UNIVERSIDAD DE HOLGUÍN
Facultad de Informática y Matemática

PYSIGREC: SOFTWARE PARA EL REGISTRO DE SEÑALES ELECTRO-OCULOGRÁFICAS

Trabajo de Diploma para optar por el Título de Ingeniero Informático

Autor

Andrés Henriquez Pérez

Tutores

Ing. Roberto Antonio Becerra García
Dr. C. Rodolfo Valentín García Bermúdez

Holguín 2012

Agradecimientos

A mi chuchu por estar ahí siempre.

A mi mamá Marlenis, mi papá Luis y mi hermano por haberme guiado en todos los caminos que he tomado.

A mi familia completa por ser tan unida.

A mis compañeros de aula y en especial al club del café.

A los amigos, los de los buenos y malos tiempos.

A Libys Martha, por enseñarme un par de cositas.

A Roberto Becerra por enseñarme todo lo que pudo.

Resumen

En esta tesis se aborda el desarrollo del PySigRec, software para capturar y registrar señales electro-oculográficas, desarrollado en Python, que realiza la captura a través de tarjetas de adquisición de datos utilizando la biblioteca de clases y drivers Comedi. Además exporta los datos en ficheros externos con formatos accesibles para que otras herramientas los puedan procesar. Permite la adaptación al dispositivo, así como la detección de sus capacidades.

Abstract

This work intended to show the development process of the software “PySigRec”, made with the objective of capturing and exporting electro-oculography signals from an acquisition device. It was develop in Python programming language, and it uses the Comedi library to accomplish the acquisition task. It also exports the data into files that can be read by other applications for further processing. Also allows to detect device capabilities and the use of them by the software itself.

Índice de contenido

Índice de tablas

Índice de figuras

Introducción

La computación es una de las ramas de la ciencia con mayor velocidad de desarrollo, y cada vez se introduce más en todas las esferas de la vida del hombre, desde los procesos científicos, tecnológicos e industriales hasta las tareas domésticas y la medicina. El aumento de los datos generados en la medicina influenciado por la utilización de tecnologías modernas y los equipos electrónicos, ha conllevado a la utilización de la informática como ciencia para dar tratamiento a estos datos y obtener resultados rápidos y veraces.

Así surgió casi desde el mismo comienzo de la computación una nueva ciencia llamada bio-informática, según una de sus definiciones más sencillas, es la aplicación de tecnología de computadores a la gestión y análisis de datos biológicos. Esta ciencia es una de las de mayor crecimiento en el mundo actual, y Cuba no se queda atrás.

El trabajo médico trae consigo grandes volúmenes de información, que necesitan a su vez ser procesados para obtener resultados clínicos. Este proceso se ha vuelto difícil de realizar de manera tradicional y esto ha provocado la inmersión y utilización de las tecnologías de la informatización. El uso de herramientas informáticas ha humanizado el trabajo de los especialistas de las disímiles materias de la medicina, además ha aportado resultados relevantes y su papel en investigaciones de este tipo es cada vez mayor.

Cuba posee un gran adelanto en este tema e incluso se han creado centros de investigaciones como el Centro de Neurociencias de La Habana y el Centro de Investigación y Rehabilitación de las Ataxias Hereditarias (CIRAH) en Holguín que realizan un uso intensivo de las aplicaciones informáticas enfocadas en la medicina. La medicina en Cuba es un tema muy importante y es además una causa noble para investigar. Estos centros investigan principalmente el cerebro y el corazón, y utilizan las señales emitidas por estos órganos para determinar anomalías, dolencias y tratamientos.

Existen muchas enfermedades y trastornos principalmente cerebrales que afectan la visión. Los movimientos oculares tienen un rol muy útil en la identificación de las disfunciones en un amplio rango de condiciones neurológicas, y entre estos los movimientos sacádicos, utilizados para cambiar bruscamente el campo visual, proveen de una útil herramienta en la exploración de las funciones neurales.

Existen varios métodos de obtener las señales emitidas por los ojos, uno de ellos se realiza usando electrodos que registran las variaciones de micro volts emitidas por estos órganos cuando se mueven.

En la Universidad de Holguín Oscar Lucero Moya (UHOLM) radica un grupo de investigación integrado en su mayoría por estudiantes que aborda los temas relacionados con el análisis de señales, para desarrollar productos informáticos con fines médicos.

Para analizar las señales es necesario capturarlas y almacenarlas en formato digital, este proceso requiere un dispositivo de adquisición de datos, que obviamente los convierta de analógicos a digital. La **adquisición de**

datos o adquisición de señales, consiste en la toma de muestras del mundo real (sistema analógico) para generar datos que puedan ser manipulados por un ordenador u otras electrónicas (sistema digital). Consiste, en tomar un conjunto de señales físicas, convertirlas en tensiones eléctricas y digitalizarlas de manera que se puedan procesar en una computadora. Se requiere una etapa de acondicionamiento, que adecua la señal a niveles compatibles con el elemento que hace la transformación a señal digital. El elemento que hace dicha transformación es el módulo de digitalización o tarjeta de Adquisición de Datos (**DAQ** por sus siglas en inglés).

Los drivers de estos dispositivos son usualmente escritos para programadores de aplicaciones, que tienen una sola aplicación en mente. En el mundo hay muchas aplicaciones desarrolladas para dispositivos de este tipo muy específicos y esto provoca que las funciones sean especializadas y no genéricas, lo cual resulta en un proceso muy difícil para adaptar o reutilizar.

Para estudiar las señales se hace necesario un software que funcione como interfaz de comunicación entre los usuarios y el dispositivo teniendo como objetivo que funcione correctamente y brinde las mismas funcionalidades utilizando la mayor cantidad de tarjetas posibles y que también permita incluirle algunas funcionalidades adicionales como guardar en formatos específicos que puedan ser interpretados por otros softwares como el EogStudio[1].

Almacenar las señales permite trabajar con posterioridad, aplicarles filtros de eliminación de ruido o realizarles procedimientos de detección de sácadas[2] como el que aplica el EogStudio. Además permite llevar un estudio detallado de un paciente, para monitorear el desarrollo de una enfermedad.

Lo antes expuesto representa un problema científico, cuya **situación problemática** gira en torno a que no existe una herramienta para registrar señales electro-oculográficas utilizando tarjetas de adquisición de datos. Además de que no existe software que almacene estas señales en formatos soportados por herramientas implementadas en la facultad para la investigación de enfermedades neurológicas mediante el procesamiento de señales, como el EogStudio.

La situación problemática antes expuesta permitió identificar el siguiente **problema de la investigación**: ¿Cómo registrar y almacenar las señales electro-oculográficas utilizando tarjetas de adquisición de datos?

A partir del problema se define como **objeto de estudio** el proceso de captura de señales electro-oculográficas.

Para solucionar el problema se persigue el siguiente **Objetivo**: Elaborar una herramienta informática que permita el registro y almacenamiento de señales electro-oculográficas utilizando tarjetas de adquisición de datos.

El objetivo de la investigación delimita el **Campo de acción**: El registro de señales electro-oculográficas.

Para guiar la investigación, se trazaron las siguientes **preguntas científicas**:

---

1 Software para el procesamiento de señales electro-oculográficas, desarrollado en la facultad de Informática y Matemática de la Universidad Oscar Lucero Moya de Holguín.

2 Movimiento brusco de los ojos.

- ¿Qué fundamentos teóricos sustentan la elaboración de una herramienta informática que permita el registro y almacenamiento de señales electro-oculográficas utilizando tarjetas de adquisición de datos?
- ¿Cuál es la situación actual referente a las aplicaciones para el registro y almacenamiento de señales electro-oculográficas?
- ¿Cómo desarrollar una herramienta informática que permita el registro y almacenamiento de señales electro-oculográficas utilizando tarjetas de adquisición de datos?
- ¿Cómo valorar la herramienta informática propuesta?

Para darles respuesta a las preguntas científicas y cumplir el objetivo trazado, se proponen las siguientes **tareas**:

1. Elaborar los fundamentos teóricos de la investigación.
2. Realizar el diagnóstico de la situación existente en las aplicaciones para el registro y almacenamiento de señales electro-oculográficas.
3. Diseñar la arquitectura de la herramienta informática.
4. Capturar los requerimientos funcionales.
5. Implementar y probar la herramienta informática.
6. Valorar el estado de aceptación de la herramienta informática propuesta mediante pruebas de aceptación.

Para darles solución a las tareas planteadas se usaron una combinación de métodos teóricos y empíricos.

Los **Métodos Empíricos** están presentes en los procesos de asimilación de información empírica y en la comprobación experimental.

Se empleó el método de **observación científica** para conocer y analizar la realidad científica referente a la captura y registro de las señales electro-oculográficas mediante la tarjetas de adquisición de datos. También se utilizó para conocer a grandes rasgos la situación problemática del diseño de la investigación.

La **revisión de documentos** se utilizó para recopilar información, entre los principales documentos analizados se encuentra "The Control and Measurement Device Interface handbook" que permitió entender el funcionamineto de las tarjetas de adquisión y sus *drivers.*

La **entrevista** permitió conocer la opinión y valoración del personal que se dedica a investigar en estos temas en la Facultad de Informática y Matemática en la Universidad de Holguín sobre los aspectos concluyentes.

Por último se utilizó la **experimentación** para comprobar mediante pruebas el funcionamiento y rendimiento de la herramienta en cuestión.

Los **Métodos Teóricos** permiten revelar las relaciones esenciales del objeto de investigación, no observables directamente. Participan en la etapa de asimilación de hechos, fenómenos y procesos

Se empleó el **análisis y síntesis** para el procesamiento de la información, lo que permitió arribar a las conclusiones de la investigación, distinguir las relaciones esenciales y características generales entre ellos,

mediante la abstracción. Permitió procesar la información en la elaboración de los fundamentos teóricos y las conclusiones, descomponer las necesidades en requerimientos del sistema para facilitar su comprensión.

El **enfoque sistémico** se utilizó para descomponer el sistema en subsistemas, así como establecer las relaciones entre ellos, lo que facilitó organizar la lógica del negocio identificada.

La **modelación** permitió representar de manera simplificada el proceso de captura y registro de señales posibilitando una mejor comprensión del mismo para obtener un producto mejor terminado.

La tesis consta de 3 capítulos, introducción, conclusiones, recomendaciones, bibliografía y anexos:

Capítulo 1. Marco Teórico: Se explican los conceptos y criterios que se utilizaron para el diseño de la herramienta.

Capítulo 2. Exploración y Planificación: Se hace un análisis de los requerimientos de la aplicación y se planifica el proceso de implementación. Se hace la exposición del negocio mediante Historias de Usuario.

Capítulo 3. Implementación y Pruebas: Primeramente se explica el diseño de la aplicación y luego el flujo de iteraciones utilizadas para su implantación. Este capítulo finaliza con las pruebas realizadas para comprobar la calidad de la aplicación.

Capítulo 1. Fundamentos teóricos

1.1 Introducción

La enfermedad conocida como ataxia espino cerebelosa se refiere a un estado patológico de la coordinación de los movimientos, caracterizada por trastornos de la marcha que se manifiestan por inestabilidad, descoordinación y aumento de la base de sustentación, como resultado de una disfunción a nivel del cerebelo o de sus vías, así como alteraciones en la médula espinal, nervios periféricos o una combinación de estas tres condiciones [1].

Uno de los órganos afectados por esta enfermedad es la visión por lo que se han desarrollado métodos para medir el movimiento de los ojos. Algunos de estos métodos utilizan equipamientos que generan señales, las cuales necesitan ser capturadas para su posterior análisis y tratamiento.

1.2 Captura de señales electro-oculográficas

Las señales electro-oculográficas como cualquier otra perteneciente al sistema nervioso contienen ruidos provenientes de otras partes del cuerpo, como el corazón, el cerebro, y los músculos. También los instrumentos electrónicos utilizados para capturar estas señales generan ruidos. Estas señales contienen información importante para comprender los mecanismos que subyacen en el funcionamiento de los seres humanos.

Para la captación de los movimientos oculares se han desarrollado un conjunto de métodos que permiten su registro digital. Cada uno de estos métodos presenta ventajas e inconvenientes, en dependencia del objetivo de las mediciones a realizar. Algunos pueden ser más exactos, pero a costa de utilizar técnicas más invasivas y molestas para el paciente, en otros, la simplicidad y comodidad, pueden ir acompañadas de menor exactitud o de respuestas espectrales inferiores.

1.2.1 Oculografía infrarroja

Es basada en la reflexión difusa de la luz infrarroja por la superficie frontal del globo ocular. Se ilumina el ojo mediante n fuentes de luz infrarroja y se colocan fotorreceptores que captan la luz reflejada, siguiendo el limbo (la frontera entre la esclerótica y el iris) para medir la rotación relativa de los ojos.

1.2.2 Video-oculografía

La video-oculografía (VOG) comprende varios métodos que descansan en el seguimiento de características visibles del ojo, o reflexiones en su superficie. Las características primarias que se siguen son la posición y forma aparente de la pupila, la reflexión corneal (imagen de Purkinje) y, para algunos sistemas que censan también el movimiento torsional, el iris. Las mejorías tecnológicas logradas en las cámaras, conjuntamente con los avances en las computadoras han permitido el rápido incremento de la velocidad y exactitud de estos métodos.

1.2.3 Electro-oculografía

El ojo tiene una diferencia de potencial entre su parte delantera y trasera, comúnmente llamada potencial córneo-

funda, el cual se deriva del epitelio pigmentado de la retina (RPE), este cambia en respuesta a la iluminación que llega a la retina por lo que es muy importante mantener condiciones de iluminación constantes cuando se emplea este método. Esto fue aprovechado por Fenn y Hursh en 1934 para introducir la electo-oculografía (EOG), midiendo este potencial por medio de electrodos ubicados en la piel alrededor del ojo, convertido en una señal que mide el ángulo de rotación de los ojos [2]. Funciona muy bien para movimientos horizontales, sobre todo cuando se desean mediciones relativas del movimiento de los ojos, más que la posición absoluta de estos.

1.3 Sistemas de captura de señales existentes

El registro y análisis de los movimientos oculares son fundamentales para un diverso conjunto de investigaciones en las que se incluyen estudios en los que la lectura, la búsqueda visual y la atención son examinadas. Las herramientas de software incluidas en el equipamiento usado generalmente para registrar señales electro-oculográficas están generalmente limitadas en funcionalidad y extensibilidad.

- kacq: Software difícil de entender, con pocas opciones de configuración que a pesar de informar que lee y escribir un fichero, este no es llenado con los datos y la señal no se muestra debido a que en realidad no es compatible con todo tipo de dispositivos. Esto se debe a que la aplicación no está implementada lo suficientemente genérica.

- comedirecord: Es un software que detecta y captura señales de manera automática, permite exportar los datos en un fichero texto de forma cruda. Sin embargo no tiene en cuenta las posibilidades que brinde el dispositivo al no permitir escogerlo, ni cual referencia o rango utilizar, tampoco la frecuencia de adquisición. Otra de las desventajas es que solo puede leer en 7 o menos canales y la interfaz gráfica no es muy amigable.

1.4 Tecnologías y herramientas

Para la construcción de sistemas informáticos un aspecto muy importante a tener en cuenta son las herramientas y tecnologías sobre las que se van a soportar estos; desde lenguajes de programación, frameworks, bibliotecas, entornos de desarrollo integrado (IDE por sus siglas en inglés) hasta la metodología de desarrollo con que se trabajará. En los siguientes epígrafes se realizará una revisión del estado del arte de estas tecnologías y herramientas.

1.4.1 Lenguajes de programación

La base de todo programa o sistema informático es el código con el que este fue elaborado. Los lenguajes de

programación permiten expresar ideas o abstracciones de la realidad en forma de código. La calidad de los programas es usualmente determinada por la calidad de este código, y la calidad de este código puede también ser condicionada por la elección de un determinado lenguaje de programación.

Cada lenguaje de programación cuenta con un cierto nivel de expresividad, lo que influye en su uso en los distintos escenarios que pueda aplicarse. Esta expresividad está dada fundamentalmente por los paradigmas que implementa, los más utilizados hoy en día son los siguientes:

- **Imperativo:** Describe la programación en términos del estado del programa y sentencias que cambian dicho estado. Los programas imperativos son un conjunto de instrucciones que le indican a la computadora cómo realizar una tarea.
- **Orientado a Objetos:** La programación orientada a objetos (POO) introduce un mayor nivel de abstracción que sus predecesores. Los conceptos de Clases y Objetos proporcionan una abstracción del mundo centrada en los seres y no en los verbos. Los datos aparecen encapsulados dentro del concepto de clase y mediante conceptos como la composición, la herencia y el polimorfismo se pueden implementar relaciones entre las Clases y Objetos [3].
- **Funcional:** El paradigma funcional está basado en el concepto matemático de función. El paradigma funcional considera al programa como una función matemática donde el dominio representaría el dominio de todas las entradas posibles y el rango sería el conjunto de todas las salidas posibles. La forma en que funciona puede ser entendida como una caja negra [4].
- **Orientado a Aspectos:** Este paradigma persigue permitir que un programa sea construido describiendo conceptos de forma separada. Los lenguajes orientados a aspectos brindan mecanismos y constructores para capturar a aquellos elementos que se diseminan por todo el sistema [18].
- **Lógica:** En este paradigma la lógica representa conocimiento, el cual es manipulado mediante inferencias. A diferencia de los demás paradigmas, trabajar en este significa qué hacer y no cómo hacerlo, por ello son llamados lenguajes declarativos [4].

A continuación se comentarán las principales características de los lenguajes de programación más conocidos. Es importante señalar que aunque no son los únicos lenguajes existentes, si son los más utilizados para desarrollo de software y de más relevancia.

- **Java:** Lenguaje orientado a objetos puro e interpretado mediante bytecode. Es el preferido para la construcción de aplicaciones empresariales. Su principal debilidad es el pobre rendimiento que posee para

escenarios que requieran uso intensivo de los recursos computacionales.

- **C:** Lenguaje procedural y compilado. De preferencia para escribir controladores de dispositivos y código de bajo nivel. Su principal debilidad es su pobre expresividad para escenarios más complejos, sacrificando productividad.

- **C++:** Lenguaje orientado a objetos y compilado. Es el indicado a la hora de implementar software de alta complejidad y que requiera altos índices de rendimiento. Su principal debilidad es debido a que su sintaxis es algo compleja y tiene menos expresividad que otros competidores como Python y C#.

- **Python:** Lenguaje orientado a objetos, funcional y orientado a aspectos e interpretado mediante código fuente o bytecode. El más utilizado para escribir aplicaciones utilitarias de sistemas operativos libres. Su principal debilidad al igual que Java yace en su rendimiento.

- **PHP:** Lenguaje orientado a objetos e interpretado mediante código fuente. El preferido para escribir aplicaciones web por su ligereza y simplicidad. Su principal debilidad está en que se encuentra orientado solo a la web y no hacia otros usos.

- **C#:** Lenguaje orientado a objetos y funcional y compilado mediante JIT. Es la elección para escribir aplicaciones de escritorio en la plataforma Windows. Su principal debilidad está dada en que la calidad de las implementaciones fuera del sistema operativo Windows es muy baja.

Luego de una extensiva revisión de las tecnologías disponibles y la tendencia actual en este tipo de aplicaciones se decide utilizar como **lenguaje de programación** Python. Este lenguaje cumple con los estándares de rendimiento requeridos por el PySigRec, además de que su sintaxis se encuentra estandarizada internacionalmente; y existen compiladores libres de gran calidad y en varios sistemas operativos, aunque el interés de esta investigación se centra en los sistemas operativos UNIX. Otra de las razones para la elección del lenguaje es la facilidad de aprenderlo rápidamente por su poca complejidad, la posibilidad de realizar pruebas al vuelo y las bibliotecas implementadas para este.

1.4.2 Bibliotecas de Interfaces de Usuario (UI)

Una parte muy importante de cualquier sistema informático es la interacción de esta con el usuario final. De ahí a que la interfaz utilizada por el sistema se convierta en un aspecto de gran importancia para la aplicación. Existen

3 grandes formas de realizar aplicaciones en la actualidad: aplicaciones de escritorio, aplicaciones web y aplicaciones para dispositivos empotrados. Para el área de procesamiento de señales las más utilizadas son las de escritorio por sus necesidades en cuanto a rendimiento.

Entre las bibliotecas de interfaces de usuario (o widget toolkits) más usadas en la actualidad para desarrollar aplicaciones de escritorio encontramos las siguientes:

- **Microsoft Foundation Classes (MFC):** Envoltorios en C++ para los componentes de las API de Windows. Una de las de preferencia para la plataforma Windows. Su principal debilidad consisten en la falta de lenguajes de programación y plataformas que la soporten.

- **Windows Forms:** Conjunto de componentes para desarrollar aplicaciones de escritorio en la plataforma .NET de Microsoft. Muy populares por su facilidad de uso, además de que pueden ser usadas por todos los lenguajes soportados por la plataforma .NET. Al igual que MFC, no se les brinda soporte a otros lenguajes que no estén en .NET, ni fuera de Windows.

- **Cocoa:** Principal conjunto de widgets de la plataforma Mac OS X. Junto con Carbón son los toolkits de preferencia para Mac OS X. Utilizados mayormente con el lenguaje Objetive-C, tienen portes hacia otros lenguajes como C y C++. No tienen implementaciones fuera de Mac OS X.

- **Gtk+ (Gimp Toolkit):** Toolkit de plataforma cruzada, inicialmente creado para la herramienta de manipulación de imágenes GIMP. Es usado actualmente por el escritorio GNOME. El API de este se encuentra implementado en C, aunque existen envoltorios para otros lenguajes como Python, C++, C#, entre otros. Fuera de las plataformas Linux, solo tiene implementación en Windows. Es liberado bajo la licencia LGPL.

- **Qt:** Conjunto de widgets y biblioteca de clases de plataforma cruzada creado por Trolltech y mantenido actualmente por Nokia. Es implementado usando lenguaje C++, aunque tiene envoltorios para casi todos los mayores lenguajes del mercado como Java, Python y C#. Es bastante eficiente y tiene portes para casi todas las plataformas y arquitecturas en la actualidad, además de tener implementaciones en múltiples dispositivos móviles y empotrados. Es liberado bajo la licencia LGPL.

- **Tk:** Toolkit multiplataforma implementado inicialmente para el lenguaje Tcl. Solo soportado por lenguajes interpretados como Tcl y Python; y solo en las plataformas en las que estos tienen implementaciones.

- **wxWidgets:** Toolkit de plataforma cruzada que contiene abstracciones de componentes visuales para varias plataformas. Cuenta con envoltorios para distintos lenguajes como C++, Python, Perl, Ruby y Haskell. Tiene implementaciones en distintas plataformas como Windows y GNU/Linux y es liberado bajo la licencia LGPL.

- **Swing:** Biblioteca multiplataforma para crear aplicaciones de escritorio en el lenguaje Java. Creada por Sun Microsystems y mantenido actualmente por Oracle, es el toolkit de preferencia para hacer aplicaciones de escritorio en Java. Su principal problema consiste en su bajo rendimiento y la falta de implementaciones en otros lenguajes.

- **PySide**: Biblioteca de QT para Python, posee las mismas funcionalidades antes expuestas, pero se pueden utilizar en programas hechos con python.

Existen 3 características fundamentales a la hora de elegir una biblioteca de interfaces de usuario. La primera es que se le brinde soporte al lenguaje de programación seleccionado para desarrollar el sistema. La segunda es que exista una implementación en la plataforma destinada a desplegar el sistema. No menos importante es la tercera de estas características, que se refiere a la licencia con la que se libera.

Las **bibliotecas de interfaces de usuario** que mejor se adaptan al Python son las PySide. Las cuales tienen además como ventajas que son liberadas bajo la licencia LGPL, que es una licencia de código abierto.

1.4.3 Entornos de Desarrollo Integrado (IDE)

Para desarrollar software moderno es preciso contar con modernas herramientas que faciliten y agilicen el trabajo de codificar, construir, desplegar y probar. A menudo estas herramientas vienen en un solo paquete denominado Entorno de Desarrollo Integrado (IDE). Los IDEs no son más que una herramienta que integra un grupo de tecnologías para ponerlas en función del desarrollador [5].

Las herramientas más comunes que encontramos hoy en un IDE son:

- **Editor de Código:** Usualmente cuentan con resaltado de código y facilidades para la navegación del mismo.

- **Gestor de Proyectos:** Manejan los archivos que pertenecen al proyecto, tanto los de código fuente como los de recursos. Se integran con los sistemas de construcción de diferentes tecnologías y lenguajes.

- **Explorador:** Facilitan la navegación del código del proyecto, permitiendo ir a determinadas clases y métodos.

- **Diseño de Interfaces Gráficas:** Se integran con distintas herramientas de diseño de interfaces gráficas WYSIWYG de diferentes tecnologías, facilitando su integración con el código del proyecto.

- **Depurador:** Permiten evaluar en tiempo de ejecución determinados fragmentos de código para buscar errores.

- **Ayuda Integrada:** Consultas a la ayuda de determinada tecnología desde el código. A veces llamada ayuda contextual, permite presionando una tecla dentro del editor, ir a la ayuda de determinada clase o método.

Los IDE de mayor calidad hoy son plataformas de desarrollo que extienden su funcionalidad mediante plugins. Dadas las características expuestas anteriormente, los de mayor relevancia son:

- **Eclipse:** Creado inicialmente como entorno de desarrollo para el lenguaje Java, se ha convertido en una plataforma de desarrollo rica, donde ya no solo se realizan aplicaciones de programación, sino que se ha extendido su uso a otros escenarios como la oficina, el diseño, etc. Cuenta con soporte para un gran número de lenguajes. Cuenta con binarios para las mayores plataformas de escritorio.

- **Netbeans:** Debido a que fue creado por los ingenieros de Sun Microsystems, su principal fortaleza se encuentra en el gran soporte que le brinda al lenguaje Java. Aunque también cuenta con soporte para otro gran número de lenguajes. Tiene binarios en las principales plataformas de escritorio.

- **Visual Studio .NET:** Es el entorno de desarrollo de Microsoft, y según la opinión de muchos el mejor IDE en este momento. Ha sido un pionero en la integración e introducción de nuevas tecnologías como IntelliSense. Solo se le da soporte a la plataforma Windows.

- **XCode:** IDE de preferencia para escribir aplicaciones para Mac OS X y plataformas empotradas de Apple Inc. como IPhone e IPad. Sus principales lenguajes de desarrollo son Objetive-C, Objetive-C++, C y C++. Terceros le han añadido soporte a otros lenguajes. Solo se despliega en la plataforma Mac OS X.

QtCreator: Es el preferido para escribir aplicaciones con las bibliotecas Qt. Le brinda soporte a los lenguajes C++

y QML, en el futuro se planea incluir a otros como Python. Cuenta con binarios en las principales plataformas de escritorio.

Como **Entorno de Desarrollo Integrado**, se selecciona el Eclipse. Este cuenta con gran soporte para el lenguaje Python. Además como herramienta adicional se puede utilizar el QtDesigner para diseñar las interfaces de usuarios y luego exportar el código de Qt a Python usando el envoltorio PySide[3].

1.4.4 Herramientas CASE

Una buena parte del tiempo de desarrollo de un sistema informático se dedica al diseño. Existen varias herramientas y lenguajes que se han creado para facilitar esa parte del trabajo. Ese es el caso de la Ingeniería de Software Asistida por Computadoras (CASE por sus siglas en inglés).

Esto no es más que la aplicación científica de un conjunto de herramientas y métodos a un sistema de software, el cual se quiere que resulte en un producto de software de alta calidad y libre de defectos[6].

También se refiere a los métodos de desarrollo de sistemas de información junto con herramientas automáticas que pueden usarse en el proceso de desarrollo de software [7].

El Lenguaje Unificado de Modelado (UML por sus siglas en inglés) es un lenguaje gráfico para visualizar, especificar, construir y documentar artefactos de un sistema de software. Este ofrece una vía estándar para escribir los planos de sistemas informáticos, donde se incluyen aspectos conceptuales como procesos de negocios y funciones de sistemas, o aspectos concretos como sentencias en lenguajes de programación, esquemas de bases de datos y componentes de software reusables [8].

Cuando se complementan CASE con el lenguaje UML en una aplicación informática se crea lo que usualmente conocemos como herramienta CASE. Existen una gran variedad de estas en la actualidad, orientada a disímiles usos y metodologías de desarrollo.

Entre ellas destaca por su gran conjunto de funcionalidades e integración con varios IDE el Visual Paradigm. Esta es una poderosa herramienta CASE de diseño UML diseñada para una amplia gama de usuarios, donde se incluyen Ingenieros de Software, Analistas de Sistema, Analistas de Negocio y Arquitectos de Sistema, así como otros interesados en construir software mediante el uso del acercamiento orientado a objetos [9].

[3] Envoltorio de clases o código que le permite al lenguaje python la utilización de las bibliotecas de Qt.

Entre sus aciertos se encuentra que implementa la última especificación del lenguaje UML (la versión 2.1), cuenta con soporte para las principales plataformas de escritorio y permite mantener sincronizado código fuente con diagramas. Cuenta con un problema mayor y es que la versión gratis que es mantenida por la comunidad dista mucho en funcionalidad de la versión empresarial cuya licencia cuesta alrededor de 800 USD.

Existen otras herramientas privativas como el Rational Rose (pionero en este tipo de herramientas) o el Enterprise Architect; y otras libres como el Umbrello UML, el ArgoUML, el BoUML, entre otras pero que distan mucho en la calidad y actualización del Visual Paradigm.

Como **Herramienta CASE** se seleccionará el Visual Paradigm, lo que se debe a que su calidad es muy superior al resto de los competidores. Para la modelación UML se utilizará la versión desarrollada por la comunidad.

1.5 Metodologías de desarrollo

Según la literatura, una metodología de desarrollo de software se refiere a los métodos, reglas, postulados, procedimientos y procesos que se usan para administrar un proyecto de ingeniería de software [10]. Estas se dividen en 2 grandes familias: metodologías tradicionales y metodologías ágiles. En los siguientes epígrafes se hará una comparación entre las dos grandes familias de metodologías, para luego seleccionar la que se utilizará en la confección del sistema en que se desarrollará este trabajo.

1.5.1 Metodologías Ágiles vs Tradicionales

El nombre de metodología ágil viene de cuando en el 2001, 17 metodólogos de procesos tuvieron una reunión para discutir tendencias futuras del desarrollo de software. En esta reunión firmaron lo que se conoce por el *manifiesto ágil [11]*, que no es más que los 12 principios básicos de una serie de metodologías que serían conocidas por ágiles para marcar el contraste con las tradicionales o pesadas existentes hasta el momento. Estos principios son los siguientes [12]:

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.

2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.

3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al período de tiempo más corto posible.

4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.

5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.

6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.

7. El software funcionando es la medida principal de progreso.

8. Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.

9. La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.

10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.

11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.

12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

En estos principios se puede observar que los mismos se centran en las personas y no tanto en los procesos, así como en entrega rápida de software y no en complejos diseños previendo el futuro. Ejemplos de metodologías ágiles son la Programación Extrema (XP), el Desarrollo Dirigido por Características (FDD) y Scrum.

Las metodologías pesadas son consideradas como la vía tradicional de desarrollar software. Estas metodologías se basan en una secuencia de pasos, como definición de requerimientos, construcción de la solución, pruebas y despliegue. Estas metodologías requieren definir y documentar un conjunto estable de requerimientos al comenzar el proyecto [11]. Existen varias metodologías pesadas diferentes, donde las más conocidas son: Cascada (Waterfall), Modelo en Espiral (Spiral Model) y el Proceso Unificado de Desarrollo (Rational Unified Process).

Las metodologías pesadas han existido durante un largo tiempo. Ellas imponen un proceso disciplinado en el desarrollo de software con vistas a que este sea más predecible y eficiente. Estos no se han dado a conocer como muy eficientes y menos aún populares [11] siendo muy criticados por ser metodologías burocráticas, ya que habiendo tanto que seguir en la metodología el proceso de software como un todo se ralentiza [13].

Existen 4 características comunes de las metodologías pesadas [11]:

- **Enfoque Predictivo:** Las metodologías pesadas tienen la tendencia a planificar primero grandes partes del proceso de software en gran detalle por un largo período de tiempo. Este acercamiento sigue una disciplina de ingeniería donde el desarrollo es predecible y repetitivo.

- **Documentación amplia:** El desarrollo de software tradicional mira los requerimientos como una pieza clave de documentación. Un proceso principal en las metodologías pesadas es el gran diseño por adelantado, en el cuál se cree que es posible recoger todos los requerimientos del cliente por adelantado, antes de escribir ningún código.

- **Orientación al Proceso:** El objetivo de las metodologías pesadas es definir un proceso que trabaje bien para cualquiera que lo utilice. El proceso consistirá en determinadas tareas que deben realizar los administradores, diseñadores, programadores, probadores, etc. Para cada una de estas tareas existe un procedimiento bien definido.

- **Orientación a las Herramientas:** Herramientas de gestión de proyectos, editores de código, compiladores, etc.; deben usarse para completar y entregar cada una de las tareas.

Las metodologías tradicionales se enfocan más en pensar en el futuro que en entregar software a mediano y corto plazo, como es la necesidad de la mayoría de los usuarios. Además de estar orientados a grandes proyectos donde se tienen gran cantidad de programadores distribuidos en una serie de roles para lograr la organización del trabajo.

Hoy en día es muy importante tener el trabajo realizado en el plazo más corto posible de tiempo. En este aspecto las metodologías ágiles han ido ganando gran popularidad, incluso en grandes empresas como Microsoft, Nokia, Google, etc.

1.5.2 Principales características de las metodologías ágiles

Existe un número de metodologías de desarrollo ágiles, como las que son apoyadas por La Alianza Ágil (The Agille Alliance). La mayoría de estas metodologías intentan minimizar el riesgo desarrollando software en ciclos de desarrollo cortos llamados iteraciones, los que típicamente duran de una a cuatro semanas.

Cada iteración es como un proyecto de software en miniatura, e incluye todas las tareas necesarias para liberar el mini-incremento de una nueva funcionalidad: planeación, análisis de requerimientos, diseño, codificación, pruebas y documentación. Mientras que una iteración puede no adicionar suficiente funcionalidad que garantice la liberación del producto, un proyecto de software ágil tiende a ser capaz de entregar nuevo software al final de cada iteración. Al final de cada iteración además el equipo puede reevaluar las prioridades del proyecto [14].

Las metodologías ágiles enfatizan en la necesidad de comunicación en tiempo real, preferentemente cara a cara en vez de documentos escritos. La mayoría de los equipos ágiles localizan a sus miembros en un mismo lugar e incluyen las personas necesarias para terminar el software. Como mínimo incluye a programadores y clientes (personas que definen el producto, ellos pueden ser gerentes del producto, analistas de sistema o realmente los propios clientes). También pueden incluirse probadores, diseñadores de iteración, escritores técnicos y gerentes [14].

Además se enfatiza que el software trabajando, es la principal medida de progreso que combinado con la preferencia de la comunicación cara a cara, producen una cantidad muy pequeña de documentación comparadas con otros métodos tradicionales [14].

Entre las principales metodologías ágiles tenemos:

- **Programación Extrema (XP):** Es una evolución de los problemas causados por los largos ciclos de desarrollo del modelo tradicional [15]. Después de un número exitoso de ensayos en la práctica, la metodología XP fue teorizada en base a las prácticas y principios utilizados [16].

El ciclo de vida de XP consiste en 5 fases como se pueden observar en la : Exploración, Planeación, Iteraciones con Lanzamientos, Producción, Mantenimiento y Muerte. Entre sus roles encontramos: Programador, Cliente, Probador, Rastreador, Instructor, Consultante y Gerente [17].

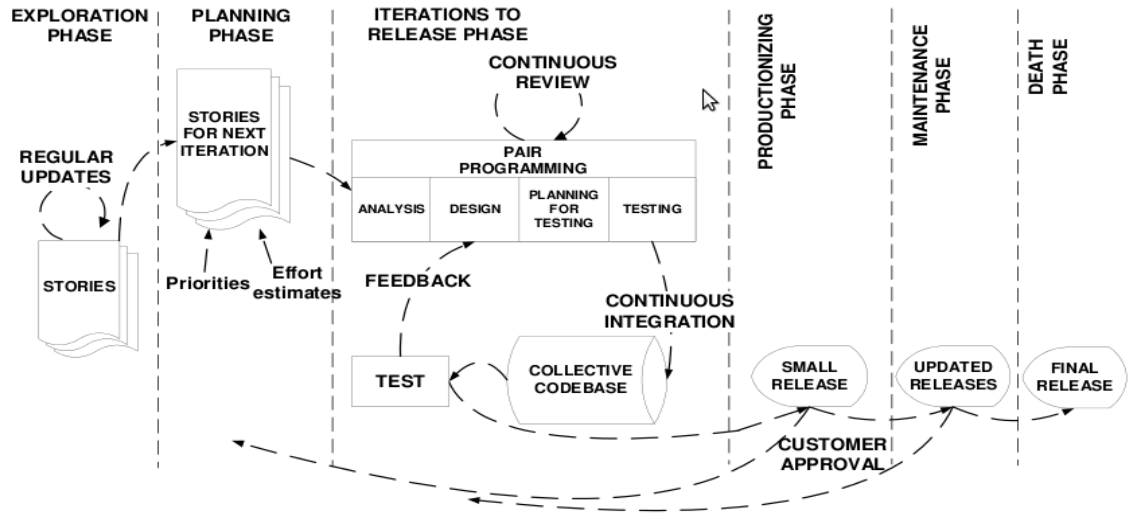

Figura 1: Proceso de desarrollo de XP

Entre sus prácticas encontramos el juego de planeación, pequeños y cortos lanzamientos, metáforas para definir el sistema como un todo, diseño simple, desarrollo dirigido por pruebas, refactorización, programación por parejas, pertenencia colectiva del código, integración continua, semanas de 40 horas, cliente en el sitio de desarrollo, estándares de código, espacio de trabajo abierto y reglas de desarrollo redactadas y aplicadas por el propio equipo [17].

En el siguiente epígrafe se hará una explicación de las prácticas de XP en mayor detalle.

- **Scrum:** La primera referencia en la literatura al término Scrum apunta al artículo de Takeuchi y Nonaka en 1986 en el cual un proceso de desarrollo de productos adaptativo, rápido y auto-organizado, originado en Japón es presentado. El enfoque de Scrum ha sido desarrollado para administrar el proceso de desarrollo de sistemas. Constituye un acercamiento empírico donde se aplican ideas de la teoría del proceso de control industrial resultando en un enfoque que reintroduce las ideas de flexibilidad, adaptabilidad y productividad [18].

El proceso de Scrum incluye tres fases como se observa en la: pre-juego, desarrollo y pos-juego. La mayor parte del proceso de desarrollo se concentra en la segunda fase, donde se realizan las iteraciones. Entre los roles de Scrum se encuentran: Maestro Scrum, Dueño del Producto, Equipo de Scrum, Cliente y Administración [17].

Entre sus prácticas se halla la utilización de pilas de historias de usuario, tanto la del producto en general como la de cada iteración, donde se detallan las características del negocio a implementar en el sistema. La estimación del esfuerzo en las historias de usuario, los sprints como unidad de iteración y todas las reuniones que se realizan

en este como la planificación del Scrum, el Scrum diario y la Retrospectiva del Sprint en las que se planifica y analiza el proceso de desarrollo Scrum [17].

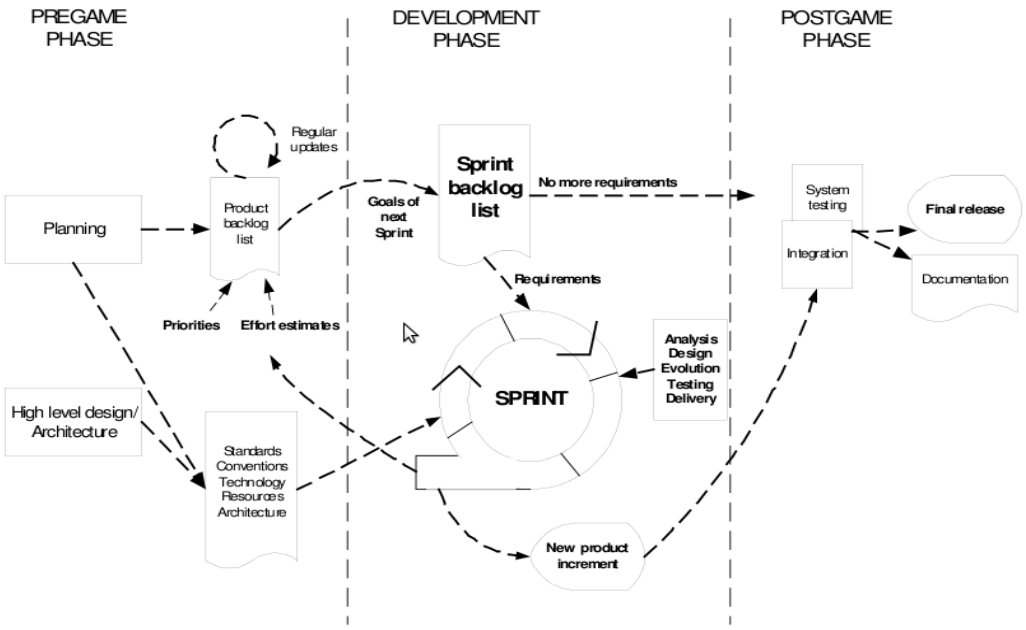

Figura 2: Proceso de desarrollo de Scrum

- **Desarrollo Dirigido por Características (FDD):** Esta metodología es un enfoque ágil y adaptativo de desarrollo de sistemas. No cubre completamente todo el proceso de desarrollo de software, sino que se enfoca en las fases de diseño y construcción. Sin embargo ha sido diseñada para trabajar con otras actividades de un proyecto de desarrollo de software y no requiere que se use ningún modelo en específico [19].

El FDD consiste en 5 procesos secuenciales que ocurren durante las fases de diseño y construcción del sistema que se desarrolla como se muestra en la figura 3.

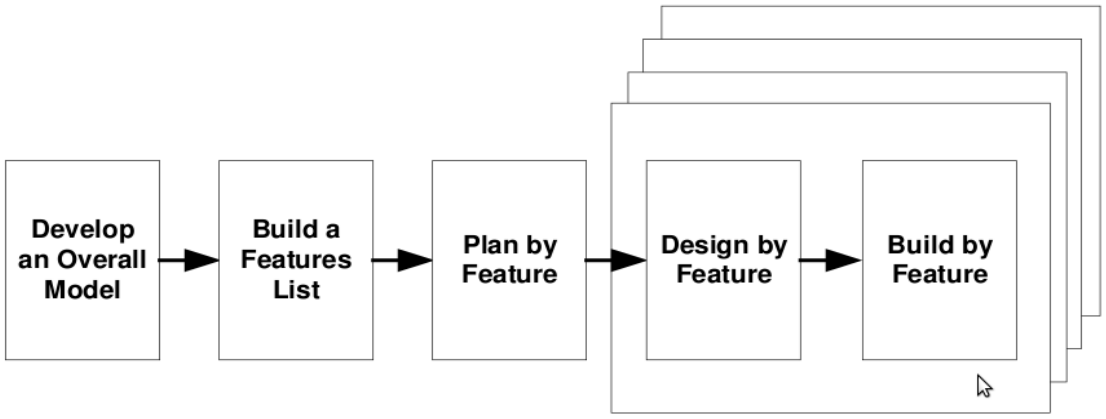

Figura 3: Proceso de desarrollo de FDD

Los roles de FDD son: Gerente de Proyecto, Arquitecto Jefe, Gerente de Desarrollo, Programador Jefe, Dueño de Clases, Experto en Dominio, Gerente de Dominio, Gerente de Lanzamientos, Gurú de Lenguaje, Ingeniero de Construcción, Constructor de Herramientas, Administrador de Sistema, Probador, Responsable de Despliegue y Escritor Técnico.

Como se puede observar casi para cada tipo de tarea existe un rol específico en FDD, que aunque una persona puede tomar más de un rol a la vez, esto puede ser tomado como desventaja en escenarios donde se requiera que los miembros del equipo sean versátiles [17].

Entre las principales prácticas de FDD encontramos la modelación del dominio de objetos donde se explora y explica el dominio del problema, el desarrollo por característica donde se le da seguimiento el progreso del proyecto a través de una lista de pequeñas funcionalidades, cada clase tiene un responsable de su integridad, los equipos son conformados con pocas personas enfocadas en una serie de características, inspección al código, construcción regular, manejo de configuración a través de sistemas de control de versiones, reporte del progreso basado en el trabajo completado [17].

1.5.3 Programación Extrema (XP)

La programación extrema fue concebida y desarrollada para solucionar las necesidades específicas del desarrollo de software de pequeños equipos que encaran requerimientos escasos y cambiantes. Esta nueva y ligera metodología desafía muchos dogmas convencionales, incluida la vieja asunción que el costo de cambiar una pieza en un software incrementa dramáticamente en el tiempo [16].

Algunos aspectos fundamentales de XP son [16]:

- **Distinción entre las decisiones a tomar por la parte del negocio y las decisiones que toman los miembros del equipo del proyecto.** Esto quiere decir que las decisiones sobre las funcionalidades del sistema y sus prioridades las toma el cliente, mientras que las de la cantidad de trabajo a realizar por cada iteración, y la organización, el equipo. Es muy importante seguir esta práctica al pie de la letra ya que permite que cada parte del equipo se enfoque en su función y no sabotee el trabajo de su contrapartida.

- **Escribir pruebas unitarias antes de comenzar a escribir el código y mantenerlas corriendo todo el tiempo.** Una de las prácticas básicas y más difíciles de seguir en XP es el desarrollo guiado por pruebas. El escribir la prueba antes del código, permite tener una medida de código terminado, lo cual ocurre cuando la prueba corre correctamente. Esto brinda las ventajas de tener software más estable y de mayor calidad, evitándose errores de regresión.

- **Integración y prueba de todo el sistema varias veces al día.** Cada vez que se añada o cambie una pieza de código realizar la compilación e integración de todo el sistema. Permitiendo llevar el seguimiento del progreso del sistema y evitar el surgimiento de nuevos errores.

- **Producción de todo el software en parejas.** En este, dos miembros del equipo se sientan en una sola estación de trabajo y crean código en conjunto. Usualmente se empareja un programador novato con uno avanzado logrando elevar tanto el nivel individual como colectivo de los miembros del equipo. También se recomienda que las parejas no sean fijas sino que los miembros de estas, roten con otros miembros del equipo, contribuyendo al principio ágil de la pertenencia colectiva de código.

- **Comenzar los proyectos con un diseño simple que evolucione constantemente para añadir la flexibilidad necesaria y elimine la complejidad innecesaria.** Otra práctica difícil de seguir en XP, debido a que mayormente los programadores tienden a pensar primero en el *futuro*, lo que entra en contradicción con lo que XP basa sus cimientos: el cambio es inevitable e impredecible.

  La tendencia a crear diseños complejos inicialmente, resulta la mayoría de las veces en funcionalidades que nadie usa, incurriendo en gastos de recursos innecesarios. Por lo que una de las ideas básicas en XP consiste en que siempre el diseño más simple es el mejor diseño.

- **Poner un sistema mínimo en producción lo más rápido posible, creciendo en la dirección que pruebe ser la de mayor valor.** Para los clientes no hay mejor medida de progreso que el software funcionando, sin importar cuan buen y flexible sea el diseño interno de este. Muchas veces los propios clientes no tienen claro que dirección quieren seguir con el proyecto. El entregar en el menor plazo posible le permite evaluar cuáles son las características de mayor importancia a implementar a continuación o detener el desarrollo de características innecesarias.

Todo equipo de desarrollo de software por mucho que se desee *auto-organizar* debe tener algún tipo de jerarquía para poder funcionar de forma correcta y cada cuál tenga una función determinada. En XP existe un grupo de roles que le dan solución a este problema. Es importante señalar que estos roles pueden o no ser exclusivos de una persona y hasta un miembro del equipo puede tener varios roles a la vez. Estos roles son [16]:

- **Programador:** Es el corazón de XP, es la clase trabajadora. Este es el que lleva a cabo el trabajo de codificación y prueba del sistema. Como miembro del equipo de desarrollo puede tomar decisiones de estimación sobre que trabajo aceptar para la iteración y realizar la estimación de este.

- **Cliente:** Este rol cumple con el trabajo como contrapartida del programador. Este es el que decide qué trabajo va a realizar el programador, además de especificar prioridades. La persona que asuma este rol debería trabajar cara a cara con el equipo de desarrollo para que la comunicación entre el cliente y el equipo fluya de forma efectiva y natural.

- **Probador:** Rol de la parte del cliente que no debe confundirse con el que realizan los programadores al realizar desarrollo guiado por pruebas. Este rol se refiere a la parte del cliente que prueba las funcionalidades pactadas entre las partes en el sistema desplegado.

- **Rastreador:** Se considera como el historiador del proyecto, este rol tiene la responsabilidad de llevar el progreso del proyecto, además de asesorar a los programadores en las estimaciones que se realicen. Al analizar el progreso del proyecto en todo su ciclo de vida, se puede predecir comportamientos futuros y la detección de problemas en los equipos de trabajo.

- **Instructor:** Es el responsable de que el proceso de XP se aplique al pie de la letra. El miembro del equipo que asuma este rol debe ser un experto en todo lo relacionado a XP y su aplicación. Además de asesorar a los miembros del equipo para que cumpla con las reglas de este. También debe ser capaz de realizar

adaptaciones al proceso que se adecuen mejor a las características del equipo de trabajo.

- **Consultante:** Este rol viene muy útil al proceso de XP cuando se necesite a alguien que tenga un determinado nivel de profundidad en un tema de utilidad para el proyecto. Usualmente al usar programación por parejas cualquiera de los programadores del equipo puede elegir tomar una determinada tarea. Cuando el conocimiento no es suficiente para completar una tarea, se puede llamar una fuente externa o un propio miembro del equipo con mayor experiencia sobre el tema en cuestión y pedirle que ejerza el rol de consultante.

- **Gerente:** Es el que mantiene al equipo enfocado en una visión general, además de realizar tareas administrativas. Este debe preferentemente tener alguna base de desarrollo de software ya que es el responsable de tomar las decisiones globales del proceso de desarrollo, además de enfocar al equipo en un objetivo general.

La *historia* es la unidad de funcionalidad en un proyecto XP. El progreso de un proyecto de XP se demuestra a través del código de una historia debidamente probada e integrada. Una historia debe ser entendible tanto por el cliente como por el programador, demostrable a través de algún tipo de prueba, de valor para el primero y lo suficientemente pequeña para que los segundos puedan construir alrededor de media docena de estas en cada iteración [20]. Las historias deben escribirse por el cliente y definir funcionalidades del sistema en el lenguaje del negocio [21]. En la se puede ver los campos más comunes en una Historia de Usuario.

Historia de Usuario	
No.: [Id]	**Nombre:** [Nombre de la Historia]
Usuario: [Todos]	
Prioridad en el Negocio: [Alta, Media, Baja]	**Nivel de Complejidad:** [Alta, Media, Baja]
Estimación: [En semanas laborables]	**Iteración Asignada:** 1
Descripción: [Breve descripción de la historia]	
Información Adicional (Observaciones): [Aquí se pone a qué requisitos da cumplimiento esta historia, entre otras consideraciones]	

Tabla 1: Prototipo de Historia de Usuario

Las historias a su vez se subdividen en tareas más concretas. Estas tareas de carácter técnico, permiten dividir una funcionalidad del negocio en unidades pequeñas estimables por los programadores. Al completar tareas podemos saber el por-ciento del trabajo realizado en cada historia de usuario y llevar trazas de las mismas.

Pueden existir además tareas independientes de las historias de usuario que den cumplimiento con una necesidad técnica en específico, como usualmente son las que tienen que ver con el soporte. En la se muestra un prototipo de Tarea, las que son escritas por los propios programadores.

Tarea	
Número de Tarea: [#]	**Número de HU:** [#]
Nombre de la Tarea: [Nombre de la tarea]	
Tipo de Tarea: [Tipo de Tarea]	**Estimación:** [Días de duración de la tarea]
Fecha de Inicio: [Fecha de Inicio]	**Fecha de Fin:** [Fecha de Fin]
Programador Responsable: [Nombre del Programador Responsable]	
Descripción: [Descripción de la Tarea]	

Tabla 2: Prototipo de Tarea

El otro artefacto usado por XP para documentar el proceso son las Pruebas de Aceptación, las que describen escenarios donde se ponen a prueba funcionalidades del sistema desplegado. Este tipo de pruebas determina cuando las historias de usuario han sido completadas [20]. En la Tabla 3 se muestra un prototipo de Prueba de Aceptación.

Caso de Prueba de Aceptación	
Código: [Código de la Prueba]	**Historia de Usuario:** [#]
Nombre: [Nombre de la Prueba]	
Descripción: [Descripción breve de la Prueba]	
Condiciones de Ejecución: [Pre-condiciones para ejecutar la prueba]	
Entrada/Pasos de Ejecución: [Descripción del caso de prueba paso a paso]	
Resultado Esperado: [Resultado Esperado de la Prueba]	
Evaluación de la Prueba: [Prueba Satisfactoria o No]	

Tabla 3: Prototipo de Prueba de Aceptación

Como se planteó en el epígrafe anterior, el proceso de XP cuenta con 6 fases (ver Figura 1). Cada una de vital importancia para todo el proceso. A continuación se describe en detalles las funciones de cada una de estas fases:

- **Exploración:** El cliente define las historias de usuario que desea incluir en la primera liberación del producto. Al mismo tiempo los miembros del equipo se familiarizan con las herramientas, tecnologías y prácticas que usarán durante el proyecto. Se probarán las tecnologías que se usarán y las posibilidades de arquitectura para el sistema, construyendo un prototipo del sistema. La fase de exploración puede tomar desde algunas semanas hasta meses, dependiendo de cuan familiar es la tecnología para los programadores [17].
- **Planeación:** En esta fase se establece el orden de prioridad para las historias de usuario y un acuerdo entre el cliente y el equipo de desarrollo sobre el contenido del primer lanzamiento del sistema. Los programadores estiman la cantidad de esfuerzo requerido por cada historia de usuario y se llega a un acuerdo en el cronograma del proyecto. Esta fase toma usualmente varios días [17].
- **Iteraciones a liberar:** Esta fase incluye varias iteraciones del sistema antes del primer lanzamiento. El cronograma establecido en la fase de planeación se separa en un número de iteraciones que tomarán de una a cuatro semanas de implementar. La primera iteración crea un sistema con la arquitectura de todo el sistema. Esto se logra seleccionando las historias que fuerzan a construir la estructura de todo el sistema. El cliente decide las historias que serán seleccionadas para cada iteración. Las pruebas funcionales diseñadas por el cliente deben correr al final de cada iteración. Al final de la última iteración el sistema está listo para producción [17].
- **Producción:** La fase de producción requiere una serie extra de pruebas y el chequeo del funcionamiento del sistema antes de que pueda ser liberado al cliente. Además, en esta fase aún se pueden realizar nuevas decisiones y cambios si se incluyen en el lanzamiento actual. Durante esta fase las iteraciones se deben apresurar desde tres a una semana. Las ideas y sugerencias pospuestas son documentadas para ser implementadas durante la fase de mantenimiento [17].
- **Mantenimiento:** Después del primer lanzamiento de producción para el uso por el cliente, el proyecto XP debe mantener el sistema corriendo y además producir nuevas iteraciones. Para lograr esto, la fase de mantenimiento requiere un esfuerzo en las tareas de soporte del cliente. De esa manera, la velocidad de desarrollo se puede desacelerar después que el sistema entre en producción. Esta fase puede requerir incorporar nuevas personas al equipo de desarrollo y realizar cambios en la estructura actual del mismo [17].
- **Muerte:** La fase final o muerte del proceso se encuentra cercana cuando el cliente no tiene más historias para implementar. Esto requiere que el sistema satisfaga otras necesidades del cliente como son rendimiento y confiabilidad. Este es el momento en el que la documentación del sistema es finalmente escrita y no se realizan más cambios al diseño o la arquitectura del proyecto. La muerte del proyecto puede ocurrir también cuando el sistema no entrega el resultado esperado o si se hace muy caro seguir su desarrollo [17].

En los Capítulos 2 y 3 se demostrará en detalle la implementación de la metodología XP en el desarrollo del PySigRec.

1.6 Conclusiones del capítulo

Después de realizar el análisis del marco teórico en que se desarrolla el problema a resolver, se logró un estudio de los principales métodos y algoritmos que más se utilizan hoy para la captura de señales electro-oculográficas. Además se hizo un resumen de las tendencias actuales del desarrollo de software, tanto de metodologías como herramientas de apoyo a este proceso. Finalmente se selecciona la metodología XP para el desarrollo de la solución propuesta debido a que es la que mejor se adapta al escenario planteado.

Capítulo 2. Exploración y Planificación

2.1 Introducción

En este capítulo se detallará la aplicación de las dos primeras fases de XP en la solución propuesta. El objetivo principal de este es describir las características fundamentales del sistema, exponiéndose de manera clara los requerimientos a cumplir por el mismo que dan paso a las futuras historias de usuario.

2.2 Exploración

Los requisitos del sistema representan las tareas que el sistema debe ser capaz de realizar para trabajar correctamente. Es por ello que la redacción detallada de estos, para ser utilizados como base de las futuras pruebas sea de vital importancia.

2.2.1 Descripción de requisitos

Los requisitos del sistema representan las tareas que el sistema debe ser capaz de realizar para trabajar correctamente. Es por ello que la redacción detallada de estos, para ser utilizados como base de las futuras pruebas sea de vital importancia.

2.2.1.1 Requerimientos funcionales

- R1: Manejar sección de captura de señales electro-oculográficas
    - R1.1: Crear una sección de captura
    - R1.2: Abrir una sección de captura
    - R1.3: Guardar una sección de captura
- R2: Capturar señales electro-oculográficas
    - R2.1: Leer los datos de la tarjeta de adquisición en tiempo real
    - R2.2: Escribir los datos en un fichero temporal
    - R2.3: Visualizar la señal en tiempo real

- R3: Exportar los datos
    - R3.1: Exportar los datos a formato de texto tabulado (csv)
    - R3.2: Exportar los datos a formato soportado por el EogStudy (eog)

2.2.1.2 Requerimientos no funcionales

Los requerimientos no funcionales son aquellos que constituyen propiedades o cualidades que el producto debe tener. Son las características que lo hacen atractivo, usable, rápido y confiable.

1. Desarrollar el producto centrado solo en los sistemas operativos UNIX.
2. Uso de un tema de colores e iconos ligero y agradable.
3. Uso de atajos del teclado para facilitar la interacción con la aplicación.
4. Creación de manual de usuario del producto en un formato estandarizado.
5. Documentar las clases del API del núcleo de la aplicación utilizando un generador de documentación.
6. Utilizar el idioma inglés en el producto informático.
7. Desarrollar el producto informático en el lenguaje de programación Python.

2.2.2 Personas relacionadas con la aplicación

Las personas relacionadas con el sistema son aquellas que obtienen algún resultado útil del mismo. Las aplicaciones de procesamiento y captura de señales biomédicas se orientan generalmente a especialistas de la rama. Como aplicación de escritorio esta no contiene restricciones de acceso a determinadas funcionalidades para un grupo específico de usuarios.

En el caso del PySigRec la persona relacionada es el **especialista**. Esta persona debe tener los conocimientos suficientes sobre el proceso de captura de señales electro-oculográficas para ser capaz de explotar este de forma correcta. Este rol es tomado generalmente por personal médico o de la universidad que conduce alguna investigación sobre el tema.

2.2.3 Historias de Usuario

Historia de Usuario	
No.: 1	**Nombre:** Creación de prototipo y estructura del proyecto
Usuario: Todos	
Prioridad en el Negocio: Alta	**Nivel de Complejidad:** Baja

Estimación: 2 semanas	**Iteración Asignada:** 1
Descripción: El *operario* debe ser capaz de ejecutar la aplicación y interactuar con un prototipo no funcional de la aplicación. **Información Adicional (Observaciones):** En esta historia de usuario se organizará la estructura del proyecto de forma modular y se elaborarán las interfaces de usuario con las que contará la aplicación. Estas interfaces no funcionales se enlazarán de forma estática para que el usuario tenga una idea de cómo sería el producto final.	

Tabla 4: Historia de Usuario #1 "Creación de prototipo y estructura del proyecto"

Historia de Usuario	
No.: 2	**Nombre:** Gestionar interfaces Comedi
Usuario: Todos	
Prioridad en el Negocio: Media	**Nivel de Complejidad:** Alta
Estimación: 3 semanas y 1 día	**Iteración Asignada:** 1
Descripción: El *operario* debe ser capaz de seleccionar la interfaz Comedi de su conveniencia según las capacidades detectadas automáticamente de esta o estas. **Información Adicional (Observaciones):** En esta historia de usuario se crearán las clases e interfaces necesarias para detectar de forma automática los dispositivos Comedi conectados a la PC. Además, de estos dispositivos se detectarán las capacidades relevantes para la captura de señales electro-oculográficas. Se implementarán las interfaces de usuario de selección de interfaces Comedi.	

Tabla 5: Historia de Usuario #2 "Gestionar interfaces Comedi"

Historia de Usuario	
No.: 3	**Nombre:** Gestionar proceso de captura de datos
Usuario: Todos	
Prioridad en el Negocio: Alta	**Nivel de Complejidad:** Alta
Estimación: 2 semanas y 2 días	**Iteración Asignada:** 1
Descripción: El *operario* debe ser capaz de crear, visualizar y almacenar una captura de datos a través de una interfaz Comedi seleccionada.	

Información Adicional (Observaciones): En esta historia de usuario se implementarán las clases e interfaces necesarias para la captura de datos, así como un componente que garantice su visualización en tiempo real. Además deberá permitir almacenar esta captura en tiempo real en un archivo temporal.

Tabla 6: Historia de Usuario #3 "Gestionar proceso de captura de datos"

Historia de Usuario	
No.: 4	**Nombre:** Exportar capturas a formatos externos
Usuario: Todos	
Prioridad en el Negocio: Media	**Nivel de Complejidad:** Media
Estimación: 6 días	**Iteración Asignada:** 1
Descripción: El *operario* debe ser capaz de guardar los datos de la captura en formatos soportados por terceros. **Información Adicional (Observaciones):** En esta historia de usuario se implementarán las clases e interfaces necesarias para exportar las capturas realizadas a formatos externos que permitan su post-procesamiento por parte de aplicaciones de terceros. Se deberán adicionar estos formatos a los diálogos de guardar archivos de la aplicación.	

Tabla 7: Historia de Usuario #4 "Exportar capturas a formatos externos"

Historia de Usuario	
No.: 5	**Nombre:** Empaquetamiento de la aplicación
Usuario: Todos	
Prioridad en el Negocio: Baja	**Nivel de Complejidad:** Media
Estimación: 3 días	**Iteración Asignada:** 1
Descripción: El *operario* debe ser capaz de instalar la aplicación de forma sencilla utilizando alguno de los gestores de paquetes de distribuciones GNU/Linux tipo Debian. **Información Adicional (Observaciones):** En esta historia de usuario se creará un paquete de Debian (.deb) para las distribuciones de este tipo más comunes (Debian 6 y Ubuntu 12.04).	

Tabla 8: Historia de Usuario #5 "Empaquetamiento de la aplicación"

Es importante señalar que las estimaciones del tiempo de duración de cada historia se utilizó como unidad la

semana laborable que consiste en 5 días, cada uno con 8 horas laborables. Cumpliendo con la práctica de XP que dice que como máximo se deben trabajar 40 horas a la semana.

2.3 Planificación

Cada iteración es planificada descomponiendo las historias de usuario que la componen en tareas. Las tareas son programadas y asignadas teniendo en cuenta la opinión de los programadores, así ellos pueden elegir cuales pueden realizar, al final se balance la cantidad asignada a cada uno.

2.3.1 Iteraciones

Para la selección del trabajo de cada iteración se tuvo en cuenta que este no tuviera más de 4 semana de desarrollo, siguiendo las prácticas básicas de XP. En la Tabla 9 se muestran la distribución de las Historias de Usuario por cada iteración.

Iteraciones	Orden de las historias de usuario a implementar	Cantidad de tiempo de trabajo
1	1. Creación de prototipo y estructura del proyecto 2. Gestionar interfaces Comedi	5 semanas y 1 día
2	3. Gestionar proceso de captura de datos 4. Exportar capturas a formatos externos	3 semanas y 3 días
3	5. Empaquetamiento de la aplicación	3 días

Tabla 9: Distribución de las historias de usuario por iteración

2.3.2 Plan de entregas

El plan de entregas es el compromiso final del equipo de trabajo con los clientes. Es una cuestión de vital importancia para el negocio entre ambas partes, ya que la entrega tardía o temprana de la solución, repercute notablemente en la economía y moral de todos los involucrados.

La estimación es uno de los temas más complicados del desarrollo de un proyecto que utiliza la metodología XP, y es por ello que resulta de vital importancia tener bien claros los requerimientos del cliente, el estilo de trabajo del equipo de desarrollo y el tiempo con que dispone el cliente para tener en sus manos la solución.

El cronograma de entregas del PySigRec se expone en la Tabla 10.

Entregable	Fin 1ra Iteración	Fin 2da Iteración	Fin 3ra Iteración
PySigRec	Versión 0.1 (17 de abril del 2012)	Versión 0.2 (14 de mayo del 2012)	Versión 0.9 (17 de mayo del 2012)

Tabla 10: Cronograma de entregas

2.4 Conclusiones del capítulo

En este capítulo se completaron de forma satisfactoria las dos primeras fases del ciclo de vida de la solución propuesta: los artefactos de las historias de usuario, el plan de iteraciones y el de entregas, los cuáles fueron detallados de forma clara y concisa.

Capítulo 3. Implementación y Pruebas

3.1 Introducción

La metodología seleccionada permite que el equipo de desarrollo decida la cantidad de artefactos a utilizar como por ejemplo los diagramas UML, dependiendo de la capacidad del equipo para comunicarse para que el proceso de desarrollo sea más fácil y comprensible. Este capítulo hace alusión al diseño del sistema, los diagramas utilizados, las tareas generadas por cada historia de usuario, el proceso de pruebas utilizado y finalmente la valoración de la propuesta.

3.2 Diseño de la aplicación

El diseño de la aplicación es el proceso donde cliente y usuario deciden como debe verse el producto final en relación a sus funcionalidades y también como estará compuesto. Esta etapa es vital en el desarrollo de un producto informático ya de ella dependen aspectos como la legibilidad, calidad, reutilización entre otros no menos importantes.

3.2.1 Arquitectura

La arquitectura propuesta para el PySigRec presenta dos tipos principales de componentes: las bibliotecas y las aplicaciones. Las primeras permiten encapsular las funciones y abstracciones comunes de la aplicación divididos en subsistemas. Las aplicaciones son la base del producto ya que estas soportan la funcionalidades y además poseen los medios para comunicarse con las bibliotecas obtener de ellas lo necesario.

Es importante señalar que en estos componentes se hace uso de las bibliotecas Qt lo que hace que la base de esta arquitectura recaiga en las posibilidades de extensión de esta. Para organizar las clases se utilizan paquetes para agruparlas. En la figura 3 se muestra la organización jerárquica de estos. Para mejorar la comprensión de la arquitectura y la jerarquía entre clases se debe recurrir al diagrama UML de clases (véase Figura 4).

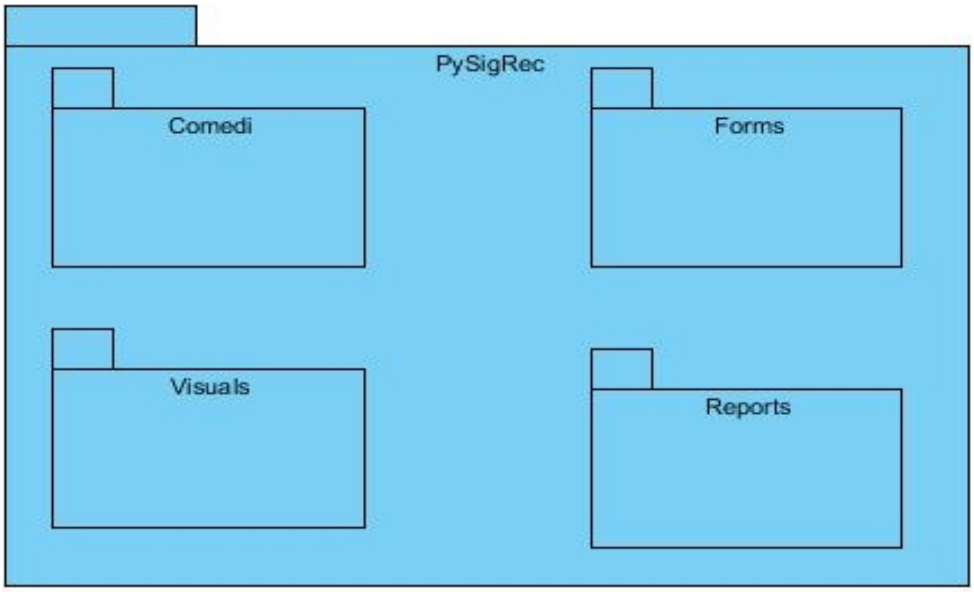

Figura 3: Diagrama de paquetes.

3.3 Implementación

XP propone comenzar la implementación partiendo de una arquitectura lo más flexible posible para evitar grandes cambios en las próximas iteraciones y en los cambios que generalmente el cliente propone. Ya que la solución que se propone posee un gran componente técnico, es necesario desde un inicio tener bien definida la arquitectura.

Trazada la misma, se procede a la primera iteración.

3.3.1 Iteración 1

La primera iteración a ejecutar tiene como objetivo principal crear un prototipo básico de la aplicación, dando cumplimiento uno de los preceptos de agilidad de XP. También, implementar las estructuras y artefactos que serán utilizados para gestionar todo lo referente a las interfaces Comedi. Esta iteración comprende las dos primeras historias de usuarios.

Para ello se trazaron las siguientes tares:

1. **Tarea #1**: Crear estructura del proyecto.

2. **Tarea #2**: Detectar capacidades de la tarjeta.

3. **Tarea #3**: Diseñar e implementar todos los aspectos referentes a la gestión de las tarjetas de adquisición de datos.

4. **Tarea #4**: Crear interfaz de usuario para la selección de parámetros de captura.

Tarea	
Número de Tarea: 1	**Número de HU:** 1
Nombre de la Tarea: Crear estructura del proyecto	
Tipo de Tarea: Configuración	**Estimación:** 1 días
Fecha de Inicio: 13 de Marzo de 2012	**Fecha de Fin: 13** de Marzo de 2012

Programador Responsable: Andrés Henriquez Pérez
Descripción: Crear un proyecto de PyDev en el Eclipse con la siguiente organización:
PySigRec
• artwork [Destino de los elementos de diseño gráfico]
• src
  ○ Forms [Clases Visuales]
  ▪ UserInterfaces
  ○ Capture Session [Clases que manipulan los datos de captura]
  ○ Reports [Clases que gestionan los reportes]
  ○ Main [Clase principal que está compuesta por las antes mencionadas]
• tests	

*Tabla 11: **Tarea #1**: Crear estructura del proyecto.*

Tarea	
Número de Tarea: 2	**Número de HU:** 1
Nombre de la Tarea: Detectar capacidades de la tarjeta	
Tipo de Tarea: Desarrollo	**Estimación:** 6 días
Fecha de Inicio: 14 de Marzo de 2012	**Fecha de Fin:** 21 de Marzo de 2012
Programador Responsable: Andrés Henriquez Pérez	

Descripción: Detectar los sub-dispositivos, rangos de captura, tipo de adquisición o referencia a utilizar, canales, así como las demás opciones soportadas por la tarjeta. Crear las estructuras que soportaran estas capacidades.

*Tabla 12: **Tarea #2**: Detectar capacidades de la tarjeta.*

Tarea	
Número de Tarea: 3	**Número de HU:** 2
Nombre de la Tarea: Diseñar e implementar todos los aspectos referentes a la gestión de las tarjetas de adquisición de datos	
Tipo de Tarea: Desarrollo	**Estimación:** 8 días
Fecha de Inicio: 22 de Marzo de 2012	**Fecha de Fin:** 2 de Abril de 2012
Programador Responsable: Andrés Henriquez Pérez	
Descripción: Crear los mecanismos que permitan conectar las interfaces de usuario con las capacidades almacenadas de la tarjeta y que a su vez le den instrucciones a la tarjeta para prepararla para la captura.	

*Tabla 13: **Tarea #3**:* Diseñar e implementar todos los aspectos referentes a la gestión de las tarjetas de adquisición de datos.

Tarea	
Número de Tarea: 4	**Número de HU:** 2
Nombre de la Tarea: Crear interfaz de usuario para la selección de parámetros de captura	
Tipo de Tarea: Desarrollo	**Estimación:** 2 días
Fecha de Inicio: 3 de Abril de 2012	**Fecha de Fin:** 4 de Abril de 2012
Programador Responsable: Andrés Henriquez Pérez	
Descripción: Crear interfaz de usuario que obtenga los datos necesarios para configurar un comando que se ejecutará posteriormente en la aplicación para obtener los datos de la captura.	

Tabla 14: **Tarea #4**: Crear interfaz de usuario para la selección de parámetros de captura.

3.3.2 Iteración 2

Esta iteración está dedicada a crear una aplicación más funcional, que permita utilizar los artefactos de captura y además guardar los datos en ficheros con formatos específicos. Para lograrlo se deben conectar las interfaces de

usuario con las opciones y funcionalidades de la aplicación.

Para ello se trazaron las siguientes tares:

1. **Tarea #5**: Captura en tiempo real de los datos desde la tarjeta de adquisición.
2. **Tarea #6**: Almacenamiento en tiempo real de la señal que se esté capturando.
3. **Tarea #7**: Captura en tiempo real de los datos desde la tarjeta de adquisición.
4. **Tarea #8**: Almacenamiento en tiempo real de la señal que se esté capturando.
5. **Tarea #9**: Representación en tiempo real de la señal que se esté capturando.
6. **Tarea #10**: Enlace de las interfaces de usuario a utilizar con las funcionalidades que estas deben brindar.
7. **Tarea #11**: Exportar las capturas realizadas a archivos de datos del EogStudio (.eos).
8. **Tarea #12**: Exportar las capturas realizadas a archivos de datos de texto tabulado (.csv).

Tarea	
Número de Tarea: 5	**Número de HU:** 2
Nombre de la Tarea: Captura en tiempo real de los datos desde la tarjeta de adquisición	
Tipo de Tarea: Desarrollo	**Estimación:** 7 días
Fecha de Inicio: 5 de Abril de 2012	**Fecha de Fin:** 13 de Abril de 2012
Programador Responsable: Andrés Henriquez Pérez	
Descripción: Ejecutar comando previamente configurado para capturar los datos utilizando todas los opciones posibles de la tarjeta. Esto asegura un cumplimiento estricto de la ejecución, asignándole tareas específicas al dispositivo y al microprocesador de la computadora.	

Tabla 15: **Tarea #5**: Captura en tiempo real de los datos desde la tarjeta de adquisición.

Tarea	
Número de Tarea: 6	**Número de HU:** 2
Nombre de la Tarea: Almacenamiento en tiempo real de la señal que se esté capturando	
Tipo de Tarea: Desarrollo	**Estimación:** 2 días
Fecha de Inicio: 16 de Abril de 2012	**Fecha de Fin:** 17 de Abril de 2012
Programador Responsable: Andrés Henriquez Pérez	
Descripción: Crear un sistema de ficheros para almacenar los datos obtenidos en tiempo real y limpiar el buffer, para evitar una sobrecarga del sistema.	

Tabla 16: **Tarea #6**: Almacenamiento en tiempo real de la señal que se esté capturando.

Tarea	
Número de Tarea: 7	**Número de HU:** 3
Nombre de la Tarea: Captura en tiempo real de los datos desde la tarjeta de adquisición	
Tipo de Tarea: Desarrollo	**Estimación:** 7 días
Fecha de Inicio: 5 de Abril de 2012	**Fecha de Fin:** 13 de Abril de 2012
Programador Responsable: Andrés Henriquez Pérez	
Descripción: Ejecutar comando previamente configurado para capturar los datos utilizando todas los opciones posibles de la tarjeta. Esto asegura un cumplimiento estricto de la ejecución, asignándole tareas específicas al dispositivo y al microprocesador de la computadora.	

Tabla 17**: Tarea #7**: Captura en tiempo real de los datos desde la tarjeta de adquisición.

Tarea	
Número de Tarea: 8	**Número de HU:** 3
Nombre de la Tarea: Almacenamiento en tiempo real de la señal que se esté capturando	
Tipo de Tarea: Desarrollo	**Estimación:** 2 días
Fecha de Inicio: 16 de Abril de 2012	**Fecha de Fin:** 17 de Abril de 2012
Programador Responsable: Andrés Henriquez Pérez	
Descripción: Crear un sistema de ficheros para almacenar los datos obtenidos en tiempo real y limpiar el buffer, para evitar una sobrecarga del sistema. Este debe ser un fichero de uso interno, que luego se utilizará	

para cargar secciones de capturas y exportarlas en los formatos implementados, también pueden usarse para representar la señal en un componente de visualización y ejecutar algunas acciones como aumentarla, escalarla y otras.

Tabla 18: **Tarea #8**: Almacenamiento en tiempo real de la señal que se esté capturando.

Tarea	
Número de Tarea: 9	**Número de HU:** 3
Nombre de la Tarea: Representación en tiempo real de la señal que se esté capturando	
Tipo de Tarea: Desarrollo	**Estimación:** 7 días
Fecha de Inicio: 18 de Abril de 2012	**Fecha de Fin:** 26 de Abril de 2012
Programador Responsable: Andrés Henriquez Pérez	
Descripción: Crear las clases encargadas de dibujar la señal y conectarlas con la señal obtenida, asegurándose que los parámetros de la frontera del área de dibujo estén bien seleccionados para evitar corrimiento en la representación de la señal.	

Tabla 19: **Tarea #9**: Representación en tiempo real de la señal que se esté capturando.

Tarea	
Número de Tarea: 10	**Número de HU:** 3
Nombre de la Tarea: Enlace de las interfaces de usuario a utilizar con las funcionalidades que estas deben brindar	
Tipo de Tarea: Desarrollo	**Estimación:** 3 días
Fecha de Inicio: 27 de Abril de 2012	**Fecha de Fin:** 2 de Mayo de 2012
Programador Responsable: Andrés Henriquez Pérez	
Descripción: Deben quedar establecidos los enlaces de las interfaces de usuario, para brindar las funcionalidades soportadas por el hardware.	

Tabla 20: **Tarea #10**: Enlace de las interfaces de usuario a utilizar con las funcionalidades que estas deben brindar.

Tarea	
Número de Tarea: 11	**Número de HU:** 4
Nombre de la Tarea: Exportar las capturas realizadas a archivos de datos del EogStudio (.eos)	

Tipo de Tarea: Desarrollo	**Estimación:** 4 días
Fecha de Inicio: 7 de Mayo de 2012	**Fecha de Fin:** 10 de Mayo de 2012
Programador Responsable: Andrés Henriquez Pérez	
Descripción: Exportar los datos adquiridos en un fichero que cumpla con los estándares del Eogstudio.	

Tabla21: **Tarea #11**: Exportar las capturas realizadas a archivos de datos del EogStudio (.eos).

Tarea	
Número de Tarea: 12	**Número de HU:** 4
Nombre de la Tarea: Exportar las capturas realizadas a archivos de datos de texto tabulado (.csv)	
Tipo de Tarea: Desarrollo	**Estimación:** 2 días
Fecha de Inicio: 11 de Mayo de 2012	**Fecha de Fin:** 14 de Mayo de 2012
Programador Responsable: Andrés Henriquez Pérez	
Descripción: Exportar los datos adquiridos en un fichero que cumpla con los estándares de texto tabulado para que puedan ser procesados en otros softwares como el Excel o el módulo Calc del OpenOffice.	

Tabla 22: **Tarea #12**: Exportar las capturas realizadas a archivos de datos de texto tabulado (.csv).

3.3.3 Iteración 3

En esta iteración se procede a culminar el proceso de creación de software, se elabora un repositorio git para asignarle a cada opción que lo requiera un ícono para ayudar al usuario a familiarizarse con el mismo, y además porque esto ayuda en la usabilidad del producto. También se crea un paquete de instalación para las distribuciones derivadas de Debian (.dev).

Para ello se trazaron las siguientes tares:

1. **Tarea #13**: Crear repositorio git para el proyecto.

2. **Tarea #14**: Empaquetamiento de la aplicación.

Tarea	
Número de Tarea: 13	**Número de HU:** 5
Nombre de la Tarea: Crear repositorio git para el proyecto	

Tipo de Tarea: Desarrollo	**Estimación:** 1 día
Fecha de Inicio: 15 de Mayo de 2012	**Fecha de Fin:** 15 de Mayo de 2012
Programador Responsable: Andrés Henriquez Pérez	
Descripción: Crear el repositorio de las imágenes usadas como íconos y empaquetarlas dentro de la aplicación. Asignar las imágenes correspondientes a cada menú o acción del proyecto.	

Tabla 23: ***Tarea #13****:* Crear repositorio git para el proyecto.

Tarea	
Número de Tarea: 14	**Número de HU:** 5
Nombre de la Tarea: Empaquetamiento de la aplicación	
Tipo de Tarea:	**Estimación:** 2 días
Fecha de Inicio: 16 de Mayo de 2012	**Fecha de Fin:** 17 de Mayo de 2012
Programador Responsable: Andrés Henriquez Pérez	
Descripción: Se creará un paquete (.dev) para que el *usuario* sea capaz de instalar la aplicación de forma sencilla utilizando alguno de los gestores de paquetes de distribuciones GNU/Linux tipo Debian.	

Tabla 24: **Tarea #14**: Empaquetamiento de la aplicación.

3.4 Pruebas

Las pruebas son en XP muy importantes ya que son las que finalizan el proceso de implementación. Esta etapa también está diseñada para el usuario en conjunto con el programador, esta fórmula genera buenos resultados.

3.4.1 Pruebas de aceptación

Se conocen como pruebas de caja negra porque son realizadas por el cliente en conjunto con un representante del equipo. Las pruebas de aceptación tienen como objetivo principal comprobar, desde la perspectiva del usuario final, el cumplimiento de las especificaciones realizadas en las historias de usuario.

A continuación, aparecen las pruebas de aceptación realizadas a la solución propuesta:

Caso de Prueba de Aceptación	
Código: HU2_P1	**Historia de Usuario:** 2
Nombre: Crear sesión de captura	

Descripción: Prueba la funcionalidad de crear nuevas sesiones de capturas.

Condiciones de Ejecución: Se debe contar con al menos una tarjeta de adquisición (DAQ) conectada a la computadora.

Entrada/Pasos de Ejecución: Después de ejecutarse el PySigRec se lanzará la acción de crear estudio. Al abrirse el ayudante se le especificarán el nombre de la sesión, la tarjeta y el sub-dispositivo a usar y se procederá continuar hacia la segunda página de este. En la segunda página se seleccionarán los canales, rangos y referencias de la captura y se ejecutará el botón de Finalizar. Después de esto se procederá a comenzar la lectura de datos.

Resultado Esperado: Se crean y se muestran las interfaces diseñadas para mostrar la señal de los canales previamente seleccionados, además, al crearse la sesión, el usuario tendrá acceso a un área visual donde puede seleccionar y cambiar otros parámetros de la captura.

Evaluación de la Prueba: Prueba satisfactoria.

Tabla 25: Caso de prueba de aceptación HU2_P1

Caso de Prueba de Aceptación	
Código: HU3_P1	**Historia de Usuario:** 2

Nombre: Capturar datos en tiempo real.

Descripción: Prueba la funcionalidad capturar datos en tiempo real.

Condiciones de Ejecución: Se debe contar con una tarjeta de adquisición (DAQ) y para que haya más fiabilidad se puede utilizar un generador de señales, este te permite comprobar que la señal obtenida coincide con la transmitida.

Entrada/Pasos de Ejecución: Luego de haber configurado una sesión de captura, se procede a activar la acción de Comenzar.

Resultado Esperado: Deberán crearse y mostrarse los componentes visuales que pintarán las señales, un componente por señal, estos a su vez contarán con información adicional, como el número del canal y el rango aproximado en el que se mueve la señal.

Evaluación de la Prueba: Prueba satisfactoria.

Tabla 26: Caso de prueba de aceptación HU3_P1

Caso de Prueba de Aceptación	
Código: HU4_P1	**Historia de Usuario:** 4

Nombre: Exportar datos en formato de texto tabulado (.csv).

Descripción: Prueba la funcionalidad de exportar datos a un fichero.

Condiciones de Ejecución: Se debe contar con una captura realizada, que no esté en ejecución.

Entrada/Pasos de Ejecución: Luego de obtener los valores de la captura se procede a accionar el evento de Guardar o Guardar como, el primero guarda en una dirección previamente seleccionada y el segundo necesita la especificación de la dirección para grabar los datos. En el ayudante se deberá especificar el formato (.csv).
Resultado Esperado: Deberá crearse un fichero con el formato de texto tabulado que puede ser utilizado por otras aplicaciones como Open Writter o Microsoft Excel. Este fichero contendrá los datos de la sesión, así como los valores obtenidos por cada canal de captura.
Evaluación de la Prueba: Prueba satisfactoria.

Tabla 27: Caso de prueba de aceptación HU4_P1

Caso de Prueba de Aceptación	
Código: HU4_P2	**Historia de Usuario:** 4
Nombre: Exportar datos en formato de EogStudy (.eog).	
Descripción: Prueba la funcionalidad de exportar datos a un fichero.	
Condiciones de Ejecución: Se debe contar con una captura realizada, que no esté en ejecución.	
Entrada/Pasos de Ejecución: Luego de obtener los valores de la captura se procede a accionar el evento de Guardar o Guardar como, el primero guarda en una dirección previamente seleccionada y el segundo necesita la especificación de la dirección para grabar los datos. En el ayudante se deberá especificar el formato (.eog).	
Resultado Esperado: Deberá crearse un fichero con el formato del EogStudy que puede ser utilizado por esta aplicación para analizar la señal. Este fichero contendrá los datos de la sesión, así como los valores obtenidos por cada canal de captura.	
Evaluación de la Prueba: Prueba satisfactoria.	

Tabla 28: Caso de prueba de aceptación HU4_P2

3.6 Conclusiones del capítulo

En el presente capítulo se abordaron todos los temas referentes a el diseño, la implementación y la estrategia de pruebas de la solución. Se presentó el diseño de la arquitectura y las tareas que se llevaron a cabo para construirla.

Cada tarea fue cumplida en fecha y finalmente se logró el producto en el plazo de tiempo planificado. La efectividad del desarrollo dirigido por pruebas y la aplicación de las pruebas de aceptación demuestran ser muy altas. Ambas juegan un papel fundamental en el proceso de desarrollo de software con una metodología ágil, aun cuando el equipo de desarrollo es de un solo integrante.

Conclusiones Generales

El software elaborado se ha utilizado en la facultad de Informática y Matemática de la Universidad de Holguín “Oscar Lucero Moya”, para capturar señales electro-oculográficas. Este mejora el proceso antes mencionado y que es vital en investigaciones médicas que se llevan a cabo en la facultad. Con él se planea la obtención de movimientos sacádicos de pacientes con ataxia para investigaciones y estudios clínicos que puedan arrojar tratamientos.

Las interfaces de usuario fueron escogidas teniendo en cuenta las funcionalidades del software con el objetivo de facilitar su uso. Así mismo se realizó un profundo análisis para determinar e implementar la mejor manera de realizar la adquisición y los componentes gráficos que intervendrían en la representación de la señal, también el método para almacenar la señal en tiempo real logrando evitar el uso excesivo de recursos como la memoria.

La elección del lenguaje de programación Python, reconocido por su facilidad de manejo, poco código y por sus potentes bibliotecas matemáticas, y del conjunto de bibliotecas Qt, se realizaron a partir de la premisa de que el resultado final pueda ser liberado bajo alguna de las licencias de código abierto existentes. Además debe destacarse que Qt brinda la posibilidad de adaptar la interfaz gráfica a la cultura del sistema operativo. Estas consideraciones son de gran importancia, teniendo en cuenta el interés del estado cubano de estimular la elaboración de herramientas de código abierto que contribuyan a romper la dependencia tecnológica con las empresas productoras de software privativo.

El uso de la metodología Xp con un enfoque ágil, orientada a equipos pequeños, en los que se espera la disminución de los volúmenes de documentación y formalización necesarios, apoyada en una interacción constante entre el diseñador-programador y el especialista, propició la obtención de un diseño y producción en consonancia con los objetivos de este proyecto.

A partir de los aspectos que se han señalado en los párrafos precedentes, consideramos que los objetivos planteados al acometer la elaboración de este software fueron cumplidos. Esta herramienta de captura y registro de señales electro-oculográficas, ha sido concebida y desarrollada teniendo en cuenta el estado actual de las aplicaciones y herramientas empleadas en este tema, y utilizando las tecnologías y herramientas de elaboración de aplicaciones informáticas de uso corriente internacionalmente, por lo que responde a las expectativas y necesidades de los especialistas del análisis de señales de la Universidad, y de otras instituciones donde pudiese ser utilizada.

Recomendaciones

La solución desarrollada en el presente trabajo ha permitido llegar a una serie conclusiones muy importantes para la continuidad del desarrollo de la misma, y, a su vez, sirven de experiencia para proyectos que persiguen objetivos similares, hasta cierto punto. Por lo cual se proponen las siguientes recomendaciones:

- Desarrollar un módulo que permita escribir a través de la tarjeta de adquisición, y hacer un puente, es decir que la señal recepcionada pueda ser emitida a la vez.

- Implementar módulo que permita utilizar las opciones de calibración de las tarjetas, para explotar todas las capacidades de las mismas.

- Añadir soporte para editar las señales y realizar operaciones gráficas para visualizarlas del modo que el usuario decida.

- Extender la aplicación de este trabajo a otras instituciones que puedan explotar sus funcionalidades.

- Mostrar la señal en una gráfica de frecuencia.

Bibliografía

[1] L. Velázquez, *Ataxia espino cerebelosa tipo 2. Principales aspectos neurofisiológicos en el diagnóstico, pronóstico y evaluación de la enfermedad*, Ediciones Holguín. Holguín: , 2006.

[2] R. Rodríguez Echeverría, A. Prieto Ramos, and E. Sosa Sánchez, *Programación Orientada a Objetos*. .

[3] H. Zárate Rea, *Paradigmas de Programación*. 2008.

[4] D. Nourie, *Getting Started with an Integrated Development Enviroment (IDE)*. 2005.

[5] D. . Kuhn, *Selecting and effectively using a computer aided software engineering tool*. Pittsburgh, PA (U.S): , 1989.

[6] P. Loucopoulos and V. Karakostas, *System Requirements Engineering*. 1989.

[7] *Unified Modeling Language Specification. Version 1.4.2*. .

[8] *Visual Paradigm for UML User's Guide*. 2007.

[9] R. Burback, *Software Engineering Methodology: The Watersluice*. 1998.

[10] M. . Awad, *A Comparison betwen Agile and Tradicional Software Development Methodologies*. The University of Western Australia: , 2005.

[11] *12 Principios del Manifiesto Ágil*. 2001.

[12] M. Fowler, *The New Methodology*. .

[13] OTS Solutions, *Agile Methodology. Changing ways of software development...* 2010.

[14] K. Beck, "Embracing Change With Extreme Programing," vol. 32, no. IEEE Computer, 1999.

[15] K. Beck, *Extreme Programing Explained*. Embrace Change, 1999.

[16] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, *Agile software development methods. Reviews and Analysis*. 2002.

[17] K. Kchwaber and M. Beedle, *Agile Software Development With Scrum*. Upper Saddle River, NJ: Prentice-Hall, 2002.

[18] F. Asteasuain y B. E. Contreras, "Programación Orientada a Aspectos. Análisis del paradigma", 2002.

[19] S. . Palmer and J. . Felsing, *A Practical Guide to Feature-Driven Development*. Upper Saddle River, NJ: Prentice-Hall, 2002.

[20] K. Beck and M. Fowler, *Planning Extreme Programing*. Addison Wesley, 2000.

[21] C. Patel and M. Ramachandran, "Story Card Based Agile Software Development," no. International Journal of Hybrid Information Technology, Abr-2009.

Glosario de Términos

.NET: Plataforma de la empresa Microsoft para el desarrollo de software con énfasis en transparencia de redes, con independencia de tecnología de hardware y que permite un rápido desarrollo de aplicaciones.

AJAX: Es el acrónimo en inglés de **A**synchronous **J**avScript **A**nd **X**ML (JavaScript asíncrono y XML), la cual es una técnica de desarrollo web para crear aplicaciones interactivas.

Alianza Ágil: Organización sin ánimos de lucro con membresía global comprometida con el avance de los principios y prácticas del desarrollo ágil.

API: Interfaz de Programación de Aplicaciones por sus siglas en inglés, la cual no es más que el conjunto de clases, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Apple Inc.: Empresa multinacional estadounidense que diseña y produce equipos electrónicos y software. Muy famosa por su serie de computadoras Mac y los dispositivos empotrados IPhone e IPad.

Bytecode: Código binario intermedio usado por determinadas plataformas de desarrollo para acelerar la ejecución de los programas y crear un estándar multiplataforma.

Error de Regresión: Son aquellos que se crean al añadir nuevo código y este hace que características que otro código que funcionaba correctamente falle.

Framework: En el desarrollo de software, es una estructura de soporte definida mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, librerías de código y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

JIT: Compilación al vuelo por sus siglas en inglés (Just In Time Compilation). El C# al ser un lenguaje perteneciente a la familia de la plataforma .NET de Microsoft posee este sistema de compilación.

Plataforma Cruzada: Las aplicaciones y librerías de plataforma cruzada son aquellas que tienen una interfaz común para cada plataforma pero se compilan de forma nativa para cada una de estas, en

contraste con soluciones multiplataforma que usualmente requieren de interpretar un código intermedio donde se sacrifica el rendimiento.

QML: Lenguaje de programación declarativo creado por Nokia muy útil para la creación de aplicaciones en dispositivos móviles.

Widget: Es un componente de interfaz gráfica de usuario. Ejemplo: Botones, Etiquetas, Listas, Listas desplegables, etc.

Widget Toolkit: Son bibliotecas que contienen un conjunto de widgets o componentes para construir interfaces gráficas de usuario.

Anexos

Diagrama de Clases UML

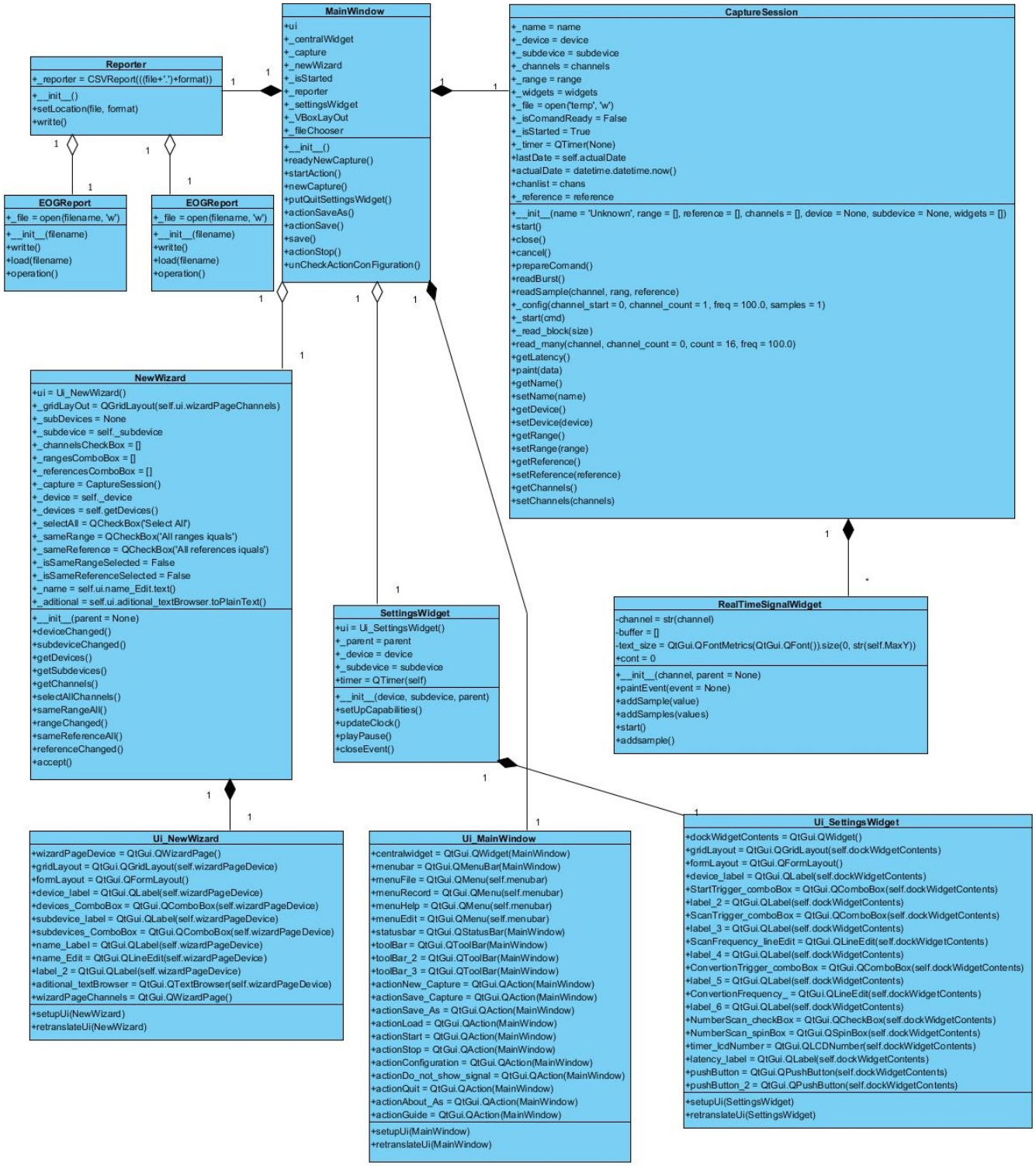

Figura 4: Diagrama de clases UML