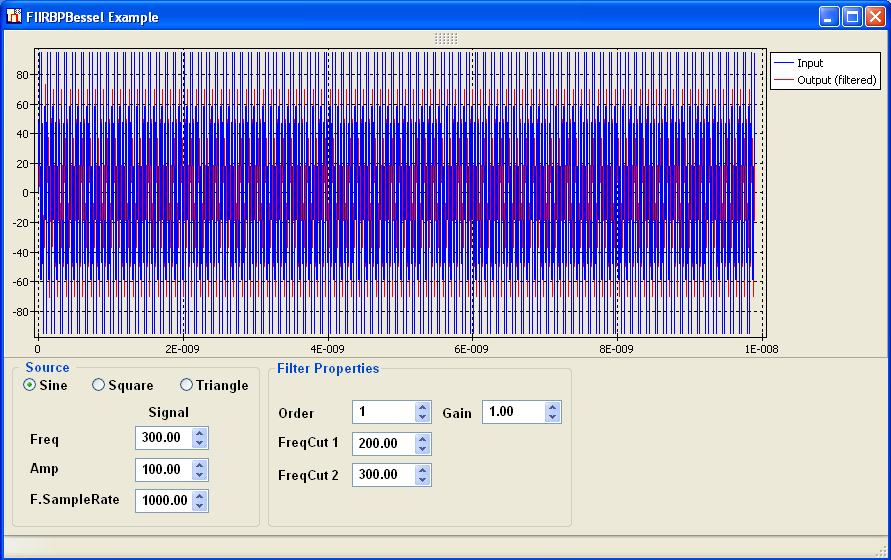


Figura Vista 1 del uso de un filtro pasa banda Bessel

Si esta misma señal de entrada se muestrea a una frecuencia de 210Hz y se aplica el filtro pasa banda Bessel con frecuencia de corte uno de 200Hz y frecuencia de corte dos de 300Hz, entonces como se trata de un filtro pasa banda, pasarían las componentes de la señal que están entre la frecuencia de corte uno y la frecuencia de corte dos, como se muestra en la figura 21.

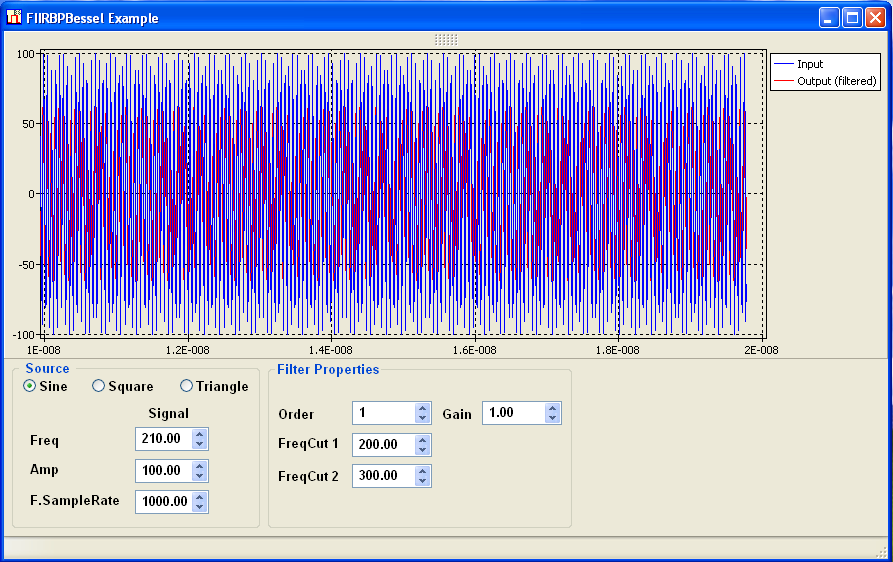


Figura Vista 2 del uso de un filtro pasa banda Bessel

Si esta misma señal de entrada se muestrea a una frecuencia de 5Hz y se aplica el filtro pasa banda Bessel con frecuencia de corte uno de 200Hz y frecuencia de corte dos de 300Hz, entonces como se trata de un filtro pasa banda, no pasaría ninguna componente de la señal, como se muestra en la figura 22.

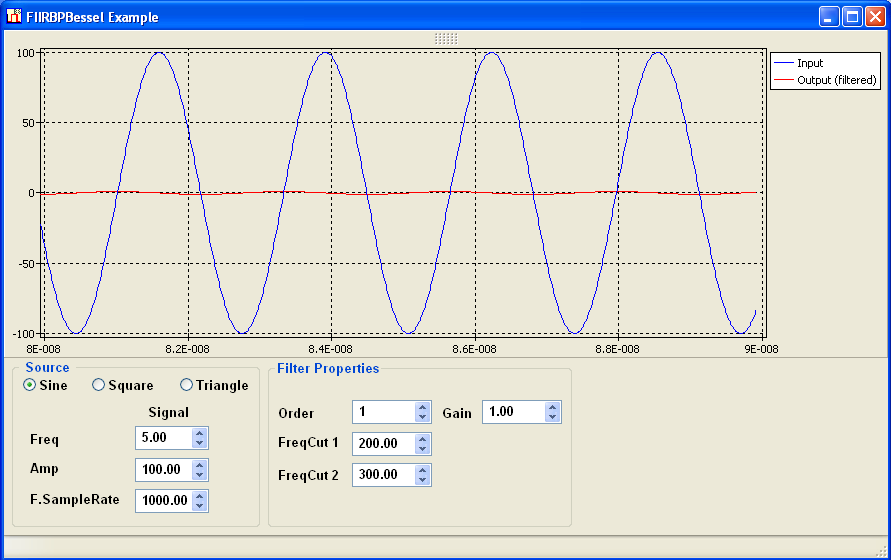


Figura Vista 3 del uso de un filtro pasa banda Bessel

De igual forma si la señal de entrada se muestrea a una frecuencia de 500Hz y se aplica el filtro pasa banda Bessel con frecuencia de corte uno de 200Hz y frecuencia de corte dos de 300Hz, entonces como se trata de un filtro pasa banda, no pasaría ninguna componente de la señal, como se muestra en la figura 23

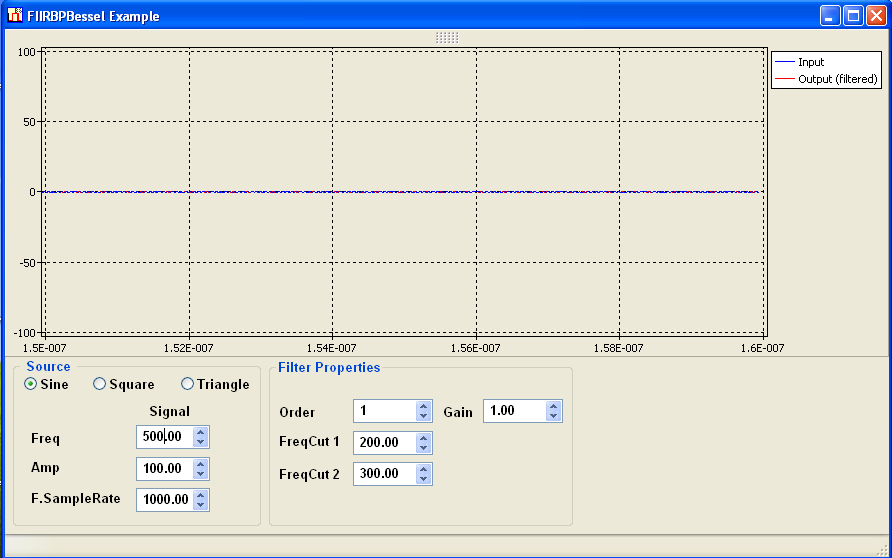


Figura Vista 4 del uso de un filtro pasa banda Bessel

**Ejemplo 4**

La figura 24 muestra una señal de entrada sinusoidal con frecuencia de 300Hz y amplitud de 100, y su correspondiente señal de salida después aplicar un filtro supresor de banda Butterworth con frecuencia de corte uno de 200Hz y frecuencia de corte dos de 300Hz, se puede observar que su amplitud se reduce aproximadamente a 70% de su valor máximo, es decir, sufre una atenuación de 3 dB.

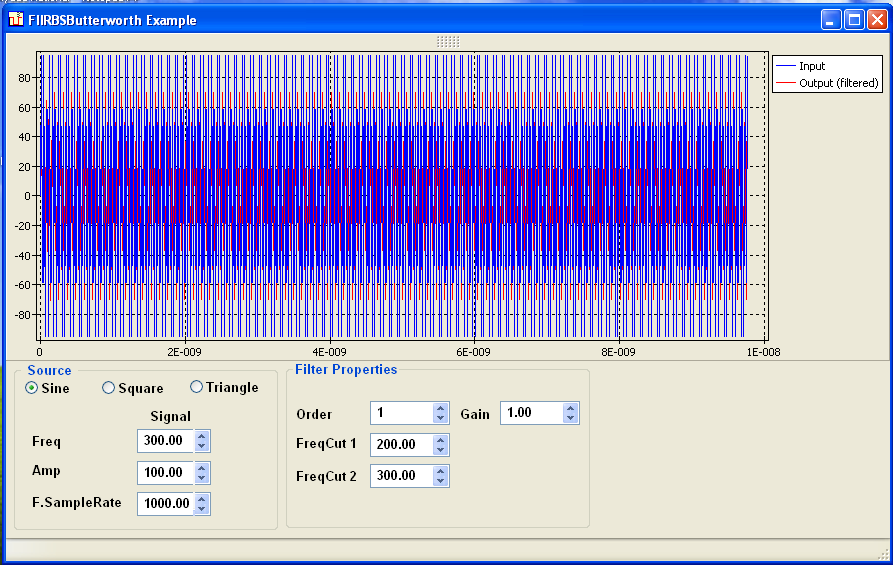


Figura Vista 1 del uso de un filtro supresor de banda Butterworth

Si esta misma señal de entrada se muestrea a una frecuencia de 100Hz y se aplica el filtro supresor de banda Butterworth con frecuencia de corte uno de 200Hz y frecuencia de corte dos de 300Hz, entonces como se trata de un filtro supresor de banda, pasarían las componentes de la señal que están por debajo de la frecuencia de corte uno, como se muestra en la figura 25.

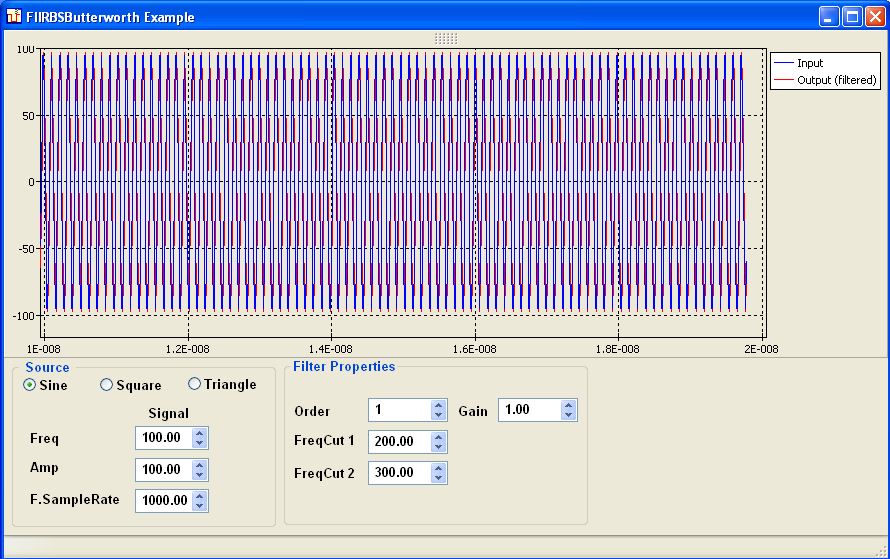


Figura Vista 2 del uso de un filtro supresor de banda Butterworth

De igual forma si esta misma señal de entrada se muestrea a una frecuencia de 700Hz y se aplica el filtro supresor de banda Butterworth con frecuencia de corte uno de 200Hz y frecuencia de corte dos de 300Hz, entonces como se trata de un filtro supresor de banda, pasarían las componentes de la señal que están por encima de la frecuencia de corte dos, como se muestra en la figura 26.

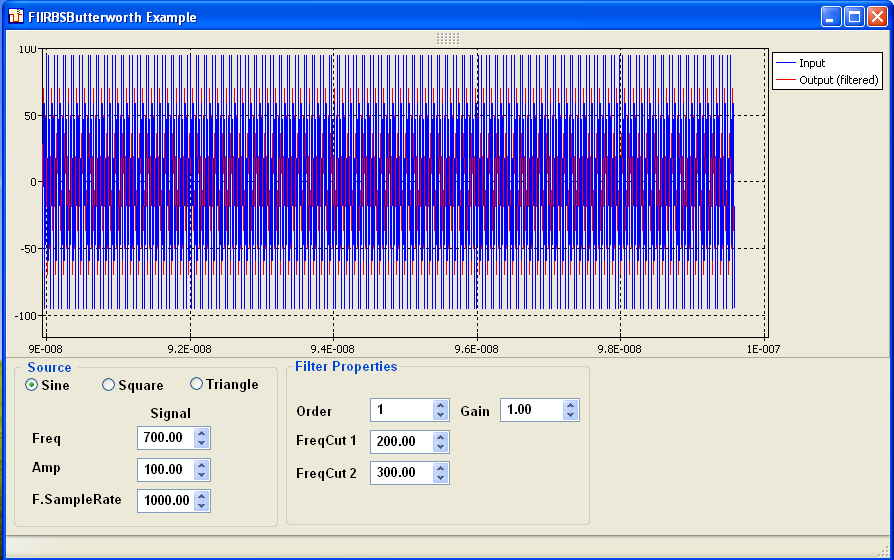


Figura Vista 3 del uso de un filtro supresor de banda Butterworth

Si esta misma señal de entrada se muestrea a una frecuencia de 250Hz y se aplica el filtro supresor de banda Butterworth con frecuencia de corte uno de 200Hz y frecuencia de corte dos de 300Hz, entonces como se trata de un filtro supresor de banda, no pasaría ninguna componente de la señal, como se muestra en la figura 27.

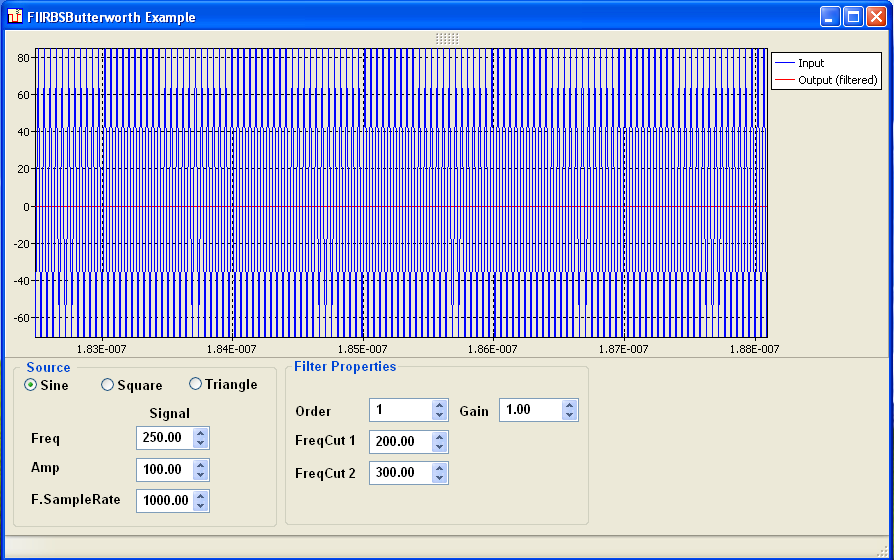


Figura Vista 4 del uso de un filtro supresor de banda Butterworth