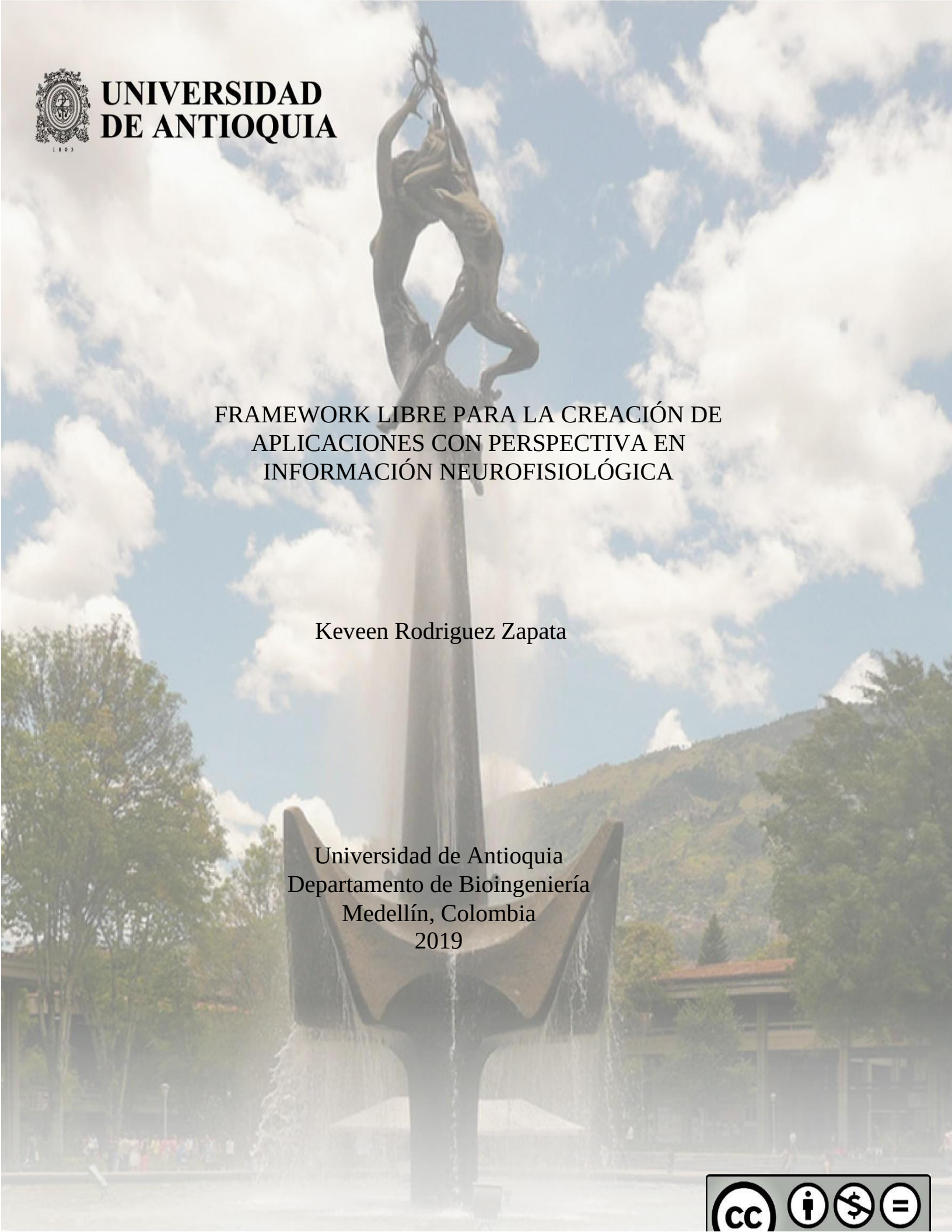

UNIVERSIDAD DE ANTIOQUIA

FRAMEWORK LIBRE PARA LA CREACIÓN DE APLICACIONES CON PERSPECTIVA EN INFORMACIÓN NEUROFISIOLÓGICA

Keveen Rodriguez Zapata

Universidad de Antioquia
Departamento de Bioingeniería
Medellín, Colombia
2019

FRAMEWORK LIBRE PARA LA CREACIÓN DE APLICACIONES CON PERSPECTIVA EN INFORMACIÓN NEUROFISIOLÓGICA

Keveen Rodríguez Zapata

Trabajo de grado presentado como requisito parcial para optar al título de:
Bioingeniero

Asesor:
John Fredy Ochoa
PhD. Ingeniería Electrónica y Computación

Línea de Investigación:
Neuroingeniería
Grupo de Investigación:
Grupo Neuropsicología y Conducta

Universidad de Antioquia
Departamento de Bioingeniería
Ciudad, Colombia
2019

Tabla de Contenido

Tabla de Figuras

Índice de Tablas

Resumen

La información acerca del comportamiento cerebral a nivel fisiológico se ha ido registrando de diferentes formas, la electroencefalografía ha sido una de las herramientas de modalidad biométrica permitiendo la constitución de índices directos e indirectos de la actividad neuronal. En el contexto de las herramientas disponibles actuales, estas han permitido la creación de prototipos de alto rendimiento, que se desean llevar al usuario final. Las metodologías adoptadas para el desarrollo de aplicaciones finales de usuario con este tipo de herramientas se ha visto limitada, obteniendo una desconexión entre las pautas de interfaz humana y los diferentes procesos realizados con herramientas de procesamiento y modalidades biométricas neurofisiológicas. El objetivo es converger las herramientas de procesamiento y análisis con un entorno de interfaz multiplataforma, cohesivo y convergente, mediante el diseño y desarrollo de un framework que permita la creación de aplicaciones con perspectiva en información neurofisiológica siguiendo las pautas de interfaz humana, logrando ser llevadas fácilmente a aplicaciones de usuario final, y la integración de nuevas herramientas. El proyecto de desarrollo incluyó la propuesta de una metodología de desarrollo en neuroingeniería, la especificación de requerimientos, el diseño del *Human Interface Guideline*, la implementación e integración. La arquitectura de software modular facilita la colaboración futura, así como la implementación de algoritmos adicionales.

Palabras clave: Framework frontend, neuroingeniería, metodología híbrida de desarrollo, creación de aplicaciones, HIG

Abstract

Information about brain behavior at the physiological level has been recorded in different ways, electroencephalography has been one of the biometric modality tools allowing the constitution of direct and indirect indices of neuronal activity. In the context of the current tools available, these have allowed the creation of high performance prototypes, which are intended to be taken to the end user. The methodologies adopted for the development of final user applications with this type of tools have been limited, obtaining a disconnection between human interface guidelines and the different processes performed with processing tools and neurophysiological biometric modalities. The objective is to converge the processing and analysis tools with a multiplatform, cohesive and convergent interface environment, through the design and development of a framework that allows the creation of applications with a perspective on neurophysiological information following the human interface guidelines, managing to be carried easily to end user applications, and the integration of new tools. The development project included the proposal of a neuroengineering development methodology, the specification of requirements, the design of the Human Interface

Guideline, the implementation and integration. The modular software architecture facilitates future collaboration, as well as the implementation of additional algorithms.

Keywords: Framework frontend, neuroengineering, hybrid development methodology, application creation, HIG

1. Introducción

El cerebro humano es un órgano complejo caracterizado por patrones heterogéneos de conexiones estructurales y funcionales cuya función depende de las interacciones dinámicas entre regiones distribuidas espacial y temporalmente que apoyan la rica cognición. La importancia de entender las propiedades estructurales y funcionales del sistema nervioso es integral en los estudios de las neurociencias cognitivas en la cual tiene como objetivo general la explicación de los procesos mentales y del comportamiento, en regiones relevantes del cerebro y en el resto del sistema nervioso central [1]□.

La evidencia emergente enfatiza en la importancia de la medición de respuestas de comportamiento como un medio para inferir lo que sucede en el sistema nervioso [1], [2]□. Las nuevas técnicas de imágenes no invasivas ahora permiten que estos patrones se puedan mapear de forma cuidadosa y completa, evolucionando a lo largo del tiempo y a medida de sus aplicaciones, jugando un papel importante en la configuración y evolución del avance en el estudio de las funciones cerebrales [3], [4]□ . Sin embargo, sigue siendo un desafío fundamental comprender exactamente cómo la arquitectura de red es compatible con nuestros procesos cognitivos.

En el procesamiento, análisis y distribución de datos, principalmente las señales EEG, MEG y la señal BOLD presentan una multitud de enfoques posibles, que se basan en una amplia gama de técnicas de procesamiento de señales e imágenes, desarrolladas en diferentes grupos de investigación alrededor del mundo y algunos de estos dedicados a encontrar biomarcadores específicos o los comportamientos de red, y aunque la instrumentación es cada vez más común en los centros de investigación de neurociencias y centros de salud, la disponibilidad y estandarización del software para ello siguen teniendo limitaciones en un amplio espectro, especialmente en el desarrollo rápido para aplicaciones finales, sin diseño limpio ni pragmático, y delimitado en el despliegue, reduciendo el progreso y uso acelerado en estos estudios y desarrollos incluso a nivel mundial.

Actualmente se encuentran diversos softwares dedicados a este tipo de señales con herramientas de creación de prototipos de alto rendimiento y ninguno enfocado al usuario final, la tendencia y el propósito de los científicos del área a nivel mundial es llevar las herramientas al usuario final y con las cuatro libertades del software, algunas de estas herramientas no se encuentran en las cuatro libertades del software libre o sólo cumplen ciertas libertades, otorgándole tres principales daños materiales provenientes de estas restricciones, se reduce el número de usuarios que usan el programa, otros desarrolladores no pueden aprender del programa y colaborar a él y ninguno de ellos pueden adaptar o mejorar el programa [5]–[13]□.

Esta perspectiva de FOSS (del inglés Free/libre and Open Source Software) en los últimos años ha ido cambiando en la comunidad neurocientífica debido a la necesidad de ésta, incursionando y aceptándose cada vez más en la comunidad, otorgando la ventaja en el entorno de investigación la simplicidad para mantener, modificar, intercambiar y reutilizar funciones y bibliotecas, dándose

una libertad de poder conocer el funcionamiento del software y poder adaptarlo a las nuevas tecnologías e investigaciones [14], [15]□. Con este avance existen ya algunos softwares con alguna de estas características mencionadas, sin embargo, la mayoría son codependientes en la interacción con el entorno MATLAB®, y ninguna con un enfoque para el usuario final, ligándose a poseer la licencia respectiva y limitándose a un diseño específico no funcional [6]–[8], [10]□, reduciendo así las altas prestaciones de uso e inclusive a nivel comercial.

El progreso de esta perspectiva FOSS ha permitido múltiples desarrollos basados en los conceptos de polivalencia y versatilidad, separándolo en dos ámbitos de desarrollo, frontend y backend, lo cual en la comunidad neurocientífica aún no se ha tenido en cuenta este afianzamiento y apropiación de tecnología de software totalmente, viéndose evidenciado en las herramientas actualmente existentes; aunque el progreso en la comunidad ha permitido la utilización de tecnologías libres aún no se establece un HIG (Human Interface Guidelines) en el desarrollo de este tipo de software científico, obteniendo como consecutivos reducciones en la calidad y prestaciones, un aumento en los costos y tiempos de desarrollo, y mínima integración con el hardware y software de los diferentes dispositivos, viéndose principalmente afectado cuando se es escalado a nivel comercial o a altas prestaciones de usos. Los desarrolladores de este tipo de software normalmente son investigadores que desean tener un conjunto de buenas prácticas y guías/normas para conceptualizar el proceso, además de su enfoque en la parte Backend y el esquema de desarrollo en el trabajo colaborativo y ágil, obteniendo el mejor resultado Human Interface Guideline posible de sus proyectos, que en línea de tiempo éste permanezca mantenido, escalable y productivo en toda la comunidad neurocientífica [16]–[18]□, Los usuarios finales de este tipo de software tampoco han tenido la experiencia de tener un adecuado entorno con las dinámicas del HIG, en la cual se apoya en base con los desarrolladores, sin lograr encontrar una convergencia entre ambos, perdiendo la facilidad en el desarrollo y despliegue de estas herramientas. Los desarrolladores habituales de software comúnmente han empleado frameworks para definir estilos globales facilitando la implementación de elementos de interfaz dinámicos, soportando diferentes complementos.

Un framework es una plataforma base sobre la cual se construyen soluciones de software listas, a base de módulos de software concretos, adquiriendo un desarrollo rápido y potente, en este caso con características multiplataforma (diferentes sistemas operativos) y convergente (diferentes tamaños de pantalla), obteniendo una estructura conceptual y tecnológica de soporte definido.

En el Grupo de Investigación Neuropsicología y Conducta (GRUNECO) se encuentra actualmente en el desarrollo del proyecto “Identificación de biomarcadores pre clínicos en Enfermedad de Alzheimer a través de un seguimiento longitudinal de la actividad eléctrica cerebral en poblaciones con riesgo genético”, en la cual se tiene como uno de sus objetivos realizar una herramienta tecnológica para el desarrollo de este tipo de estudios a través de técnicas que permitan la caracterización de procesos preclínicos; se propone un framework multiplataforma y convergente como herramienta interactiva y flexible para el desarrollo de procesamiento y visualización de EEG de alta densidad, en el marco de estudio de los ritmos cerebrales. Así incorporando las

técnicas de ingeniería de software en lo que respecta a diseño y construcción de sistemas de información, permitiendo creaciones de aplicaciones cohesivas y coherentes en neuroingeniería en su lenguaje más común (python3), más rápidas con menos código y para la mayoría de los dispositivo, además con la intención de ser encontrada al frente de la comunidad neurocientífica como software en perspectiva FOSS, permitiendo la diversidad de los usuarios y los enfoques metodológicos de algunas vías de análisis como cooperación, adopción e integración en la solución de software integrada, proporcionando estandarización y reproducibilidad progresiva de las herramientas con las vías de la ingeniería de software.

2. Objetivos

2.1. Objetivo general

Desarrollar un framework libre para la creación de aplicaciones de procesamiento de información neurofisiológica basada en una metodología con enfoque en neuroingeniería.

2.2. Objetivos específicos

- Definir la metodología de desarrollo de software desde la perspectiva de trabajo en el ámbito de la neuroingeniería
- Realizar el levantamiento de información necesaria para el sistema
- Desarrollar la guía de diseño e implementación, así como la documentación requerida para el desarrollo, operación y mantenimiento de la plataforma
- Diseñar y desarrollar el HIG en base a criterios de UI/UX
- Desarrollar el conjunto de controles y herramientas con plantillas para el procesamiento y visualización de señales EEG

3. Marco teórico

3.1. Visión general en el marco de metodología de desarrollo de software

Mi objetivo en el marco de la metodología de desarrollo, por lo tanto, es comenzar a llenar el vacío de metodologías en el ambiente de la neuroingeniería, enfocada en laboratorios de investigación y academia, mediante la realización de una revisión de las metodologías tradicionales, ágiles y su convergencia. La selección de una metodología basada en la visión general de las características y enfoques de estas mismas. Teniendo en cuenta las características específicas aplicables a la investigación y desarrollo en áreas académicas y científicas, sin alejarse del contexto de los procesos de desarrollo de software, adaptando las metodologías contemporáneas de forma que se logre beneficiar los principios y prácticas de varias maneras. Después de esto, la implementación del proceso en el Grupo Neuropsicología y Conducta (GRUNECO) en su línea de neuroingeniería, de la Universidad de Antioquia, analizando la metodología seleccionada en base de evidencia anecdótica, con el objetivo de obtener diferentes perspectivas sobre las experiencias, y finalmente la observación en la práctica de ésta.

3.2. Marcos de proceso de ingeniería de software

El último medio siglo ha sido testigo de un avance vertiginoso de avances técnicos en las áreas de informática, software y tecnología de comunicaciones. Con cada avance vinieron rápidos cambios en la forma en que la sociedad trabaja y vive. El soporte de todos estos avances es el software que se ejecuta en algún tipo de computadora. En este entorno explosivo, los profesionales de software tienen el desafío de entregar una acumulación aparentemente infinita de proyectos de software, mientras se mantienen al tanto de los últimos avances.

Los marcos de proceso de ingeniería de software establece la base para un proceso completo de desarrollo de software mediante la identificación de un pequeño número de actividades de marco que son aplicables a todos los proyectos de software, independientemente de su tamaño o complejidad. Además, el marco del proceso abarca un conjunto de actividades generales que son aplicables en todo el proceso. Estos son empleados para crear y evolucionar el producto, servicio o resultado durante el ciclo de vida del proyecto. Un marco de proceso genérico para la ingeniería de software abarca cinco actividades: comunicación, planeación, modelado, construcción y despliegue. Para muchos proyectos de software, las actividades del marco se aplican de manera iterativa a medida que avanza un proyecto. Cada iteración produce un incremento de software que proporciona a los interesados un subconjunto de funciones y funciones generales del software [18]□. A medida que se produce cada incremento, el software se vuelve más y más completo asegurando así buenas prácticas de desarrollo.

Entre estos marcos de proceso existen algunas metodologías de desarrollo que proveen marcos para planear, ejecutar y gestionar el proceso de desarrollo de sistemas de software. Existen muchas metodologías, incluidas las metodologías de cascada, prototipos, predictivas, iterativas, incrementales, rápidas, estructuradas, orientadas a objetos, ágiles, híbridas, entre otras [19]□. Cada uno tiene sus virtudes, y cada uno tiene sus partidarios y críticos en las diferentes áreas en donde se desarrolle.

El éxito de estas metodologías, sea individuales o híbridas, se enfocan en cuatro partes distintas del proceso de desarrollo de software, empezando por la recopilación y levantamiento de requisitos, diseño y desarrollo de software, pruebas de los resultados y gestión general del proyecto. En ello la definición y el análisis de los requisitos formales abordaron el problema de los requisitos que estaban incompletos o que no reflejaban las necesidades del interesado. El diseño formal antes de la implementación abordó los objetivos de re utilización, consistencia de la operación y reducción del re trabajo. La detección de defectos mediante pruebas antes de llegar a los usuarios. La gestión de proyectos abordó el problema de coordinar los esfuerzos de los equipos de varios departamentos/áreas [20]□.

Estos puntos siguen un enfoque lógico para la mejora de procesos, como la mejora en la calidad de los insumos (requerimientos), la mejora de la calidad del producto (diseño y desarrollo, gestión de proyectos) y la mejora en la detección y eliminación de defectos antes del envío (pruebas). Los métodos y herramientas que se centran en estas cuatro áreas distintas del desarrollo de software han proliferado.

Para ciertos proyectos, estos esfuerzos han sido efectivos, para otros, produjeron silos en la empresa u organización. Las técnicas de gestión de proyectos de software intentan coordinar mejor las actividades de los múltiples grupos involucrados en la entrega del proyecto, pero añaden otro silo y área de responsabilidad. Las técnicas de gestión de proyectos más populares o se podría decir, las técnicas tradicionales se centran en desarrollar un plan y apegarse a ese mismo, debido a este aspecto la metodología tradicional llegó a ser conocida como pesada (*heavyweight*, el término común en inglés). Esto mejora la coordinación pero reduce la capacidad del proyecto en la su adaptación con la nueva información relacionada con los requerimientos o los detalles de la implementación, dando a las fallas producidas.

La baja proporción de éxito, producidas por las fallas en los procesos de los sistemas de desarrollo en las empresas u organizaciones proporcionaron el impulso para el desarrollo de diferentes metodologías y prácticas, emergiendo así principalmente las metodologías ágiles de desarrollo, basadas en mejoras iterativas, introducido en 1975 por la misma comunidad de software. En ello se genera una representación global del agilismo entre las diferentes metodologías no tradicionales, creándose en el 2001 el *Agile Manifesto [21]□* por 17 metodologistas de software, en la cual se formaliza el término *Agile* ya que sus métodos tenían mucho en común, significando que es tanto ligero como suficiente. Las metodologías ágiles pretenden poner más énfasis en las personas, la interacción, el software de trabajo, la colaboración con el cliente y el cambio, en lugar de en procesos, herramientas, contratos y planes.

Las metodologías ágiles ganaron popularidad y uso en la industria [22]–[24]□, aunque comprometen una mezcla de prácticas de ingeniería de software aceptadas y controvertidas. Lo más probable es que la industria del software encuentre que las características específicas del proyecto, como lo son los objetivos, el alcance, los requerimientos, los recursos, la arquitectura y el tamaño determinarán qué metodología se adapta mejor a ellos. Ya sea ágil o tradicional, o tal vez un híbrido.

En la creciente volatilidad e incertidumbre de hoy, se desea trabajar con un control sobre el cómo se trabaja, la interacción, los clientes y la administración. Los problemas cambian, las personas cambian, y aún más fuerte, las ideas cambian. Si bien todavía hay una necesidad de desarrollo y gestión del estilo impulsado por el plan en algunas situaciones, el mayor crecimiento se encuentra en ágil y flexible. Aquí se formaliza una metodología idónea en el desarrollo de software desde la perspectiva de trabajo en el ámbito de la neuroingeniería, pero más allá de eso el enfoque abarca una metodología donde los laboratorios de investigación y la academia se unen, lográndose adaptar en sus necesidades y formas de trabajo.

3.3. Metodologías tradicionales

Estas metodologías se basan en una serie de pasos secuenciales, como la definición de requisitos, la creación de soluciones, las pruebas y la implementación. Requieren definir y documentar un conjunto estable de requisitos al comienzo de un proyecto. Hay muchas metodologías diferentes tradicionales, sin embargo, sólo describiré las tres más significativas: Cascada (*Waterfall*), Modelo Espiral (*Spiral Development Model*) y Proceso Unificado (*Unified Process*).

3.3.1. Cascada

Propuesta en 1970 por Winston Royce. Se considera el modelo clásico de ciclo de vida de desarrollo de software (SDLC). Es el primer modelo de proceso que sigue las etapas secuenciales del desarrollo de software. También es el primer modelo impulsado por un plan que hace hincapié en la planificación temprana, el análisis de requisitos y el diseño del proceso. El enfoque de cascada enfatiza una progresión estructurada entre fases definidas. Cada fase consiste en un conjunto definido de actividades y entregables que deben realizarse antes de que pueda comenzar la siguiente fase. Las fases siempre tienen nombres diferentes, pero la idea básica es que la primera fase trata de captar lo que hará el sistema y sus requerimientos, la segunda fase determina cómo se diseñará, la tercera etapa es donde los desarrolladores comienzan a escribir el código, la cuarta fase es la prueba del sistema y la fase final se enfoca en tareas de implementación tales como capacitación y documentación pesada. Sin embargo, en la práctica de la ingeniería de software, el término cascada se usa como un nombre genérico para toda la metodología de ingeniería de

software secuencial [25]. Figura 1. Ciclo de vida modelo cascada a continuación muestra un ciclo de vida tradicional en cascada.

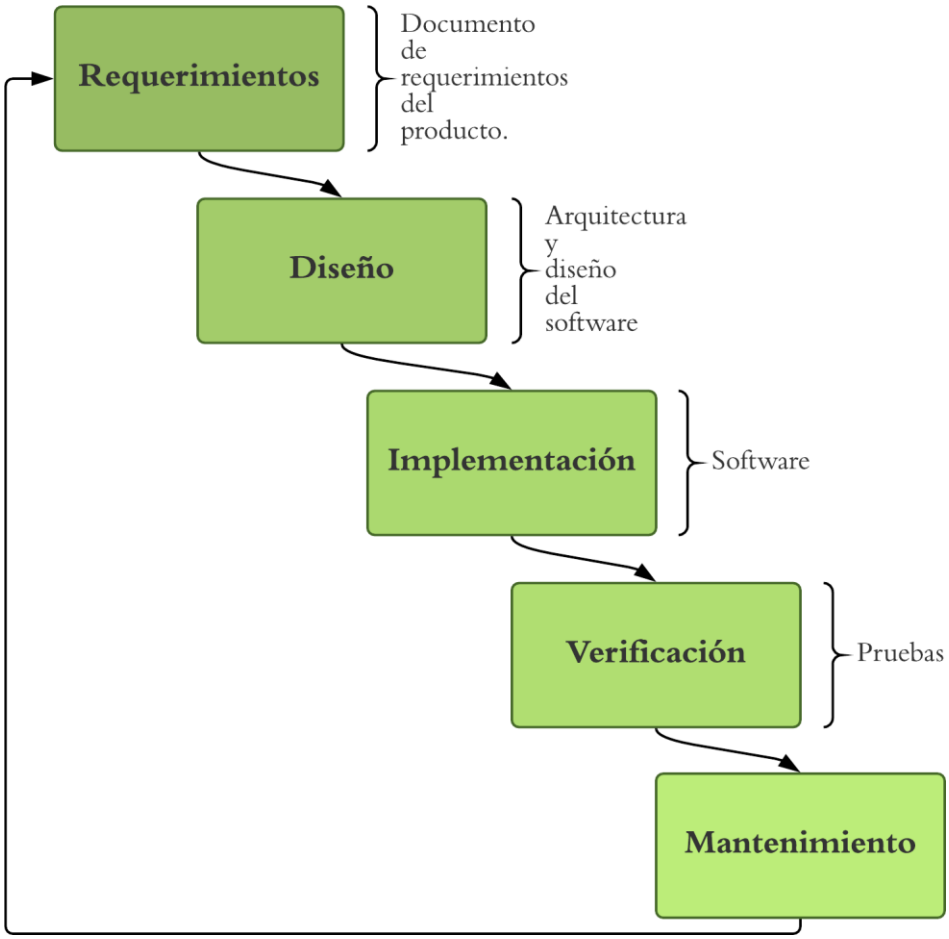

Figura 1. Ciclo de vida modelo cascada

Las entregas necesarias para pasar a la siguiente fase son las siguientes, presentadas en la Tabla 1. Entregas por fase en modelo de cascada.

Tabla 1. Entregas por fase en modelo de cascada

Requerimientos	Diseño	Implementación	Verificación	Mantenimiento
Requerimientos	Vistas / diseño pantalla	UI	Script de pruebas	Entrenamiento
	Base de datos	Lógica	Reporte de defectos	Documentación
	Objetos (Clases)	Reportes	Retro alimentación de usuario	
Plan de pruebas				
Administración del proyecto				
Carta del proyecto, Reporte de estado, Cambio de requerimientos				

3.3.2. Modelo Espiral

Definido por Barry Boehm, basado en la experiencia con varios refinamientos del modelo de cascada aplicado a grandes proyectos de software, con elementos de diseño y creación de prototipos en etapas, en un esfuerzo por combinar las ventajas de los conceptos de arriba hacia abajo y de abajo hacia arriba. El modelo en espiral tiene cuatro fases observables en la Figura 2. Ciclo de vida modelo espiral:

- Planificación: Se identifican los objetivos específicos para la fase del proyecto
- Análisis de riesgos: Se identifican, analizan riesgos clave y se obtiene información para reducir estos riesgos
- Desarrollo y validación: Se elige un modelo apropiado para la siguiente fase de desarrollo
- Evaluación: El proyecto se revisa y los planes se elaboran para la próxima ronda de espiral

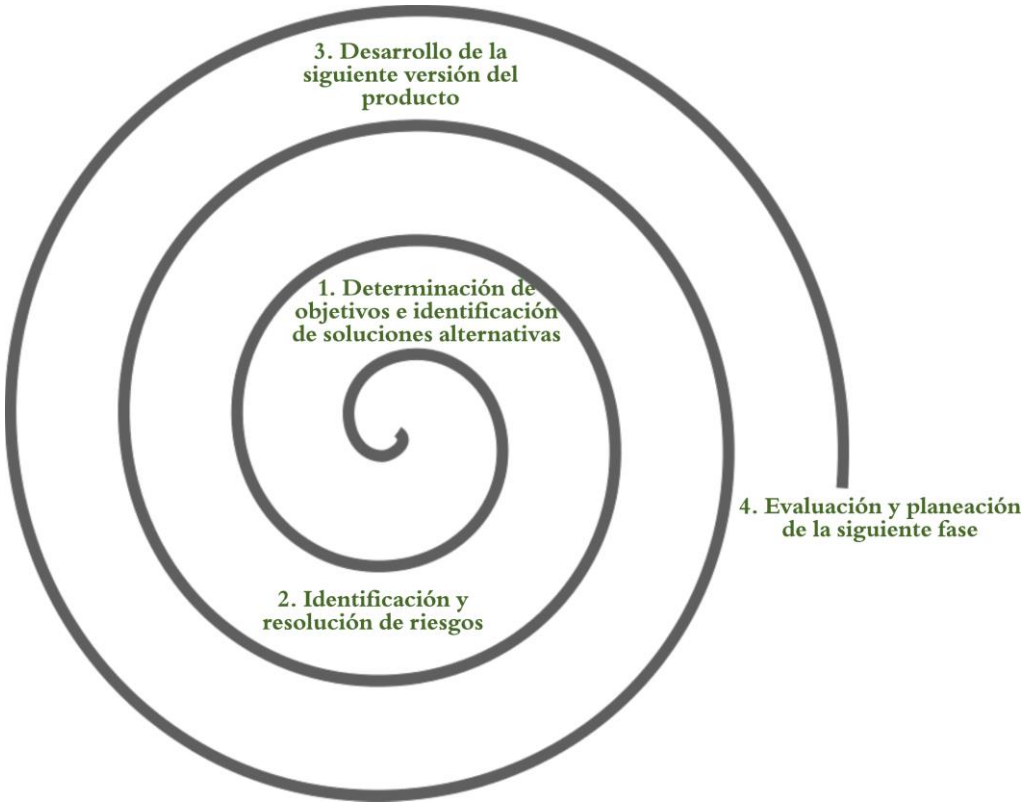

Figura 2. Ciclo de vida modelo espiral

Este modelo está más preocupado por el análisis de riesgo. Este modelo se adapta a los cambios en los requerimientos, ya que se garantiza la participación temprana del usuario en el desarrollo del sistema. Este modelo ayuda a ver el sistema muy pronto. El modelo en espiral es adecuado cuando la evaluación de riesgos y costos es importante y los requerimientos son complejos. Este modelo no es significativo para su uso en proyectos pequeños y el éxito del proyecto depende en gran medida del análisis de riesgo. Por lo tanto, es costoso hasta cierto punto [26]□.

3.3.3. Proceso Unificado

En el proceso unificado todos los esfuerzos, incluido el modelado, se organizan en flujos de trabajo en y se realizan de manera iterativa e incremental [26]□. El ciclo de vida es presentado en la Figura 3. Ciclo de vida UP.

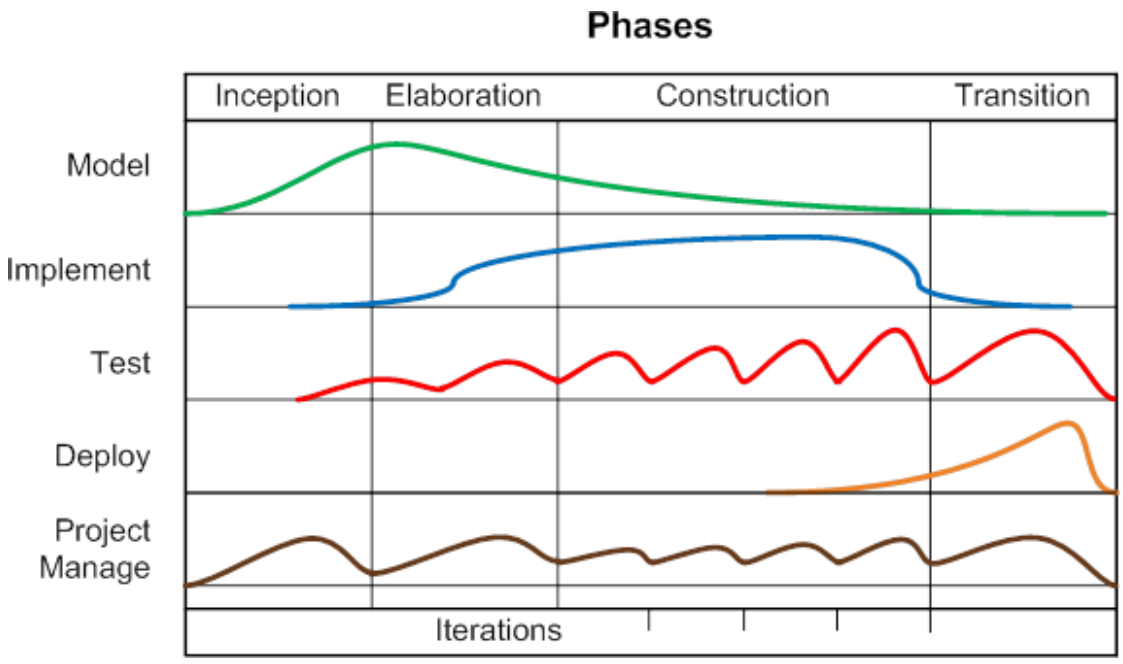

Figura 3. Ciclo de vida UP

De las características principales se destacan:

- Utiliza una arquitectura basada en componentes que crea un sistema que es fácilmente extensible, promueve la re utilización del software y es intuitivamente comprensible. El componente comúnmente utilizado para coordinar proyectos de programación orientados a objetos.
- Utiliza software de modelado visual, como UML, que representa su código como una notación esquemática para permitir que personas menos técnicamente competentes que pueden tener una mejor comprensión del problema tengan una mayor aportación.
- Se ha comprobado que los requisitos de gestión de uso utilizando casos de uso y escenarios son muy efectivos tanto para capturar los requisitos funcionales como para ayudar a mantener a la vista los comportamientos anticipados del sistema.
- El diseño es iterativo e incremental: esto ayuda a reducir el perfil de riesgo del proyecto, permite una mayor retroalimentación de los clientes y ayuda a los desarrolladores a mantenerse enfocados.
- Verificar la calidad del software es muy importante en un proyecto de software. UP ayuda en la planificación del control de calidad y la evaluación incorporados en todo el proceso que involucra a todos los miembros del equipo.

3.3.4. Características de las metodologías tradicionales

Las metodologías tradicionales han sido exitosas durante mucho tiempo. Éstas imponen un proceso disciplinado y estricto al desarrollo de software con el objetivo de hacer que el desarrollo de software sea más predecible y más eficiente [26]□. Aún en la actualidad no se ha observado que tengan una alta tasa de éxito. Algunos autores señalan a las metodologías tradicionales como burocráticas, ya que seguir con todos los procesos genera una ralentización en todo el ritmo de desarrollo [27]□. Las metodologías tradicionales tienen las siguientes características en común:

Enfoque predictivo: Las metodologías tradicionales tienden a planificar primero una gran parte del proceso del software con gran detalle durante un largo período de tiempo. Este enfoque sigue una disciplina de ingeniería donde el desarrollo es predictivo y repetible.

Documentación integral: Las metodologías tradicionales ven la documentación de requerimientos como la pieza clave de la documentación. Un proceso principal en estas metodologías es el proceso de gran diseño por adelantado (BDUF, del inglés B*ig Design UpFront*), en el que se cree que es posible reunir todos los requerimientos del cliente por adelantado, antes de escribir cualquier código. La previsibilidad es muy importante en los proyectos de software que son críticos para la vida.

Orientado al proceso: El principal objetivo de las metodologías tradicionales es definir un proceso que funcione bien para quien lo esté usando. El proceso consiste en ciertas tareas que deben ser realizadas por los administradores, diseñadores, codificadores / desarrolladores (C*oders*), probadores (*Testers*), etc. Para cada una de estas tareas hay un procedimiento bien definido.

Orientado a herramientas: Aquí se le da enfoque a las herramientas de administración de proyectos, editores de código, compiladores, etc., deben estar en uso para completar y entregar cada tarea durante las fases.

Aspectos importantes de estas metodologías se encuentran en la fase de diseño, en la cual mediante diferentes técnicas y modelos se describe todo el software a realizar desde el inicio, abarcando todos los requerimientos funcionales y no funcionales establecidos en la fase de toma de requisitos, impactando directamente sobre la capacidad del sistema para cumplir o no el total de requerimientos establecidos.

3.4. Metodologías Ágiles

El modelo ágil es un subconjunto de iterativos y evolución. Los métodos de desarrollo de software intentan ofrecer una vez más una respuesta a la comunidad empresarial ansiosa que solicita un peso más ligero junto con procesos de desarrollo de software más rápidos y ágiles.

La metodología ágil utiliza un proceso iterativo en el que todos los equipos y colaboradores colaboran, y el cliente proporciona comentarios durante todo el proceso de desarrollo de un nuevo producto de software. El principal beneficio de la metodología ágil es que el producto se entrega al cliente en un lapso de tiempo más corto.

En el año 2001, en Estados Unidos de América se tuvo lugar una reunión en donde 17 de los mejores críticos de los modelos de mejora en desarrollo de software, los cuales fueron convocados por el ingeniero Kent Beck, quién años atrás se había constituido en uno de los progenitores de las metodologías de desarrollo de software. Consecuentemente los integrantes de esta reunión sintetizaron una serie de principios de toda metodología ágil como una comunicación directa con el cliente, gente altamente motivada crearon 12 principios que a su vez se puede resumir en 4 postulados (Tabla 2. Postulados manifiesto ágil), los cuales se han nominado el manifiesto ágil [21]□.

Tabla 2. Postulados manifiesto ágil

Individuos e interacciones	Sobre	Procesos y herramientas
Software funcionando		Documentación extensiva
Colaboración con el cliente		Negociación contractual
Respuesta ante el cambio		Seguir un plan

De estos postulados cabe resaltar la importancia del agilismo, ya que sobre esta "filosofía" es la que se trabaja todas las técnicas y metodologías ágiles existentes. Entendiéndose y aplicando los postulados se ejerce poder sobre el "Hay que ser para poder hacer".

Principios del manifiesto [21]□:

1. La prioridad principal es satisfacer al cliente mediante tempranas y continuas entregas de software que le reporte un valor
2. Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.
3. Entregar frecuentemente software que funcione, desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre una entrega y la siguiente
4. La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto
5. Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir el trabajo
6. El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo
7. El software que funciona es la medida principal de progreso

8. Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante
9. La atención continua a la calidad técnica y al buen diseño mejora la agilidad
10. La simplicidad es esencial
11. Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos
12. En intervalos regulares, el equipo reflexiona respecto de cómo llegar a ser más efectivo, y según esto ajusta su comportamiento

Dentro de las metodologías ágiles se encuentran una gran variedad de métodos, modelos, prácticas y metodologías (actualmente se encuentran más de 60 metodologías ágiles), aquí presente no se hará una revisión de todas las metodologías ágiles, pero se hace énfasis en que el agilismo presenta dos estados, o más bien, posee dos enfoques ágiles como lo son los procesos ágiles y las producciones ágiles.

Agilismo:

- Proceso: Gestiona el proceso del desarrollo de software
  - SCRUM
  - Kanban
- Producción: Gestiona las tareas, herramientas y técnicas para desarrollar producto de software
  - *Extreme Programming* (XP)
  - *Behavior Driven Development* (BDD)
  - *Test Driven Development* (TDD)

Cuáles son metodologías de programación basadas en la adaptabilidad a cualquier cambio como un medio para aumentar las posibilidades de éxito de un proyecto. La metodologías aquí presentes intentan minimizar los riesgos durante la ejecución de un proyecto desarrollando software en iteraciones, que generalmente duran de una a cuatro semanas (estructura de las iteraciones se observa en la Figura 4. Ciclo de vida metodologías ágiles). Cada iteración es como un proyecto en miniatura del proyecto final, e incluye todas las tareas necesarias para implementar nuevas funcionalidades: planificación, análisis de requisitos, diseño, codificación, pruebas y documentación. Un proyecto de programación ágil tiene como objetivo lanzar un nuevo software al final de cada iteración, y entre cada iteración el equipo re evalúa sus prioridades.

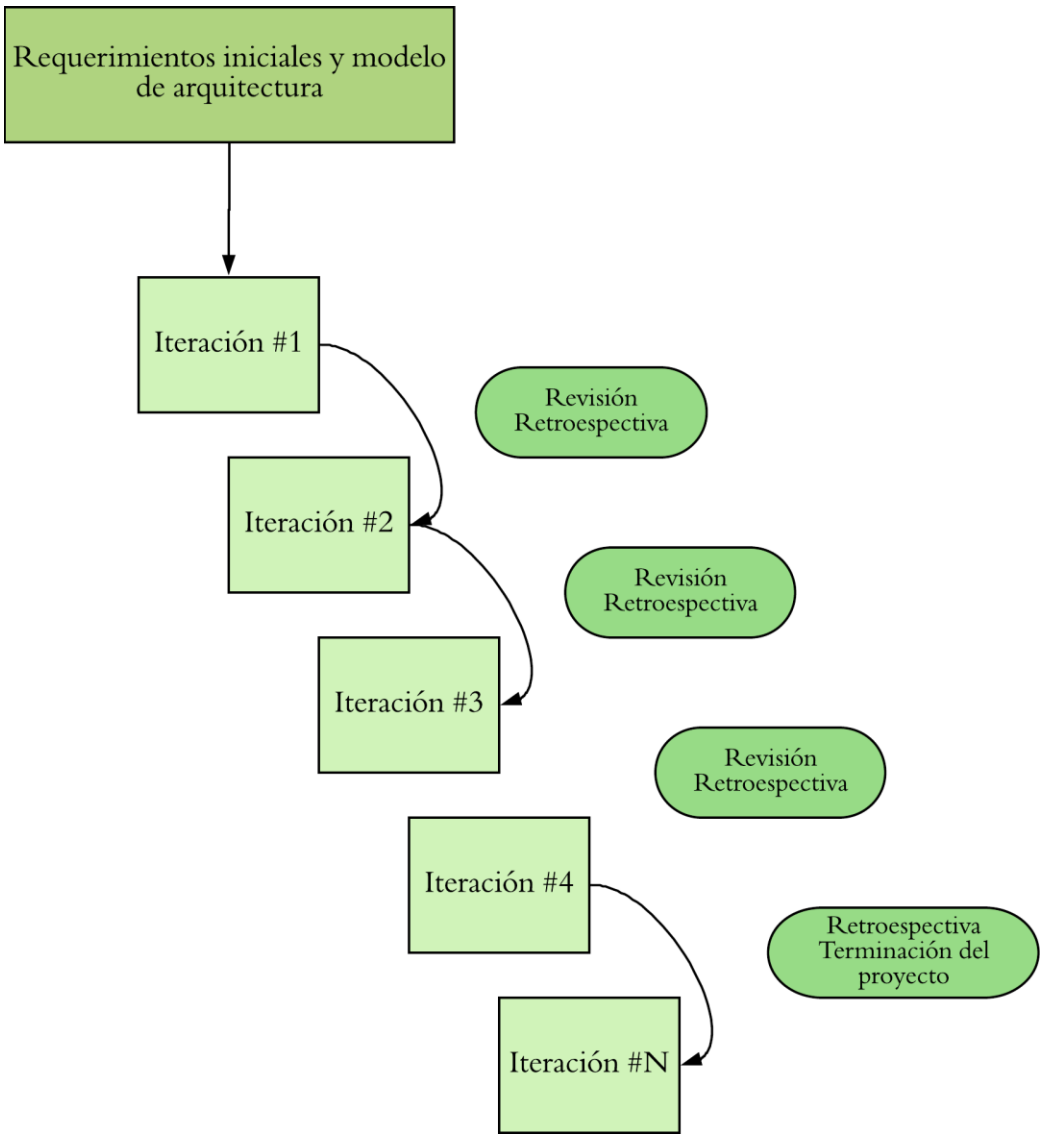

Figura 4. Ciclo de vida metodologías ágiles

3.4.1. Programación Extrema (XP)

La programación extrema (XP) se origina a partir de las desventajas del modelo de proceso tradicional. Se puede caracterizar por ciclos de desarrollo cortos, planificación incremental, retroalimentación continua, confianza en la comunicación y diseño evolutivo [28]□. Con todas las cualidades anteriores, los programadores de XP responden a un entorno cambiante con mucho más valor. Además, según varios autores [28]–[31]□, los miembros del equipo de XP dedican unos

minutos a la programación, unos minutos a la gestión de proyectos, unos minutos a los diseños, unos minutos a los comentarios y unos minutos a la formación de equipos muchas veces al día. El término "extremo" viene de llevar estos principios y prácticas de sentido común a niveles extremos. A continuación, se muestra un resumen de los términos y prácticas de XP [26], [30]□:

- Planificación: El programador estima el esfuerzo necesario para la implementación de las historias de usuario y el cliente decide el alcance y el calendario de los lanzamientos según las estimaciones.
- Versiones pequeñas / cortas: Una aplicación se desarrolla en una serie de versiones pequeñas y actualizadas con frecuencia. Las nuevas versiones se lanzan en cualquier momento, con frecuencias diarias o mensuales.
- Metáfora: El sistema se define mediante un conjunto de metáforas entre el cliente y los programadores, que describe cómo funciona el sistema.
- Diseño simple: El énfasis está en el diseño de la solución más simple posible que se implementa, y la complejidad innecesaria y el código adicional se eliminan de inmediato.
- Refactorización: Implica la re estructuración del sistema eliminando la duplicación, mejorando la comunicación, simplificando y agregando flexibilidad pero sin cambiar la funcionalidad del programa.
- Programación de pares: Todo el código de producción está escrito por dos programadores en una computadora.
- Propiedad colectiva: Ninguna persona individual posee o es responsable de los segmentos de código individuales, en lugar de que cualquiera pueda cambiar cualquier parte del código en cualquier momento.
- Integración continua: Una nueva pieza de código se integra con el sistema actual tan pronto como está listo. Al integrarse, el sistema se construye de nuevo y todas las pruebas deben pasar para que se acepten los cambios.
- 40 horas por semana: Nadie puede trabajar dos horas extraordinarias seguidas. Un máximo de 40 horas por semana de trabajo, de lo contrario se trata como un problema.
- Cliente en el sitio: El cliente debe estar disponible en todo momento con el equipo de desarrollo.
- Normas de codificación: Las reglas de codificación existen y son seguidas por los programadores para brindar coherencia y mejorar la comunicación entre el equipo de desarrollo.

Este modelo de proceso aborda los riesgos fundamentales en el desarrollo de software y se basa en principios de sentido común y prácticas comprensibles. XP tiene seis fases: Exploración, Planificación, Iteraciones de liberación, Producción, Mantenimiento y Muerte, representadas en Figura 5. Ciclo de vida XP para un sprint. El modelo XP requiere la práctica de desarrollo centralizado y solo es adecuado para los proyectos aptos para XP [31]□.

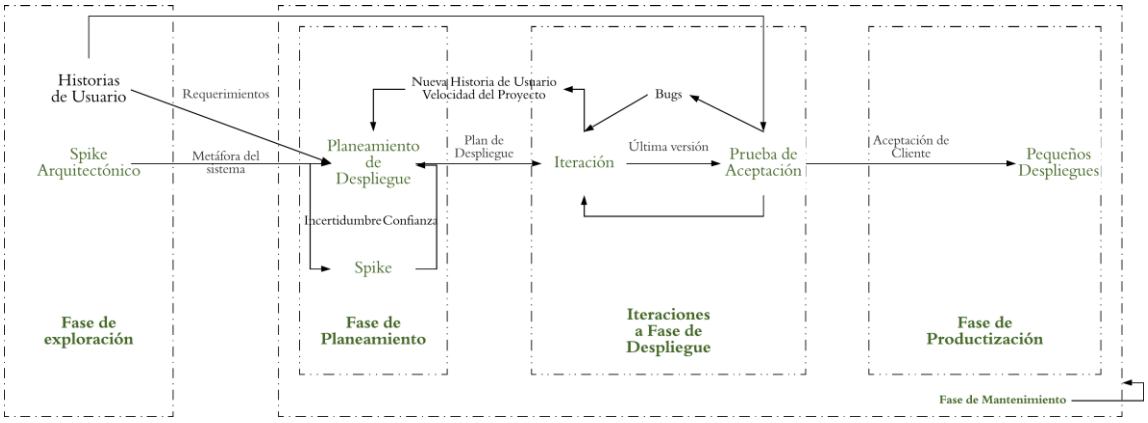

Figura 5. Ciclo de vida XP

En la fase de Exploración, el cliente escribe las tarjetas de la historia (descripción en sus términos) que desea incluir en su programa. Esto conduce a la fase de planificación donde se establece un orden de prioridad para cada historia de usuario y se desarrolla un calendario de la primera versión. A continuación, en la fase Iteraciones para liberar, la primera iteración del equipo de desarrollo es crear un sistema con la arquitectura de todo el sistema y luego integrar y probar su código continuamente. Las pruebas adicionales y la comprobación del rendimiento del sistema antes de que el sistema pueda entregarse al cliente se realizan en la fase de producción. Las ideas propuestas y las sugerencias encontradas en esta fase se documentan para su implementación posterior en las versiones actualizadas realizadas en la fase de mantenimiento. Finalmente, la fase de muerte está cerca cuando el cliente no tiene más historias para implementar y toda la documentación necesaria del sistema está escrita, ya que no se realizan más cambios en la arquitectura, el diseño o el código [29], [31]□.

En XP se desarrolla fundamentalmente, como se describió antes, en cuatro fundamentos, retroalimentación a escala fina, proceso continuo en lugar de lotes, entendimiento compartido, y bienestar del programador, en la cual XP es resaltada. Desde los fundamentos se desarrollan valores, llamados también valores originales de XP, la cuales son: simplicidad, comunicación, retroalimentación (*feedback*) y coraje. Un quinto valor, respeto, fue añadido en la segunda edición de *Extreme Programming Explained.* Los cinco valores se detallan a continuación [32]□:

- Simplicidad: Para la programación extrema la simplicidad es su base de construcción. Una base simple constituye el agilismo en el desarrollo y mantenimiento, evitando numerosas modificaciones que aumentan la complejidad del código por parte de los desarrolladores. La simplicidad es mantenida por parte de la refactorización del código, la documentación de cada función y método del código, teniendo una guía de estilo adecuada para ello. Aplicando la simplicidad junto con la autoría colectiva del código y la programación por parejas se asegura que cuanto más grande se haga el proyecto, todo el equipo conocerá más y mejor el sistema completo [29], [32]□.

- Comunicación: Entre mas simple sea el código, éste comunica mejor a los programadores, cuando el código se vuelve complejo es necesario un mayor esfuerzo para hacerlo comprensible. Los comentarios se desfazan en relacion al código a medida que se modifica. Las pruebas unitarias y la programación por parejas han sido las formas de comunicación mas útiles ya que de esta forma se describe de manera concreta el diseño de cada componente del código, facilitando además la comunicación fluida con el cliente. El cliente decide que características tienen prioridad y siempre debe estar disponible para solucionar dudas [29], [32]□.

- Retroalimentación (*feedback*): El estado del proyecto se conoce en tiempo real cuando se tiene al cliente integrado en el proyecto. Los resultados se ven visibles en ciclos muy cortos, además de que se minimizan el tener que rehacer segmentos que no hacen parte de los requerimientos, ayudando a los programadores a centrarse en lo importante. Con ayuda de las pruebas unitarias se permite encontrar fallos debido a cambios en el código, dandonos la oportunidad de tener el código como una fuente de retroalimentación también [29], [32]□.

- Coraje o valentía: Se debe tener valentía para construir la confianza con los compañeros de trabajo, especialmente la programación en parejas, con una confianza sólida se obtiene calidad de código sin repercutir negativamente en la productividad. La simplicidad se debe tener presente en cada bloque de código, aunque ésta sea dificil de adoptar. Evitar rutas flexibles que en un futuro deba ser modificado en su totalidad sólo por realizar una solución rápida, aquí es donde se requeire el coraje para poder implementar las funcionalidades que requeire el cliente sin tener que optar rutas que puedan cambiar su curso en el tiempo. La forma correcta de construir marcos de trabajo es mediante la refactorización del código en sucesivas aproximaciones [29], [32]□.

- Respeto: Uno de las características principales para que el trabajo colectivo sea productivo corresponde al respeto mutuo, los programadores no deben realizar cambios que entorpezcan el funcionamineto de las pruebas y generen fallas en el proceso de desarrollo. Una eficiente refactorizacion del código va en conjunto de la alta calidad y el diseño óptimo del producto [29], [32]□.

En la programación extrema el flujo de trabajo se puede ver como la arquitectura de Harvard de los procesadores, de esta manera se logra definir la eficiencia (e) mediante la ecuación 1.

$$e = x + (n * nc - 1) \quad (1)$$

Dónde

x: Tiempo de cada ciclo

n: Número de participantes

nc: Número de iteraciones del proyecto

De tal manera que con un número n de escritorios remotos donde n es igual al número de participantes en la programación del proyecto se puede aumentar la eficiencia del trabajo hasta 4 veces, entiéndase que cada x tiempo un programador deja de programar una clase o parte del programa para programar otra diferente en el escritorio remoto de otro de los componentes del equipo.

3.4.2. SCRUM

Un proyecto Scrum consiste en un esfuerzo de colaboración para crear un nuevo producto, servicio u otro resultado tal como se define en la Declaración de la visión del proyecto (*Project Vision Statement*). Los proyectos se ven afectados por limitaciones de tiempo, costos, alcance, calidad, recursos, capacidades organizacionales y demás limitaciones que dificultan su planificación, ejecución, administración y, por último, su éxito. Sin embargo, la implementación exitosa de los resultados de un proyecto terminado le proporciona ventajas económicas considerables a una organización. Por lo tanto, es importante que las organizaciones seleccionen e implementen un método adecuado de gestión de proyectos.

Scrum es uno de los métodos ágiles más populares y más usados actualmente. Es un framework de procesos iterativo e incremental para desarrollar cualquier producto o administrar cualquier trabajo. Scrum se concentra en cómo deben funcionar los miembros del equipo para producir la flexibilidad del sistema en un entorno en constante cambio. Al final de cada iteración produce un conjunto potencial de funcionalidades.

Scrum no requiere ni proporciona ningún método / práctica de desarrollo de software específico para ser utilizado. En cambio, requiere ciertas prácticas y herramientas de administración en diferentes fases de Scrum para evitar el caos por imprevisibilidad y complejidad. Se podría decir que SCRUM es un marco de trabajo. Una fortaleza clave de Scrum radica en el uso de equipos

interfuncionales (*cross-functional*), auto organizados y empoderados que dividen su trabajo en ciclos de trabajo cortos y concentrados llamados Sprints. Diferencia entre algunas metodologías ágiles radica en la duración del Sprint. La Figura 6. Ciclo de vida SCRUM para un sprint□ [33]□ proporciona una visión general de flujo de un proyecto Scrum.

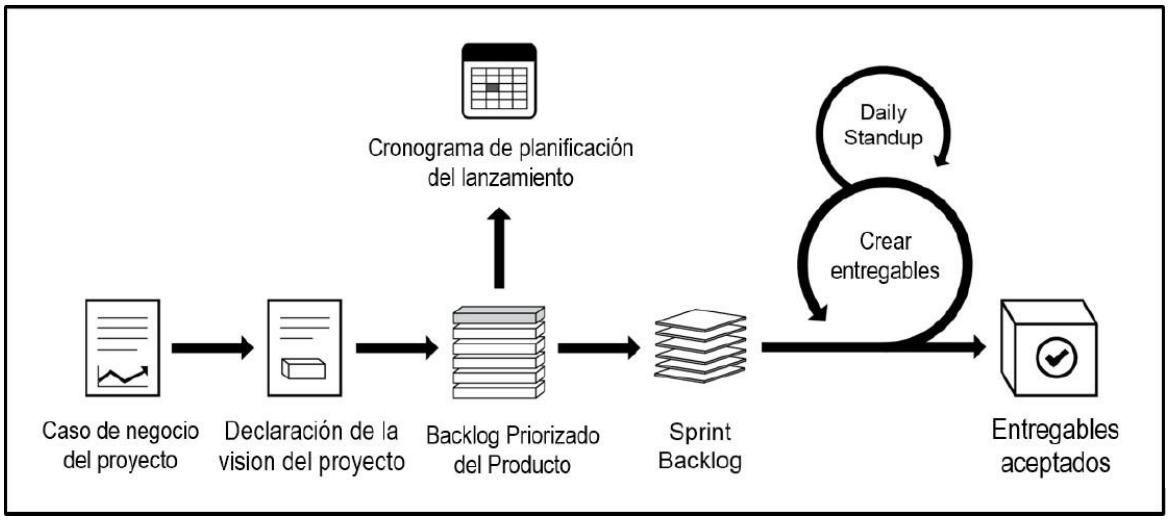

Figura 6. Ciclo de vida SCRUM para un sprint

El ciclo de Scrum empieza con una reunión de Stakeholders, durante la cual se crea la visión del proyecto. Después, el Product Owner desarrolla una Backlog Priorizado del Producto(*Prioritized Product Backlog*) que contiene una lista requerimientos del negocio y del proyecto por orden de importancia en forma de una historia de usuario. Cada sprint empieza con una reunión de planificación del sprint (*Sprint Planning Meeting*) durante la cual se consideran las historias de usuario de alta prioridad para su inclusión en el sprint. Un sprint generalmente tiene una duración de una a seis semanas durante las cuales el Equipo Scrum trabaja en la creación de entregables (del inglés *deliverables*) en incrementos del producto. Durante el sprint, se llevan cabo Daily Standups muy breves y concretos, donde los miembros del equipo discuten el progreso diario. Hasta el final del sprint, se lleva a cabo una Reunión de Revisión del Sprint (*Sprint Review Meeting*) en la cual se proporciona una demostración de los entregables al Product Owner y a los Stakeholders relevantes. El Product Owner acepta los entregables sólo si cumplen con los criterios de aceptación predefinidos. El ciclo del sprint termina con una Reunión de Retrospectiva del Sprint (*Retrospect Sprint Meeting*), donde el equipo analiza las formas de mejorar los procesos y el rendimiento a medida que avanzan al siguiente sprint [33]□.

El proceso Scrum puede cambiar considerablemente la descripción del trabajo y personaliza el equipo del proyecto Scrum. Por ejemplo, el gerente del proyecto, es decir, Scrum Master, ya no necesita organizar el equipo, sino que el equipo se organiza solo y toma decisiones sobre qué hacer. La frase "La mayoría de la administración está acostumbrada a dirigir el proyecto, le dice al equipo qué hacer y luego se asegura de que lo haga. Scrum confía en la auto-organización, con el equipo

decidiendo qué hacer mientras la administración ejecuta la interferencia y elimina los obstáculos ” dicha por Ken Schwaber [34]□, se ha visto evidenciada en organizaciones donde se aplica el marco. Scrum se ha utilizado con éxito en miles de proyectos en diferentes organizaciones que producen mejoras significativas en la productividad.

3.4.2.1. Historias de usuario (HU)

A medida que los profesionales hacen la transición al desarrollo ágil y/o híbrido, los requerimientos y requisitos se expresan cada vez más como historias de usuario. Inventado por Connextra en el Reino Unido y popularizado por Mike Cohn y Rachel Davies [35], [36]□. Una historia de usuario (HU) describe la funcionalidad que será valiosa para un usuario o comprador de un sistema o software. Las historias de usuarios se componen de tres aspectos:

I. Una descripción escrita de la historia utilizada para la planificación y como recordatorio
II. Conversaciones sobre la historia que sirven para desarrollar los detalles de la historia
III. Pruebas que transmiten y documentan detalles y que pueden utilizarse para determinar cuándo se completa una historia

Estas HU se escriben tradicionalmente en tarjetas, de una forma aliterada de tarjeta, conversación y confirmación. Diferentes autores [35]–[38] han dicho que las tarjetas "representan los requisitos y requerimientos del cliente en lugar de documentarlas". Esta es la manera perfecta de pensar en las HU’s: aunque la tarjeta puede contener el texto de la historia, los detalles se resuelven en la conversación y registrado en la confirmación.

Las HU solo capturan los elementos esenciales de un requerimiento: para quién es, qué espera del sistema y, opcionalmente, por qué es importante. El formato más conocido [35], [37]□ es: "Como <*tipo de usuario / rol*> necesito <*descripción de la funcionalidad*>, con la finalidad de <*descripción de razón o resultado*>”, seguido de la cantidad de escenarios posibles, de la forma: “<*Título del escenario*>, en caso que <*contexto*> y/o <*contexto*>, cuando <*evento*>, el sistema <*resultado / comportamiento*>”. Principalmente estas se centran en el valor del cliente con una fuerte intención de fomentar la comunicación.

3.5. ¿Ágil o Tradicional?

Si bien hay decenas de enfoques de gestión de proyectos de software diferentes, el enfoque principal es la selección de alguna de estas metodologías, pero la elección final debe hacerse teniendo en cuenta la naturaleza del negocio, o en nuestro caso del laboratorio / grupo de

investigación y sus requerimientos. De todos los métodos, las metodologías tradicionales y ágiles a menudo se enfrentan entre sí.

En metodologías tradicionales, el resultado del proyecto se fija desde el principio. El objetivo es crear un plan sobre cómo utilizar mejor a las personas y los recursos para lograr este objetivo. Los proyectos a menudo siguen el llamado modelo de cascada, donde el proceso del proyecto se divide en fases secuenciales y todos los subresultados se construyen unos sobre otros. Por ejemplo, si un hito no se completa a tiempo, todos los hitos posteriores se retrasarán. Las metodologías tradicionales se adapta bien al logro de objetivos fijos de manera bien planificada con anticipación. Casi siempre hay desviaciones del plan durante los proyectos, especialmente en los de software, o más generalizado los de TI. Se agregan nuevas características, y las antiguas desaparecen. Los desafíos que parecen simples al principio resultan ser monstruos hambrientos que devoran decenas o incluso cientos de días de trabajo. La economía de mercado es despiadada, y los centros investigativos no se libran de ella. Los procesos rápidos, las dependencias complejas y la creciente inundación de información hacen que sea imposible mirar hacia el futuro y pensar en todas las eventualidades desde el principio. Cuanto más lejos esté el final del proyecto en el futuro, más inciertos serán los pronósticos. La experiencia muestra que los proyectos de TI son proyectos a largo plazo con un esfuerzo difícil de estimar. Y a menudo hay grandes cambios en la implementación y en el producto durante la implementación, ya sea porque los requisitos del cliente cambian o porque existen dificultades con la codificación / programación. Las metodologías tradicionales simplemente no pueden manejar esta situación muy bien. Cualquier cambio en el plan del proyecto, sea objetivos y/o requerimientos requiere de esfuerzo y costos (a menudo indirectos). Hay retrasos porque los planes deben actualizarse o los conflictos de recursos deben resolverse. Además, el presupuesto a menudo está sobrecargado, lo que genera caos y, al final, promesas rotas, objetivos no cumplidos y clientes insatisfechos. Aquí es exactamente donde entra en juego de las metodologías ágiles. Los equipos ágiles no trabajan para conseguir un objetivo bien definido, sino que desarrollan un producto durante un período más largo en estrecha colaboración con el cliente. Al mismo tiempo, las metas a corto y largo plazo se comparan entre sí de manera iterativa. Las metodologías ágiles no tienen problemas con los cambios; de hecho, les da la bienvenida.

Tabla 3. Comparación de las ventajas y desventajas de los procesos ágiles y tradicionales [39]

	Modelos	Ventajas	Desventajas
Tradicional	Cascada Desarrollo incremental Desarrollo iterativo Desarrollo en espiral Modelo de prototipo Desarrollo rápido de aplicaciones	Adecuado para grandes sistemas y equipos. Maneja sistemas altamente críticos de manera efectiva. Apropiado para un entorno de desarrollo estable. Requerir personal experimentado al inicio.	Mayor longitud en cada iteración o incremento. No puede acomodar cambios en cualquier momento. Falta de participación del usuario a lo largo del ciclo de vida del producto.

		Éxito logrado a través de la estructura y el orden.	Costoso para el entorno de desarrollo dinámico. Supongamos que, los cambios futuros no ocurrirán.
Ágil	Scrum Extreme Programming (XP) Dynamic System (DS) Development Method Kanban Feature Driven Development (FDD)	Adecuado para sistemas pequeños y medianos y equipos. Puede acomodar cambios en cualquier momento. Eficaz para el entorno de desarrollo dinámico. Personal experto ágil requerido a lo largo del ciclo de vida. Éxito logrado a través de la libertad y el caos.	No es adecuado para grandes sistemas (excepto FDD). Longitud más corta en cada iteración. Puede acomodar cambios en cualquier momento. Costoso para el entorno de desarrollo estable. Supongamos que, frecuentes cambios futuros ocurrirán.

Entre sus mayores diferencias, como se observa en la Tabla 3. Comparación de las ventajas y desventajas de los procesos ágiles y tradicionales [39]□, y se puede considerar que es el hito de éxito en el agilismo, la cual toma el mayor problema de las metodologías tradicionales y lo transforma, generando como solución la iteración, las planeaciones segmentadas y el cambio de roles. Teniendo como el problema del modelo de cascada, la pre definición de roles al equipo del proyecto haciéndolo menos flexible, y asumiendo que la única función de los usuarios es especificar los requisitos, y que todos los requisitos se pueden especificar de antemano. Desafortunadamente, los requisitos crecen y cambian a lo largo del proceso y más allá, y requieren una retroalimentación considerable y una consulta iterativa.

Las metodologías ágiles dictan que los proyectos se administren en Sprints de 2 o 3 semanas para entregar un producto medible y comprobable. Las metodologías tradicionales funcionan bien cuando los resultados del proyecto se entienden razonablemente. Pero muchos proyectos duran meses o años. Si los entregables son demasiado complicados para que todo el proyecto se administre a través de un enfoque tradicional definido, se aplica ágil a entregas con incertidumbre relativa y tradicionales a entregas con certeza. Esto crea un enfoque más efectivo para la gestión de sus proyectos.

3.5.1. Culpabilidad – Metodologías Tradicionales

En el artículo de agosto de 2015, "*Agile IT Remains 'immature' in Federal Government*", Billy Mitchell lo explica bien [40]□: "A medida que las metodologías ágiles comenzaron a tomar impulso, algunos lo consideraron como una excusa para deshacerse de las cosas que son solo una buena práctica, con cada nueva ola, todos piensan que es una oportunidad de deshacerse de todo y

luego comenzar de nuevo". Los defensores de las metodologías ágiles culpan a la metodología cascada por crear proyectos que no tienen éxito. Las causas fundamentales de la mayoría de los problemas de la metodología cascada son el resultado de proyectos de escaso alcance, mal planificados y mal ejecutados, creados por equipos no entrenados. No gastan el tiempo requerido para comprender completamente los requisitos del cliente, para definir claramente un documento de alcance medible, para desarrollar un plan detallado del proyecto o para evaluar continuamente el riesgo.

3.5.2. Culpabilidad – Metodologías Ágiles

Los defensores de las metodologías tradicionales también culpan a las metodologías ágiles por la creación de proyectos que no están claramente definidos con pocos requisitos de los clientes entendidos por adelantado. Las causas fundamentales de la mayoría de los problemas de las metodologías tradicionales son el resultado de no entender cómo funciona el agilismo, la falta de un proceso ágil claro y una comunicación deficiente sobre el agilismo. Además, los equipos ágiles a menudo se unen sin la capacitación adecuada y el entrenamiento continuo. Es un cambio que requiere una gestión cuidadosa. Operar dentro de los Sprints es solo un elemento de todo el proceso de las metodologías ágiles.

En realidad, no existe una metodología sea tradicional o ágil que se adapte perfectamente a cada proyecto u organización. La elección de implementar una metodología depende en gran medida de factores como la naturaleza del proyecto, el tamaño, los recursos involucrados, entre otros. La mayoría de las veces, los gerentes de proyectos inteligentes deciden qué metodología adoptar durante el inicio o en la primera parte del proyecto. Existen algunos factores definidos por profesionales del área que se puede tener en cuenta al elegir la metodología correcta para los proyectos:

- Los requisitos del proyecto. ¿Están los requisitos claros?. Si los requisitos del proyecto no están claros o tienden a cambiar, se elige la metodología ágil. Y, la metodología tradicional se adapta mejor a una situación en la que los requisitos están claramente definidos y bien entendidos desde el principio.
- Tecnología involucrada en el proyecto. La metodología tradicional de gestión de proyectos es más apropiada si no se involucran nuevas tecnologías o herramientas. La metodología ágil, al ser más flexible que la anterior, permite más espacio para experimentar más con la nueva tecnología.
- Riesgos y amenazas. Teniendo en cuenta la naturaleza rígida de la metodología tradicional, no es recomendable utilizar esta metodología. Sin embargo, los riesgos pueden abordarse antes en el enfoque ágil, parece ser una mejor opción en términos de gestión de riesgos.

- Recursos. El enfoque tradicional funciona mejor con equipos y proyectos grandes y complejos. Mientras que un equipo ágil generalmente consiste en un número limitado de miembros experimentados del equipo.
- La criticidad de un producto final depende mucho de la naturaleza de la metodología de gestión de proyectos elegida. Como el método tradicional implica una documentación, es muy adecuado para productos críticos en comparación con la metodología ágil de gestión de proyectos.

Estos factores se encasillan en marcos empresariales, entonces, la variabilidad en las necesidades genera un campo difuso para estos factores. Los ingenieros de software encargados, se han decidido por seleccionar los factores principales, enmarcarlos y así en conjunto generar modificaciones a las metodologías o combinarlas para crear otras.

3.6. Framework FOSS

El software libre y de código abierto (FOSS) es un software que se puede clasificar como software libre y de código abierto. FOSS mantiene los derechos de libertad civil del usuario del software, que son las 4 libertades esenciales, libertad de ejecutar (L0), estudiar (L1), redistribuir (L2) y distribuir (L3) [5]□. Es decir, es una cuestión de libertad, no de precio. FOSS no significa que "no es comercial". Un programa libre debe estar disponible para el uso comercial, la programación comercial y la distribución comercial [14], [15]□. Otros beneficios de usar FOSS pueden incluir menores costos de software, mayor seguridad y estabilidad (especialmente en lo que respecta al malware), proteger la privacidad, la educación y otorgar a los usuarios más control sobre su propio hardware y software [41]□. Para este tipo de especificaciones existen diferentes tipos de licencias [14], [15], [41]□.

Un framework es un artefacto de diseño e implementación cohesivos y representa un conjunto de clases, interfaces y patrones para resolver un grupo de problemas, que es un método popular para mejorar la eficiencia del desarrollo y reducir el costo de desarrollo. Define un diseño abstracto, proporciona un comportamiento común y permite al usuario insertar su comportamiento específico por subclasificación o complemento en implementaciones específicas [42]□. Existen varios frameworks de fuente completa y FOSS para ayudar en el desarrollo de aplicaciones, especialmente en el lado frontend y web.

3.7. Human Interface Guidelines (HIG)

Los HIG son una forma de guiar el trabajo de usabilidad para el usuario, tanto de diseño como de debates, sobre alternativas de diseño [43]□.

Los HIG proporcionan una guía explícita (presumiblemente la mejor práctica) sobre cuestiones de diseño de interfaz que permite a la comunidad orientarse hacia la "acción", escritos por y para desarrolladores de software como de diseñadores de interfaces, que contienen consejos sobre cómo usar de manera efectiva los elementos visuales (cuadros de diálogo, barras de herramientas, botones, etc.) que algunos programas ofrecen a los usuarios de ese software, además de ayudarlos en la creación de experiencias bellas y consistentes dentro de los sistemas operativos [43], [44]□. Los HIG han sido ya bastante comunes en los últimos años, tanto en el desarrollo comercial como en el libre y abierto. Se puede resaltar las importantes contribuciones de Sun Microsystems a GNOME HIG, el de la comunidad de KDE para KDE HIG, los aportes de Google LLC para Android o el de Apple Inc para sus diferentes dispositivos iOS, entre otros [43], [45], [46]□, en la cual en su mayoría ha sido en forma de “cultura racional”, en el que los miembros de proyectos de código abierto "intentan hacer que su comportamiento sea lógicamente plausible y siempre se eligen opciones tecnológicamente superiores en la toma de decisiones", permitiendo un medio eficaz para evitar discusiones largas, ya que actuan como autoridad sobre los cambios que deben de realizarse.

Se extrae entonces que no solo se definen elementos y principios de diseño específicos, sino que también inculcarán una filosofía que lo ayudará a decidir cuándo es apropiado desviarse de las directrices.

Para la comunidad de KDE, el KDE HIG ofrece un conjunto de recomendaciones para producir interfaces de usuario hermosas, utilizables y consistentes para aplicaciones convergentes de escritorio y móviles y widgets para espacio de trabajo. Teniendo como objetivo mejorar la experiencia de los usuarios al hacer que las interfaces de aplicaciones y widgets sean más consistentes y, por lo tanto, más intuitivas y fáciles de aprender [47]□. Se trae como relevante este HIG en específico, ya que a partir de éste es que se construye el UI/UX del framework.

Cada proyecto de software pasa por varias etapas de diseño, a través de un número interminable de decisiones antes de que se forme un producto mínimo viable o final. La experiencia del usuario (UX) y el diseño de la interfaz de usuario (UI) forman parte de este desarrollo, en la cual como conjunto siguen los lineamientos del HIG.

3.7.1. Experiencia de usuario (UX)

La experiencia de usuario (UX) es el proceso de diseño centrado en el ser humano se centra en una comprensión profunda de los usuarios, los factores emocionales y los factores desencadenantes, las capacidades y las limitaciones. Cómo se sienten acerca de cada interacción con lo que está frente a ellos en el momento en que la están usando. Las mejores prácticas promueven la mejora de la calidad de la interacción del usuario y la percepción de un producto. Tiene en cuenta todo un proceso de adopción de un producto, construcción y transmisión de una marca, usabilidad, accesibilidad, objetivos comerciales y confiabilidad de un producto o servicio. Teniendo como ejes principales ser útil, utilizable, deseable, findable, accesible, creíble.

3.7.2. Interfaz de usuario (UI)

Una interfaz de usuario (UI) es un conducto entre la interacción humana y la computadora, el espacio donde un usuario interactuará con una computadora o máquina para completar tareas. El propósito de una interfaz de usuario es permitir que un usuario controle efectivamente una computadora, máquina, o dispositivos electrónicos protátiles con la que está interactuando, pero también considera dispositivos sin interfaz como interfaces controladas por voz (VUI), diseños de conversación (chatbots) o interfaces basadas en gestos, y que se reciba retroalimentación para comunicar la finalización efectiva de las tareas. Una UI exitosa debe ser intuitiva (no requiere capacitación para operar), eficiente (no crea fricción adicional o innecesaria) y fácil de usar (fácil de usar). Con el enfoque en satisfacer los requisitos de usabilidad, maximizar la experiencia del usuario propuesta y desencadenar la satisfacción del usuario.

Para un mejor entendimiento de estos dos conceptos vamos a aclararlo usando un lenguaje universal, y logrando observar la Figura 7. Diferencias entre UI y UX. Lo más fácil es ir por la terminología que uno puede respaldar con la experiencia de la vida real con la que un cliente puede relacionarse, su forma de vivir el servicio que eventualmente puede proporcionar y viceversa.

La experiencia del usuario es una historia que comienza antes de saber que hay un problema por resolver. Mientras que la interfaz de usuario está traduciendo un problema humano ya resuelto en un lenguaje legible por máquina que es estéticamente agradable para el usuario [48]□.

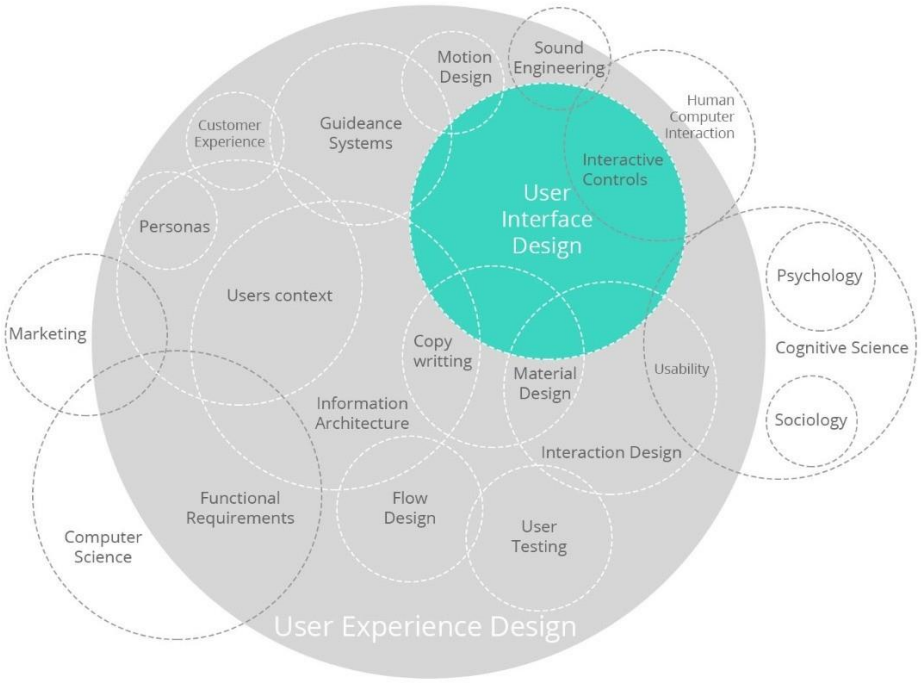

Figura 7. Diferencias entre UI y UX

En términos mínimos se podría extraer:

- UX: Propósito, proceso completo
- UI: Apariencia, proceso fragmentario

3.8. Información neurofisiológica

La información acerca del comportamiento cerebral a nivel fisiológico se ha ido registrando de diferentes formas, constituyendo índices directos e indirectos de la actividad neuronal, originadas respectivamente por las propias neuronas o hemodinámicamente.

En los últimos años se ha ido incursionando en el estudio de técnicas no invasivas para la medición de la actividad cerebral espontánea, de los cuales se encuentran la electroencefalografía (EEG), magnetoencefalografía (MEG), la resonancia magnética funcional (fMRI) y espectroscopia de infrarrojo cercano (NIRS), manifestando una señal rítmica claramente visible en los registros [1]□.

Principalmente este tipo de señales se almacena en diferentes formatos sean planos, hipertexto, JSON, compuestos, entre otros, esto significa que el tipo de los datos es no estructurado [49], [50]□, y el estándar para ello se encuentra en proceso, teniendo como primer el formato BIDS (del inglés *Brain Imaging Data Structure*) [51]□. Para ello siempre se debe de realizar el proceso de decodificación de la información, para así obtener los valores de interés, este proceso es diferente para cada formato y tipo de señal, cambiando los niveles de dificultad entre ellos.

4. Metodología

Para el desarrollo del framework libre para la creación de aplicaciones con perspectiva en información neurofisiológica se eligió trabajar bajo el marco de metodología ágil, que define los procesos de forma iterativa e incremental para desarrollar cualquier producto o administrar cualquier trabajo, produciendo un conjunto potencial de funcionalidades al final de cada iteración para así darle cumplimiento a los objetivos planteados, describiendose en la Figura 8. Diagrama de flujo de la metodología. En este marco de trabajo, se puede observar su planeación por etapas (cada color representa una etapa) siendo consecuentes a su vez con los lineamientos de los objetivos.

Se eligió esta metodología porque es apropiada también para proyectos pequeños o con pocos recursos debido a que evita la elaboración de documentación extensa, adicionalmente su organización con iteraciones cortas y micro incrementos permite mantener el control, evitar errores e incrementar las posibilidades de éxito.

En resumen, se eligió el marco de metodología ágil por ser una metodología libre, ágil y centrada en el producto.

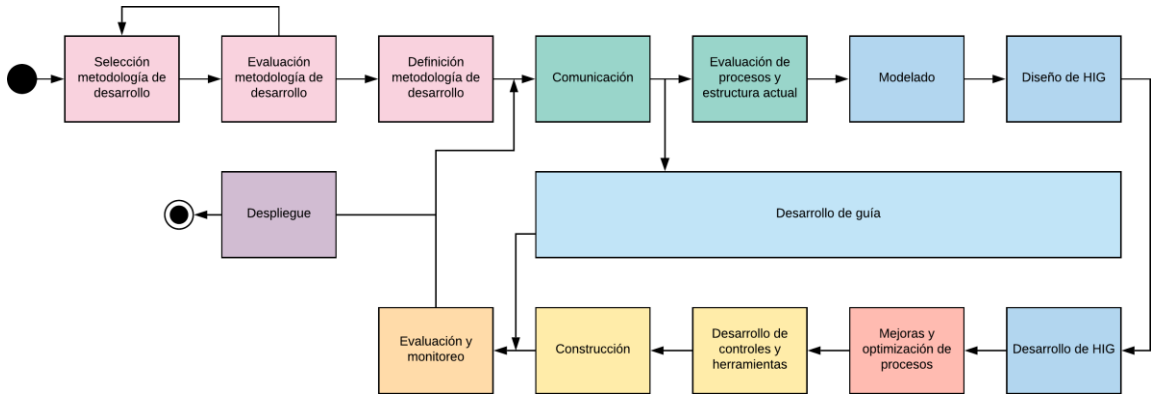

Figura 8. Diagrama de flujo de la metodología

Siguiendo los lineamientos de este flujo, se tienen varias etapas:

1. Metodología de desarrollo: Aquí se presentan 3 fases, en las cuales fueron guiadas a encontrar la adaptación acorde a la perspectiva de trabajo planteada.
2. Levantamiento de la información: Presentada por 2 fases, aquí se encuentran encapsulados el segundo y tercer objetivo específico, en dónde se encontró una revisión del estado del

arte y los temas viables, además de realizar el levantamiento de requerimientos, requisitos y sus implicaciones.

3. HIG (UI/UX): Aquí se presentan 3 fases, en las cuales se tuvieron en cuenta los criterios de interfaz de usuario (UI) y experiecia de usuario (UX) para la construcción del modelo, diseño y desarrollo.
4. Guía: En esta fase, la cual es transversal a la mayoría del flujo de trabajo, es donde se desarrolló la guía para el uso del framework.
5. Conjunto de controles y herramientas: Aquí se presentan 2 fases, en las cuales se realizó el desarrollo y construcción de los controles y herramientas empleadas, complementadas de una alteración en el intérprete para que en la fase de compilación a binario se puedan incluir todas las caracteristicas de las herramientas empleadas.
6. Evaluación y monitoreo: En esta fase se evaluó y monitoreó el correcto funcionamiento.
7. Despliegue: En esta fase es cuando todo el código fué suficientemente probado (fase anterior), se aprobó por el *Product Owner* y se dejó listo para su distribución en el entorno.

5. Resultados y análisis - Desarrollo del framework libre

5.1. Metodología de desarrollo de software desde la perspectiva de trabajo en el ámbito de la neuroingeniería

Aquí se hace necesario una forma de organización de la investigación en neuroingeniería dentro de los marcos académicos, integrando investigación y desarrollo, especialmente en grupos de investigación. En neuroingeniería el campo cada vez es más amplio y diverso, combinando aspectos del razonamiento matemático, fisiológico, clínico, la metodología de la ingeniería y los enfoques empíricos del método científico. La neuroingeniería es un espacio en expansión, donde sus diferentes ramas crecen continuamente en profundidad, uniendo a su vez diferentes campos de la ciencia. La forma de trabajo está evolucionando, la comunidad de software libre y abierta, la estandarización, el despliegue y apertura de datos, y los sistemas comerciales sobre los que se construyen nuevas herramientas y/o sistemas, están definiendo la nueva perspectiva de trabajo [52]□.

El objetivo de este marco de desarrollo es brindar beneficios significativos y prácticos a agentes internos (estudiantes, investigadores) como externos (usuarios) y hacerlos rápidamente. El desarrollo de las ideas y su viabilidad pueden lograr ser trascendentales, requiriéndose de esfuerzos para ser un éxito o un resultado negativo útil.

Para tener una metodología adaptada en el ámbito de la neuroingeniería, se es necesario evaluar las características más influyentes, de este modo se puede lograr crear, adaptar o escoger cierta metodología(s) que ayuden al proceso. De la experiencia obtenida, se extraen ciertas características a tener en cuenta para la selección de la metodología más apropiada:

- Esfuerzo colaborativo: Por ser investigadores y/o estudiantes sean *junior* o *senior*, la productividad, rendimiento y diversidad son agentes importantes dentro del equipo, reuniendo profesionales de distintas áreas, habilidades y alta competitividad.
- Adaptabilidad: Siendo del campo de la ciencia e investigación, como operación tecnológica generalizada, no solo tiene objetivos reales, sino también objetivos potenciales que se deben tomar en cuenta. Además, en el transcurso del tiempo las condiciones cambian.
- Alta calidad: Los diferentes aspectos de actividades, sea productos, procesos y/o servicios, cumpliendo con las especificaciones del diseño, funcionalidad y satisfacer las necesidades abordadas.
- Equipo y experiencia: En este campo se encuentran personas del diferentes áreas del conocimiento, la interdisciplinariedad es un factor común. Desde una perspectiva más local se encuentra poco personal trabajando en un solo proyecto, los conocimientos sobre el área se encuentran en formación principalmente, teniendo unos pocos experto en el tema, la

cual guían y forman a los otros integrantes, teniendo un abanico de conocimiento explícito y tácito.

- Desarrollo: En la neuroingeniería, como una abstracción entre ingeniería y neurociencias, la evolución de sus proyectos es continuo.
- Requerimientos de usuario: En su generalidad son detallados y definidos desde la construcción del proyecto, sin embargo, nuevos requerimientos se añaden en el transcurso del proceso, siendo estos emergentes con rápidos cambios.
- Envolvimiento con el cliente/interesado: Constantemente el cliente o interesado se encuentra altamente involucrado en el proyecto, siendo aveces parte del proceso.
- Escalabilidad: En neuroingeniería los proyectos pueden ser de baja o alta escalabilidad, de acuerdo a la solución planteada.
- Tamaño: Normalmente el equipo lo forman pequeños equipos de colaboradores.
- Documentación: Cada fase del proceso se documenta en detalle para eliminar cualquier malentendido o atajo. Esta documentación debe ser interna y externa, por ende una previa documentación de los desarrollos es necesaria.

Siendo estos las principales características en el campo de la neuroingeniería, abarcando también los grupos y laboratorios investigativos, siendo no únicos. Estas características surgen de la forma en que se constituye normalmente el área de neuroingeniería, la cual se centraliza en la academia desde sus centros investigativos, sin excluir la industria. A partir de ello, se encuentra que los proyectos son evolutivos, tomando parte del núcleo, y ahí es donde el modelo de ciclo de vida del proyecto toma relevancia. Siguiendo con los factores decisivos anteriormente mencionados, este marco no entra completamente en el marco de las metodologías tradicionales ni ágiles.

Se tiene que priorizar todos los diferentes proyectos, decidir qué hacer, cuándo y con qué recursos. ¿Se debe también permitir que los equipos trabajen con diferentes métodos y herramientas? ¿Cómo se supone que esto funcione?. Estas preocupaciones son justificables. Los equipos ágiles y tradicionales no solo trabajan de manera diferente, sino que hablan diferentes idiomas y tienen diferentes procesos [53]□, como se describió. Además, existen prejuicios que pueden afectar la colaboración entre los equipos ágiles y la administración, si no sabe cómo dirigirlos. Sin embargo, los beneficios de un enfoque híbrido son muy poderosos y convincentes.

5.1.1. Metodología Híbrida en Neuroingeniería: Adaptación y Definición

En la práctica, la mayoría de las metodologías ágiles tiene sus propias características únicas y comparte características comunes como funcionalidad y complejidad en comparación con las metodologías tradicionales. Las metodologías híbridas están diseñadas de acuerdo con principios similares: combinan las fortalezas tanto de las tradicionales como de las ágiles en un solo enfoque [39], [53]□. Así las metodologías híbridas tienen un 16% más de probabilidades de cambiar de velocidad rápidamente, un 17% más de probabilidades de producir trabajo de alta calidad y un 24% más de probabilidades de un despliegue [54]□.

El enfoque de la neuroingeniería, de su naturaleza evolutiva de la investigación, donde los grandes resultados suelen ser la composición de muchos pasos discretos, ha llevado a adoptar un marco híbrido.

Adaptándose a las características de la neuroingeniería, se logran extraer ciertos parámetros de las diferentes metodologías, como:

De las metodologías tradicionales:

- Documentación: Cada fase del proceso se documenta en detalle para eliminar cualquier malentendido o atajos.
- Equipo flexible: No siempre toma todo el tiempo y la atención de un equipo de desarrollo. Dependiendo de la fase, los miembros individuales del equipo pueden enfocarse en otros aspectos de su trabajo. Esto esencialmente permite un equipo de desarrollo más flexible (aunque menos enfocado).

De las metodologías ágiles:

- Adaptabilidad: El desarrollo ágil resalta la importancia de poder cambiar el diseño, la arquitectura, los requisitos y los entregables en el camino. Este es un enfoque inherentemente más flexible.
- Participación del cliente: Debido a los cambios constantes en el diseño y las unidades de entrega, el desarrollo de software ágil requiere una colaboración estrecha entre el cliente y el equipo de desarrollo. Esto significa garantizar la satisfacción del cliente en los entregables tempranos y la retroalimentación continua sobre los entregables. Si el cliente final no está disponible, el gerente de producto como experto en el mercado a menudo asume el rol del cliente.

- Desarrollo Lean: los valores de desarrollo ágil hacen que el producto final sea lo más simple posible. Si se puede lograr el mismo resultado final con dos pasos en lugar de cinco, el desarrollo ágil diseñará el software en consecuencia. El objetivo aquí es reducir la hinchazón y la complejidad que pueden afectar el rendimiento. En la práctica, esto significa encontrar cosas que se están haciendo que no producen valor y eliminarlas.
- Trabajo en equipo y colaborativo: Como mencionamos anteriormente, las metodologías ágiles valoran el trabajo en equipo y colaborativo casi por encima de todo lo demás. La metodología de desarrollo reconoce que ningún lenguaje de desarrollo es perfecto, y ningún desarrollador puede traer todo a la mesa. En cambio, los equipos deben evaluar continuamente cómo pueden ser más efectivos y ajustar el proyecto a medida que avanzan. La Programación Extrema (haciendo honor a su nombre) insiste en que los desarrolladores trabajan en pares sobre el principio de que dos cabezas son mejores que una.
- Prácticas de programación: Proveer prácticas detalladas y sugerencias con respecto a las buenas prácticas de programación, así como para diseñar un sistema desarrollado y probarlo. "Apunta a producir software de mayor calidad y mayor calidad de vida para el equipo de desarrollo" .
- Tiempo: El desarrollo ágil toma un enfoque muy diferente del tiempo durante los proyectos, dividiendo los proyectos en unidades de tiempo muy pequeñas, estos son los Sprints.
- Sostenibilidad: En lugar de presionar por plazos más rápidos a cambio de un proyecto sin terminar, el desarrollo ágil valora establecer un ritmo sostenible para el desarrollo de software.
- Pruebas: A diferencia de los enfoques en cascada donde hay una fase de prueba distinta, los enfoques ágiles insisten en realizar pruebas en todas las fases del proyecto. Los plazos de las metodologías ágiles son más cortos que en un proyecto tradicional, y las pruebas continuas están integradas como parte del proceso de desarrollo. El equipo debe proporcionar comentarios sobre la calidad del proyecto de forma regular.

Debido al marco de tiempo y al esfuerzo involucrado, el enfoque de este marco es iterativo y, por lo general, implica escribir código de producción o casi producción desde el primer día.

Por lo tanto, se define el modelo híbrido en neuroingeniería como uno que:

I. Genera avances en científicos e ingenieriles.

II. Divide los proyectos en partes pequeñas y alcanzables (discretizar el proyecto), donde cada uno pueda tener valor con propósito.

III. Genera guías de diseño para la estructuración y favorecer el desarrollo continuo.

IV. Trabaja colaborativamente, donde a su vez existe un trabajo en equipo intrínseco.

V. Creación de software en el marco de buenas prácticas de programación.

VI. Se aprovecha al máximo los datos para apoyar la investigación in vivo. Se enfatiza la difusión del conocimiento utilizando una colección flexible de diferentes enfoques.

VII. Toma la retrospectiva, inspección y adaptabilidad en la toma de decisiones para los cambios, dónde "fallar rápido para mejorar rápido".

VIII. De forma continua y sostenible, ejecutarse y ver resultados en el tiempo.

Las actividades de desarrollo se dividen en tres fases distintas, inicial, intermedia y final, siguiendo ciertos delineamientos de las metodologías anteriormente descritas.

En las actividades iniciales se realiza la recopilación de los requerimientos, la planificación, el presupuesto y la documentación del progreso del proyecto, tomado de las metodologías tradicionales. En la fase intermedia del proceso, cuando hay suficientes detalles para comenzar el desarrollo se construye el proyecto / producto se aplica Scrum/XP principalmente, un enfoque iterativo y adaptable para lograr el plan general que se presentó por primera vez en la primera fase [55]□. La fase final controla la frecuencia de lanzamiento formando puertas, un ciclo de lanzamiento de producción controlado e infrecuente que se rige por las políticas del grupo investigativo y/o organización y las limitaciones de infraestructura. Para todas las fases se debe limitar el tiempo dedicado a las actividades, para algunos autores [23], [28], [30], [39], [56]□, este tiempo debe realizarse acorde a las metodologías ágiles, no mayor a 4 semanas, el número de semanas cambia de acuerdo al tamaño y complejidad del proyecto. Esta metodología utiliza principios ágiles y técnicas de comunicación Scrum en el desarrollo de productos relacionados con las actividades diarias / semanales.

Se ofrece también recomendaciones para el equipo que enfrentan esta realidad, siendo así los valores propios de esta metodología y así aumentar la agilidad entre ellos:

a) Un equipo de neuroingeniería adecuado debe incluir a todas las personas necesarias para entregar el proyecto. Normalmente, esto significa que los desarrolladores, evaluadores y analistas de negocios trabajan colaborativamente para lograr un objetivo común, abriendo paso a la comunicación cara a cara.

b) La retroalimentación es crucial para el buen proyecto y permite introducir los cambios necesarios, la apertura permite que todas las inquietudes se aborden libremente.

c) El respeto es merecido por todos, cada miembro del equipo es igualmente valioso.

d) El coraje es necesario para decir la verdad sobre los problemas, estimaciones y fallas, no hay por qué temer, no se trabaja solo.

e) Pasar demasiado tiempo por adelantado no aumentará la calidad del lanzamiento; por el contrario, es un desperdicio.

f) La simplicidad y el enfoque en una sola cosa a la vez es la clave para un proyecto exitoso, al igual que maximizar el trabajo no realizado.

g) La documentación es un proxy deficiente para el proyecto y, por lo tanto, cualquier documento creado debería ser suficiente para introducir el área problemática y permitir que comience el trabajo de planificación, desarrollo de alto nivel y mantenimiento.

5.1.2. Metodología Híbrida en Neuroingeniería: Fases

La metodología propuesta aquí se considera como un subconjunto de metodologías tradicionales y ágiles (SCRUM / XP) Figura 9. Metodología Híbrida en Neuroingeniería , reformado a base de experiencia en prácticas de industria y profesionales del área. Esta metodología se enfatiza en la agilidad y se enfrenta a entornos de desarrollo dinámicos y estables. La metodología en neuroingeniería presentada tiene el siguiente conjunto de fases: fase previa, fase de desarrollo y fase posterior, cada una se divide en subfases con sus pasos.

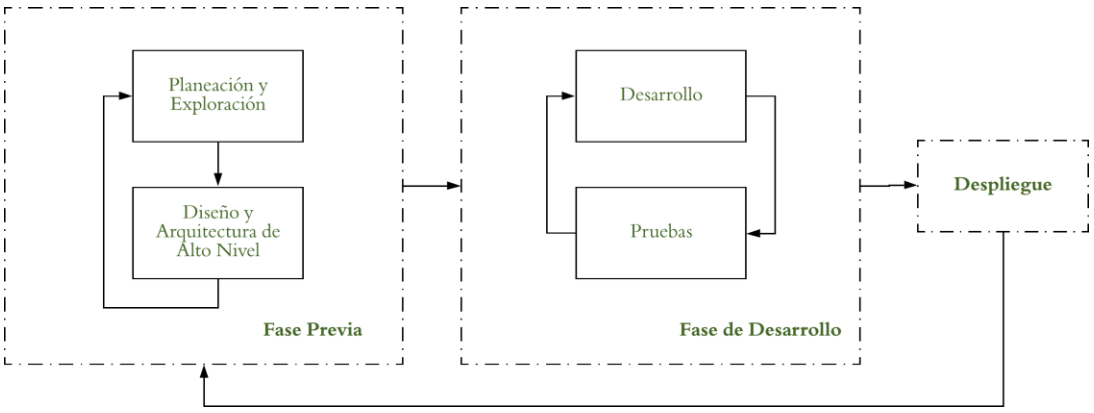

Figura 9. Metodología Híbrida en Neuroingeniería

5.1.2.1. Fase Previa

5.1.2.1.1. Planeación y exploración

Tras la comunicación inicial y el acuerdo con el cliente o asesor, el trabajo real de desarrollar un proyecto comienza con la planificación. La definición del nuevo producto se define a partir de los requerimientos globales junto con la estimación y la prioridad para cada tarea de esos requerimientos. El diseño de alto nivel o la arquitectura del sistema también se desarrolla en esta fase. Además, se identifican y cuantifican los atributos de calidad del software y su escala o unidad de medida, en caso tal de requerirse [39]□ (aplica cuando se va a construir un sistema completo). Si el proyecto a desarrollar es nuevo, entonces en esta fase se analiza si se requiere capacitación de personal y un análisis más amplio en términos de arquitectura y diseño, por otro lado, si el sistema que se está desarrollando es el existente y se está mejorando, esta fase consiste en un análisis limitado [39], [57]□. El cliente, los desarrolladores y el líder del equipo colaboran con el desarrollo de las actividades de esta fase. Los pasos generales en la fase de planificación son los siguientes:

I. Comunicación y levantamiento de requerimientos (funcionales y no funcionales), tanto su análisis como las especificaciones a base de la visión del proyecto, mediante HU.
II. Identificación y selección de herramientas, tecnologías y personal requerido para el desarrollo del proyecto, además del estándar de codificación.
III. Identificación y cuantificación de los atributos de calidad del software y su escala o unidad de medición.
IV. Capacitación en herramientas, tecnología o prácticas si es necesario.
V. Creación del backlog del producto (del inglés *Product Backlog*) con los requerimientos generales.
VI. Estimación de costos y esfuerzos para cada tarea en la lista del backlog del producto.

Como resultado de la subfase de planeación y exploración, tanto el cliente como el equipo de desarrollo deben compartir una visión común del proyecto, saber cuál es el resultado esperado y tener una idea de cómo se implementará el producto terminado [33], [55]□. Se debe tener así ciertos documentos como:

1) Visión, proporcionando una descripción breve y consisa del proyecto (por qué es necesario, caracteristicas previstas, qué se espera tener, entre otros).
2) Especificación del proyecto. Conteniendo las especificaciones funcionales y técnicas.

3) Backlog del producto, conteniendo la lista priorizada de ls requerimientos generales.
4) Versión inicial del plan, proporcionando aspectos más técnicos del proyecto, además de contener la información de las iteraciones pasadas, presentados en una tabla con sus bjetivos, fechas de inicio y fin, velocidad y eficiencia.
5) Documento de la guía de estilos de desarrollo, cubriendo las prácticas y estándares de codificación adecuados para las tecnologías a emplear.

5.1.2.1.2. Diseño y arquitectura de alto nivel

Aquí se construye la arquitectura inicial del sistema y el diseño de alto nivel (el diseño general de todo el proyecto de forma concisa) para implementar los requerimientos de la lista del backlog del producto. Además, la identificación de cambios y el refinamiento de la arquitectura para implementar nuevos requerimientos y el análisis de riesgos también se incluyen.

Al finalizar esta fase, la cual es una fase iterativa, se debe tener las historias de usuario seleccionadas despejadas de cualquier ambigüedad. Tanto el cliente como los desarrolladores deben saber qué funcionalidades son las más importantes para el cliente y cuáles son las tareas que les corresponden. Se debe tener un solo documento del plan de iteración que proporciona las tarjetas de tareas, estimaciones de tareas, así como las fechas de publicación de destino [55]□.

5.1.2.2. Fase de Desarrollo

5.1.2.2.1. Desarrollo

En esta subfase se presenta el gran cambio. El trabajo en el producto se realiza en ciclos iterativos (Sprint). El equipo de desarrollo junto con los demás interesados determinan si el tiempo de desarrollo, la calidad o las funcionalidades del producto se cumplen como requerimientos y, al final de las iteraciones, se entrega el producto.

Se sigue un proceso macro, la cual consiste en:

I. Revise los planes de lanzamiento en la reunión del equipo.
II. Conformidad con la distribución, revisión y ajuste del estándar.
III. Varios Sprints hasta la entrega del producto.

Durante cada Sprint, los desarrolladores trabajan en la implementación de historias de usuarios a partir de la acumulación de iteraciones para este Sprint, el desarrollo debe seguir un estándar de codificación. Los Sprints deben tener una duración constante, para que la planificación y medición del progreso sea simple y confiable. La duración recomendada es de 1 a 3 semanas, de acuerdo a la complejidad de esas historias de usuario / tareas. El Sprint debe finalizar después de un período de tiempo específico, incluso si no se realizan todas las tareas.

Las reuniones se realizan dos veces por semana como mínimo, preferiblemente cada dos días. Si es posible, el cliente/interesado debe estar presente durante el mismo. Alternativamente, en caso de Sprints de más de una semana, se podría celebrar una reunión semanal al final de la semana. Como resultado del Sprint (la fase de desarrollo), se debe obtener un software de trabajo listo para ser probado. Se debe crear un conjunto de documentos necesarios, detallando cómo funcionan ciertos aspectos del proyecto, qué tabla de base de datos almacena qué datos y finalmente un diagrama de actividad y de proceso de la función realizada.

Los pasos generales de un Sprint son:

I. Reunión de planificación de Sprint para asignar la lista del backlog.

II. Reunión de scrum para rastrear el progreso y resolver problemas

III. Revisión por pares para reducir el riesgo y la incertidumbre.

IV. Actualización de esfuerzo y costo basado en nuevos cambios.

V. Agregar / eliminar / corregir mejoras para aumentar la productividad y disminuir la posibilidad de retrasos en la entrega.

VI. Integración de nuevo incremento al final del Sprint.

VII. Reunión de revisión de Sprint al final de cada Sprint para discutir los resultados y averiguar el alcance de la mejora.

Las actividades anteriores como comunicación y diseño, análisis de requerimientos, especificación y creación de prototipos se realizan de forma iterativa en el Sprint y esta iteración continúa hasta que el cliente/interesado llega a un acuerdo con el equipo desarrollador. Cuando el cliente/interesado está de acuerdo con el desarrollador, el resto de las actividades de desarrollo como la implementación, las pruebas y la entrega del nuevo incremento se realizan de manera secuencial, como las metodologías tradicionales.

5.1.2.2.2. Pruebas

La subfase de pruebas ocurre simultáneamente a la subfase de desarrollo. El proyecto se prueba constantemente utilizando pruebas unitarias (preferiblemente automatizadas, en caso de tenerse al alcance), así como manualmente por los desarrolladores. Sin embargo, después del final del Sprint, el proyecto obtenido debe probarse con más profundidad. Durante esta subfase, se procura encontrar e informar cualquier deficiencia que deba solucionarse lo antes posible. Después de corregir estos errores menores, se ejecutan las pruebas de aceptación creadas por el cliente/interesado. Si se pasan las pruebas de aceptación, el proyecto está listo para ser implementado [55]□.

La prueba de software es un proceso muy complicado. No es independiente, está limitado y condicionado por el proceso de desarrollo.

5.1.2.3. Fase Posterior o Despliegue

Al final de cada Sprint o cuando no haya más requerimientos que implementar para la próxima versión. Se concluye el trabajo sobre el incremento y se comienzan los preparativos para las siguientes iteraciones. En esta fase, después de cada Sprint, se debe llevar a cabo una reunión de revisión del Sprint realizado. El producto o la tarea debe ser mostrado al cliente/interesado, incluso si no ha pasado la prueba de aceptación; la obtención de los comentarios del cliente/interesado es crucial para un mayor desarrollo. Durante esta fase se realiza la reunión retrospectiva del Sprint. Permite que el equipo discuta los éxitos y fracasos de las iteraciones, o posibles mejoras. La fase posterior o de despliegue completa un ciclo de la gestión del proyecto. Los comentarios y la información recopilada durante esta fase son la base para el siguiente ciclo de la gestión del proyecto, comenzando con la reunión de planificación para el próximo Sprint. En caso de finalizar el proyecto, se realizan pruebas de integración y el producto se prepara para su distribución general [39], [55]□□.

5.1.3. Metodología Híbrida en Neuroingeniería: Roles

Las funciones y responsabilidades de los integrantes del equipo se basan en las funciones del equipo presentes en Scrum [16]□, y aunque esta metodología de desarrollo no sea Scrum, su parte híbrida lo define. Teniendo los siguientes roles, nombrados en el idioma inglés:

5.1.3.1. Product Owner

El Product Owner es la persona responsable de maximizar el valor del negocio para el proyecto. Este rol es responsable de articular los requisitos del cliente y de mantener la justificación del negocio del proyecto. El Product Owner representa la voz del cliente.

5.1.3.2. Srum Master, o en su defecto Coach

El Scrum Master o Coach es un facilitador que asegura que el Equipo Scrum esté dotado de un ambiente propicio para completar con éxito el desarrollo del producto. El Scrum Master o Coach guía, facilita y enseña las prácticas de la metodología a todos los participantes en el proyecto, elimina los impedimentos que enfrenta el equipo y se asegura de que se estén siguiendo los procesos de la metodología.

Debe tenerse en cuenta que el rol de Scrum Master o Coach es muy diferente a la función que desempeña el Project Manager en las metodologías tradicionales, en el que el Project Manager trabaja como gerente o líder del mismo. El Scrum Master o Coach sólo trabaja como un facilitador y está en el mismo nivel jerárquico que cualquier otra persona en el Equipo Scrum.

Cualquier persona del Equipo Scrum que aprenda a facilitar proyectos dentro de esta metodología puede convertirse en el Scrum Master de un proyecto o sprint.

5.1.3.3. Equipo Scrum

El Equipo Scrum es un grupo o equipo de personas responsables de entender los requerimientos del negocio especificados por el Product Owner, de estimar las historias de usuarios, los diseños de software y de la creación final de los entregables del proyecto.

5.1.3.4. Tracker

El Tracker es la persona cuya tarea principal es monitorear el progreso del desarrollo de software y detectar todos los problemas en él.

5.2. Levantamiento de información necesaria para el sistema

5.2.1. Levantamiento de la información

Teniendo en cuenta que la metodología utilizada para el desarrollo de este proyecto, la metodología híbrida en Neuroingeniería, se siguió a cabalidad con sus lineamientos, estableciendose en esta

sección la Fase previa de la metodología, realizando reuniones continuas con el Product Owner para obtener la información que se encontrará a continuación en la Tabla 4. Historias de usuario. y que muestra una recopilación final de las diferentes necesidades que se tiene en forma de tarjeta.

Tabla 4. Historias de usuario

	Enunciado de la historia				Criterios de aceptación			
Identificador (ID) de la historia	Rol	Característica / Funcionalidad	Razón / Resultado	Número (#) de escenario	Criterio de aceptación (Título)	Contexto	Evento	Resultado / Comportamiento esperado
HU-001-001	Como usuario general	Necesito crear o abrir un proyecto	Con la finalidad de tener dentro del proyecto diferentes archivos de señales y procesos	1	Crear proyecto	En caso de que se desee crear un proyecto en blanco ubicado en una ruta especifica	Cuando se ingrese el nombre del archivo/proyecto	A continuación del nombre del tipo, se mostrará en una columna continua el número de habitaciones disponibles.
				2	Abrir proyecto	En caso de que el proyecto exista	Cuando se despliegue el listado de los proyectos existentes en una determian da dirección	A continuación el archivo .conf seleccionado será cargado en la plataforma
				3	Abrir proyecto no compatible	En caso de que el proyecto exista pero este no sea compatible		La plataforma no mostrará los archivos diferentes al tipo de archivo .conf
HU-001-002	Como usuario general	Necesito cargar archivo(s) de señal tipo EEG dentro de un proyecto	Con la finalidad de agregar diferentes señales al proyecto	1	Formato de archivo de señal no compatible	En caso de que el formato no sea de tipo .set, .vhdr, .edf, .bdf o .egi	Cuando se realice el listado de los archivos de señal	El sistema no mostrará archivos con formatos ddiferentes a los establecidos
				2	Seleccionar archivos de señal ubicados en diferentes direcciones	En caso de que el archivo de señal se encuentre en diferente dirección		El sistema permitirá la navegación entre loss archivos del sistema guardando la lista de los archivos de señal previamente seleccionadas
				3	Deseleccionar archivos de señal	En caso de que el archivo de señal ya no se requiera cargar		El sistema permitirá la eliminación de la selección del archivo de señal
				4	Seleccionar archivos de señal	En caso de que el archivo de señal exista		El sistema permitirá la selección del archivo de señal
				5	Cargar archivos de señal seleccionados	En caso de que los archivos de señal ya estén	Cuando se acepte	El sistema cargará las señales inmediatamente en diferentes

						seleccionados		lineas de procesamiento
HU-001-003	Como usuario general	Necesito graficar la señal	Con la finalidad de observar la señal en el tiempo de duración	1	Gráfica automática	En caso que el archivo de señal se encuentre en el proyecto pueda seleccionar qué archivo de señal se va a graficar	Cuando se tengan cargadas todos los archivos de señal deseados en el proyecto	El sistema dará la opción de seleccionar el archivo de señal a graficar
				2	Gráfica de 1 solo archivo de señal	En caso que en el proyecto sólo se encuentre cargado un solo archivo de señal		El sistema graficara inmediatamente ese archivo de señal
HU-001-004	Como usuario general	Necesito generar reporte	Con la finalidad de tener los resultados en un archivo PDF	1	Generar reporte teniendo archivos de señal	En caso de que se desee tener un reporte en archivo externo	Cuando se seleccione la opción de reporte	El sistema generará un reporte en PDF con los archivos de señal del proyecto y con las funcionalidades pre-establecidas en el backend
				2	Generar reporte sin tener archivos de señal	En caso de que se desee tener un reporte en archivo externo pero sin archivos de señal		El sistema generará un reporte en PDF con la información del proyecto creado
HU-002-001	Como desarrollador	Necesito recibir los valores de la(s) señales	Con la finalidad de realizar procesos con estos archivos	1	Valores de señales listas para ser procesadas	N/A	Cuando se tenga en existencia valores de señal	El sistema entregará todos los valores de señal que se tengan por proyecto
				2	Sin valores de señal		Cuando no se tenga en existencia valores de señal	El sistema no entregará error, informará en un popup que no se encuentran señales cargadas
HU-002-002	Como desarrollador	Necesito entregar una salida	Con la finalidad de que el usuario general pueda ver y/o generar reporte	1	Tipo de salida de un solo valor numerico	En caso de que la salida del procesamiento sea de tipo entero o flotante	Cuando se realice algún tipo de procesamiento sobre los valores de señal	El sistema recibirá un valor la cual almacenará y mostrará
				2	Tipo de salida tipo array	En caso de que la salida del procesamiento sea de tipo array		El sistema recibirá un valor la cual almacenará

HU-002-003	Como desarrollador	Necesito salida de valores por cada proceso	Con la finalidad de tener los archivos de salidad de cada etapa de procesamiento	1	Archivo de salida del segmento de procesamiento	N/A	Cuando se tenga al menos una etapa de procesamiento	El sistema almacenará cada salida en forma de archivo de cada etapa de procesameinto

5.2.2. Análisis de la información

Se establecieron dos roles principales, el rol de desarrollador y el rol de usuario general, la cual su diferencia radica en la forma de interacción con el framework, el desarrollador es el que modificará el código y el usuario general es el que tendrá la interaccción final con la plataforma ya desplegada. A cada rol se le asignaron diferentes funcionalidades, haciendo una referencia a la descripción de las actividades y servicios que el framework debe proveer, estando vinculado principalmente con las entradas, las salidas de los procesos y los datos a almacenar en el sistema, describiendo así lo que el framework debe hacer, la cual es dependiente de la arquitectura establecida, logrando distinguir y comunicar los componentes de cada escenario.

5.2.3. Dominio de la arquitectura

El framework libre para la creación de aplicaciones con perspectiva en información neurofisiológica, no posee un límite de usuarios como tal, por ser framework permite es construcción, sin embargo, en su modelo más básico la cual el usuario podría interacturar se encontraría utilizado por una cantidad limitada de usuarios, la cual además de crear proyectos, pueden ingresar información, visualizarla y generar reportes. Para el framework es importante tener las características básicas que el usuario final podría emplear.

El framework se construyó teniendo en cuenta los dos componentes, *backend* y *frontend*, en la cual para cada una de estas dos hay integración entre sí pero que en su independencia poseen arquitecturas diferentes.

La descripción de esta arquitectura va dirigida a los desarrolladores que deseen colaborar en el proyecto.

5.2.3.1. Alcance y restricciones

5.2.3.1.1. Alcance

Se desarrolló la etapa inicial del framework, la cual permitirá la creación de proyectos, la carga de información neurofisiológica, visualización y la generación de reportes.

Para cada una de ellas se planteó un esquema de flujo de información. Adicionalmente, el framework cuenta con las siguientes funcionalidades:

- Módulo de gestión de proyectos
- Módulo de gestión de carga de archivos
- Módulo de gestión de procesos y reportes

El framework está basado de forma que permanezca multiplataforma y convergente, la cual se deberá tener en cuenta en el momento de la selección de las herramientas y tecnologías para la colaboración continua de éste.

5.2.3.1.2. Restricciones

De acuerdo a las historias de usuario establecidas, se establecen las siguientes restricciones:

- El framework en su versión final de usuario básica no tiene la posibilidad de generar reportes comparativos
- El framework sólo aceptará señales biológicas tipo EEG en su primera versión, formato *.cnt*
- El framework sólo otorgará el módulo de gráfica 2D sin posibilidad de sub graficas
- El framework no manejará certificados de seguridad al encontrarse en un servidor local
- El esquema de seguridad y protección corresponderá al brindado por el usuario final a nivel de hardware y a nivel de software por el desarrollador que lo empleará en sus proyectos

5.2.3.2. Metas de la arquitectura

- Brindar una solución flexible y de fácil construcción

- Crear componentes que permitan una fácil manipulación de manera que cuando se necesite hacer modificaciones no afecte la totalidad de la aplicación
- Ser multiplataforma y convergente. El framework debe de ser *responsive* sobre cualquier pantalla, además de correr sobre las principales plataformas
- Permitir la escalabilidad y mejoramiento continuo del sistema

5.2.3.3. Dependencias y suposiciones

Los desarrolladores que emplearán el framework para el desarrollo de sus aplicaciones con perspectiva en información neurofisiológica deberán tener instalado las últimas versiones de desarrollo de KDE y Qt, además de estar sobre una plataforma UNIX/Linux. Las dependencias especificas dependen de cada tecnología y herramienta empleada, entre sus principales son las presentes en la Figura 11. Tecnologías y herramientas para la construcción del framework.

Dentro de las suposiciones se encuentran los requisitos significativos de la arquitectura, la cuales son: El desarrollo colaborativo debe tener en cuenta solo herramientas compatibles FOSS y debe proveer soporte para futuros requerimientos.

5.2.3.4. Mecanismo arquitectónico de trabajo

La arquitectura utilizada para el componente *Backend* corresponde al modelo MVC (Modelo Vista Controlador) [58]□ el cual permite la separación de la capa de negocio y datos de la interfaz de usuario. Esta Vista es realmente el componente Frontend, la cual corresponde al modelo multicapa [59], [60]□, donde cada capa tiene un papel amplio dentro de la aplicación y se comunica con otras capas a través de interfaces bien definidas, además, cada capa permite ser procesos separados e incluso sistemas físicamente separados, esto es lo que permite la interacción entre el *Backend* y el *Frontend*, como se obversa en la Figura 10. Resumen arquitectura empleada. Se definió de esta manera principalmente por la naturaleza de las principales tecnologías usadas (Qt/pyside 2 - QML).

Estas dos arquitecturas se comunican, permitiendo al framework ser multiplataforma y convergente. La arquitectura promueve una alta cohesión y un bajo acoplamiento, ambas características del buen diseño de software.

La persistencia de la información será realizada por un motor de base de datos relacionales como lo es SqLite, además de tener componentes no relaciones pero estáticos.

Para el patrón de diseño se empleó el paradigma de Programación Orientada a Objetos (POO) y la programación funcional (PF), esta combinación de patrones y anti patrones de diseño permite a este framework ser colaborativo con facilidad dentro de la comunidad neuroingenieríl, en la cual se es muy empleada, además de otorgar una correcta representación del diseño en la construcción de frameworks [61]□, en la cual se tiene muy en cuenta aspectos como diseño, implementación, instanciamiento y mantenimiento [61]□.

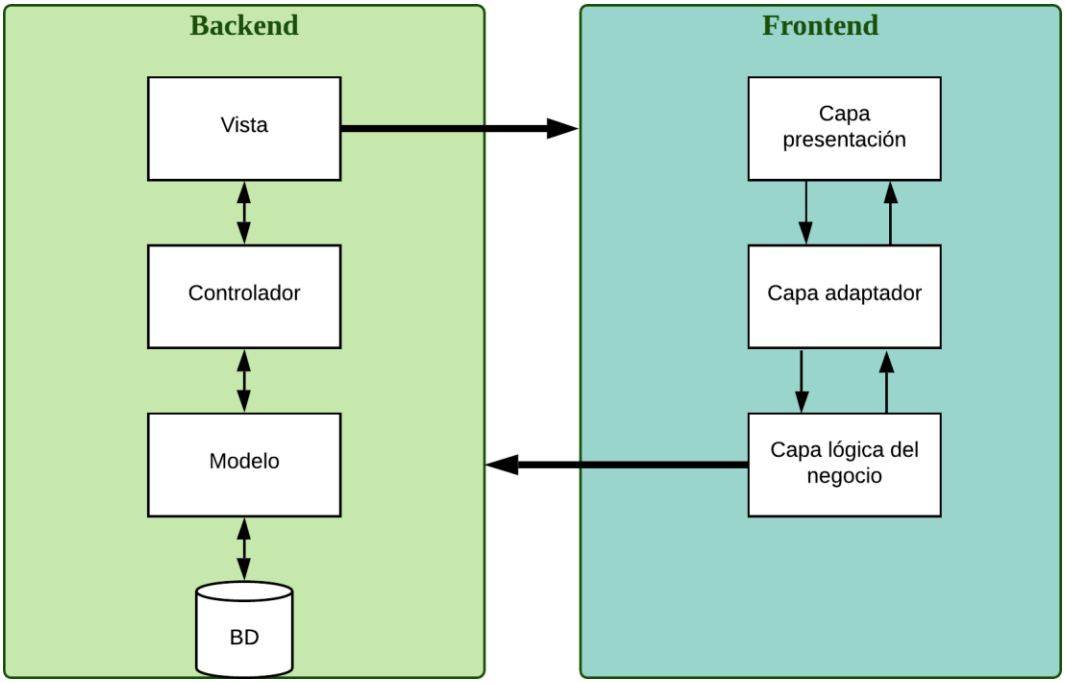

Figura 10. Resumen arquitectura empleada

5.2.4. Herramientas y tecnologías

En el desarrollo del proyecto se establecieron ciertas tecnologías para su construcción de núcleo, partiendo de los requerimientos iniciales y de las herramientas necesarias para la existencia de las características principales como ser multiplataforma y la convergencia, éstas fueron agrupadas por su característica principal, como se observa en la Figura 11. Tecnologías y herramientas para la construcción del framework. , en la Tabla 5. Tecnologías y herramientas empleadas. se encuentra una breve descripción del criterio final de selección. Por ser este proyecto pensado en ser libre, teniendo una licencia LGPL v3, se tomó en cuenta en el momento de la selección de las herramientas y tecnologías entérminos de compatibilidad, siendo compatibles todas ellas, para más información ver anexo 1-licenses_compatibility.

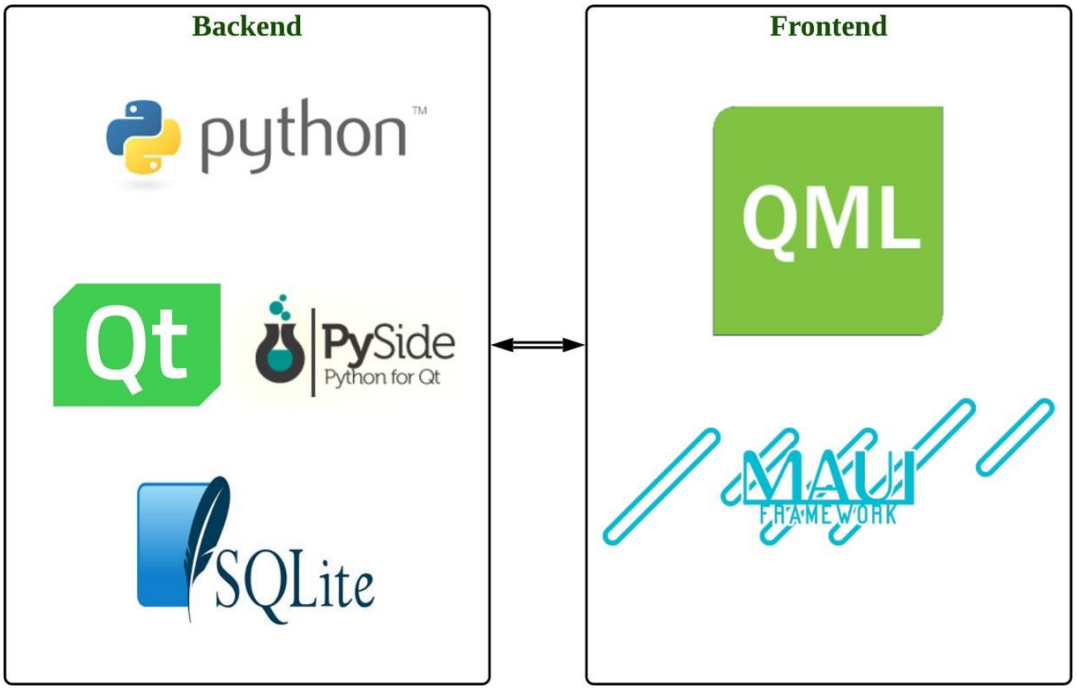

Figura 11. Tecnologías y herramientas para la construcción del framework

Tabla 5. Tecnologías y herramientas empleadas

Nombre	Razón	Licencia
Python 3.x	Actualmente es el más frecuente, maduro y compatible entre los lenguajes de programación en el área de procesamiento de señales/imágenes, aprendizaje automático, backend, entre otros.	PSFL
Qt / Pyside2	Es una biblioteca para Python que hace de *binding* para las herramientas de interfaz gráfica de usuario (GUI) del framework Qt, siendo el más estable, continuamente mantenido y soportado por Qt.	LGPL v2.1
SQLite	Es el sistema de gestión de bases de datos relacional, de baja densidad, y es muy adecuado para los sistemas integrados, por ende ideal para poder ser multiplataforma.	Public domain
QML	Es un lenguaje declarativo y dinámico, creado para diseñar aplicaciones aplicaciones centradas en la interfaz de usuario, basado en JavaScript además de permitir la integración de éste, adoptado por diferentes aplicaciones *open source*.	LGPL v3 *
MAUI Kit	Un framework front-end modular y libre para desarrollar experiencias de usuario rápidas y potentes.	LGPL v3

5.3. Estructura de estilo - HIG

La esencia de este proyecto es su naturaleza libre, por ende, se aspira que en un futuro la continuación del poryecto será en colaboración de la comunidad de software libre y abierta, personas con criterio FOSS que van a colaborar, adaptar y/o emplear sea en sus proyectos propios en general. Para una colaboración se es necesario llevar una convención de codificación y estilo, teniendo como distribución principal dentro del desarrollo lo mejor de las mejores prácticas BOBP (del inglés *Best of the Best Practices*). El objetivo de tener pautas de estilo es tener un vocabulario común de codificación para que las personas puedan concentrarse en lo que estás diciendo en lugar de en cómo lo estás diciendo.

Dentro del marco de este proyecto se creó la guía de estilo, la cual se encuentra en el Anexo 2-Style_Guideline_Nebula. Basada en lass mejores prácticas de la comunidad Python, de las propuestas de desarrollo creadas por Google y el estandar BIDS creado por la comunidad de neuroingeniería.

5.3.1. HIG

Para la realización de este proyecto y en conjunto con la comunidad de KDE, la guía del HIG construido otorga beneficios como:

- Los usuarios aprenderán a usar su aplicación más rápido, ya que comparte elementos comunes con los que ya están familiarizados.
- Los usuarios realizarán tareas más rápidamente, porque tendrá un diseño de interfaz directo que no es confuso o difícil.
- Su aplicación aparecerá nativa del sistema operativo a usar y compartirá el mismo aspecto elegante que las aplicaciones predeterminadas dentro de los sistemas UNIX/LINUX.
- Su aplicación será más fácil de documentar, porque un comportamiento esperado no requiere explicación.
- La cantidad de soporte que deberá proporcionar, incluidos los errores archivados, se reducirá (por los motivos anteriores).

Para alcanzarlos, las pautas propuestas por el HIG cubren elementos básicos de la interfaz, cómo usarlos y armarlos de manera efectiva, y cómo hacer que la aplicación se integre bien con la pantalla, siendo sensible a su tamaño.

Lo más importante para recordar es que seguir estas pautas del HIG propuesto en el framework facilitará el diseño de una nueva aplicación, no más difícil, teniendo como ventaja que viene ya de forma predeterminada.

El diseño puede ser visto como una plantilla, en la cual se acceden utilidades, otorgando bastante facilidad en la creación de aplicaciones, ya que entrega los componentes principales que se deben tener en un aplicación con perspectiva en información neurofisiológica, la cual el desarrollador sólo tendrá que instanciarla y/o editarla, así obtendrá su funcionamiento.

Además, este entrega una interacción y visualización similar entre las diferentes aplicaciones que se puedan desarrollar con el framework.

Para entender estos componentes, primero es necesario verlo como plantilla desplegable, en la cual cada pliego puede ser una vista/pantalla, como se observa en la Figura 12. Plantilla base.

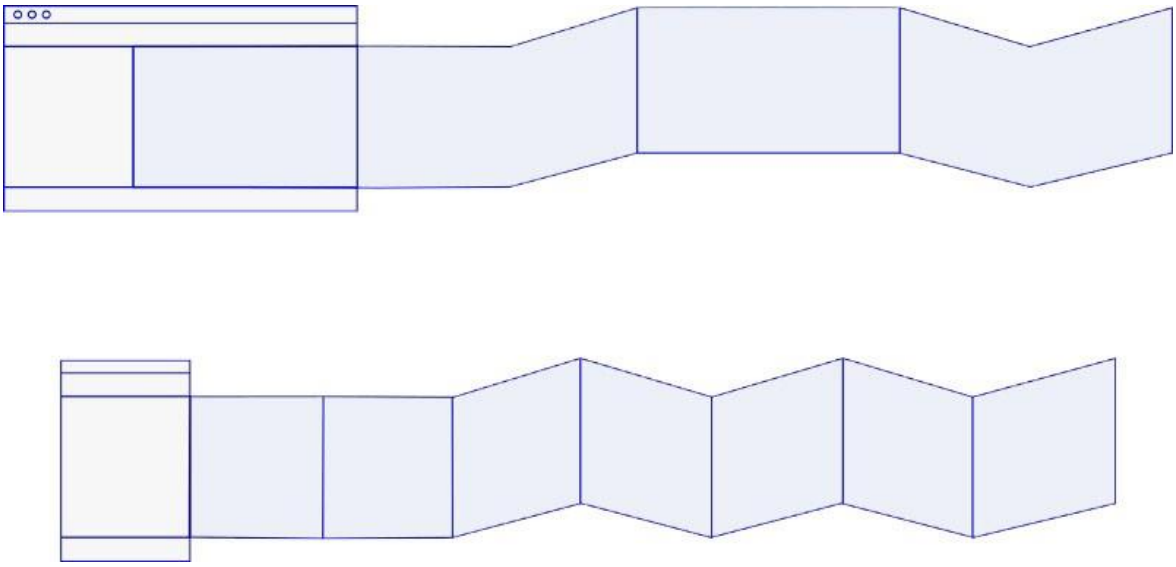

Figura 12. Plantilla base

La plataforma contiene 5 componentes principales:

- **headBar**: Barra horizontal ubicada en la parte superior o cabeza, dedicada a contener herramientas que se consideren fijas y/o permanentes.
- **ActionSideBar**: Barra vertical ubicada en el lateral izquierdo, dedicada a contener los componentes adicionales, la cual se crean a medida, estos no son fijos, varían de acuerdo a la construcción.

- **Page**: Componente de visualizacción y/o interacción, puede ser tomado como pila o como carrusel de pantallas, se crean a medida que se le hace el llamado.
- **Dialog**: Despliegue de información en forma de ventana emergente, activa durante cierto tiempo (~2 s) o bajo una acción externa.
- **FileDialog**: Cuadro de diálogo modal de documento, bloquea al usuario fuera de la ventana principal, permitiendo también el uso de otras ventanas de documentos en la misma aplicación.

Estos componentes permiten la interacción con las funcionalidades posibles, creados de manera en la cual puedan adaptarse al entorno de pantalla en el que se encuentren, teniendo una transformación dinámica cuando los cambios son extremos, adaptandose a cada medida, ver Figura 13. Mockup componentes principales y Figura 14. Mockup componentes secundarios.

El diseño y construcción reducido a 5 componentes es por dos principales motivos, mantener la consistencia en la interación con el usuario, teniendo una experiencia de usuario intuitiva, y facilidad en el desarrollo por parte del programador que emplea el framework.

En términos de interacción con cada componente, este se construyó para no limitarse a clicks, sino también a deslizamientos, tanto verticales como horizontales, usados principalmente en la interacción entre vistas/pantallas.

Cada uno de estos componentes fué construido de manera independiente, aislada, pero con capacidad de interactuar entre ellos, siendo posible entonces modificar cada uno de ellos, teniendo en cuenta preservar la sensibilidad al cambio.

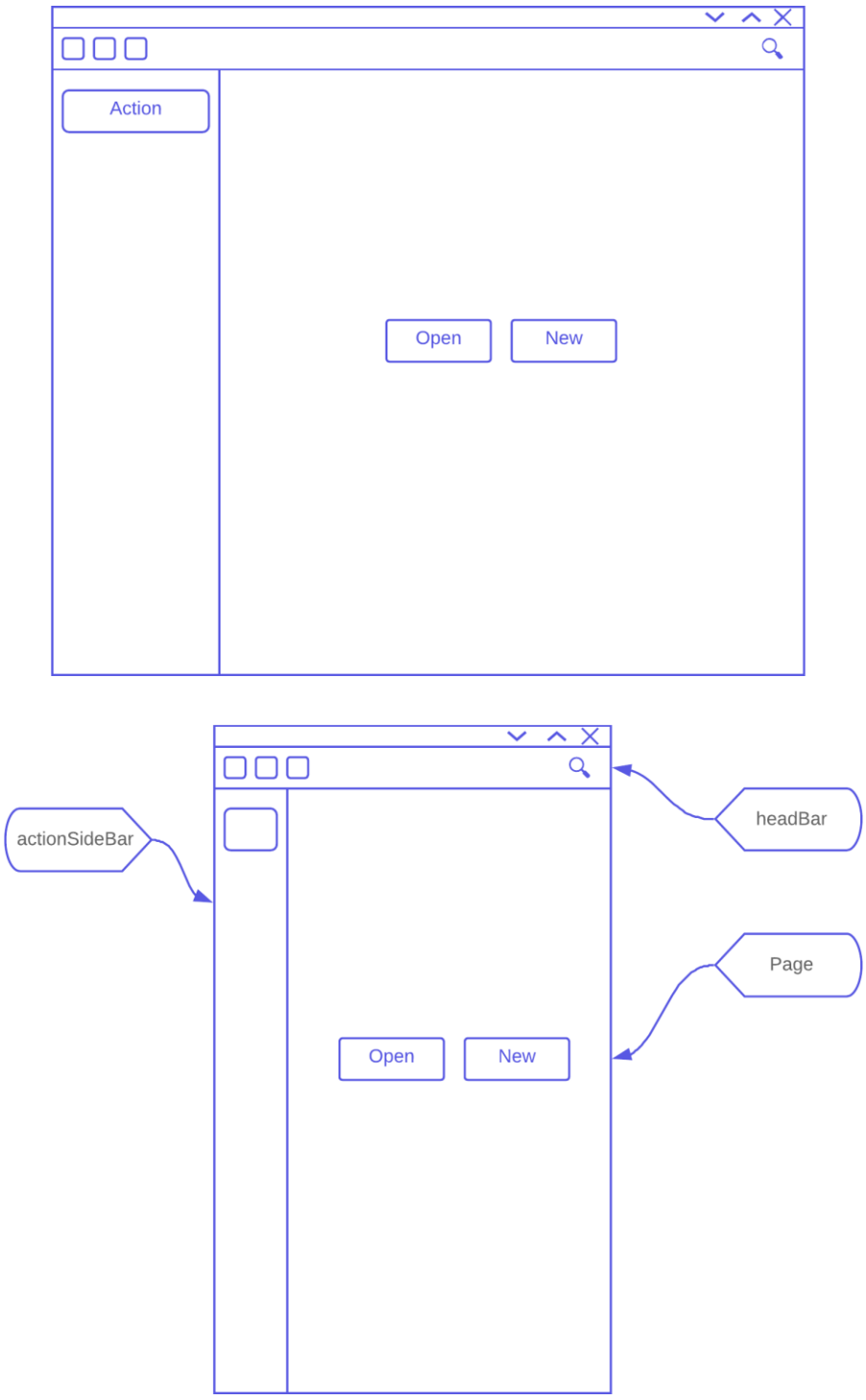

Figura 13. Mockup componentes principales

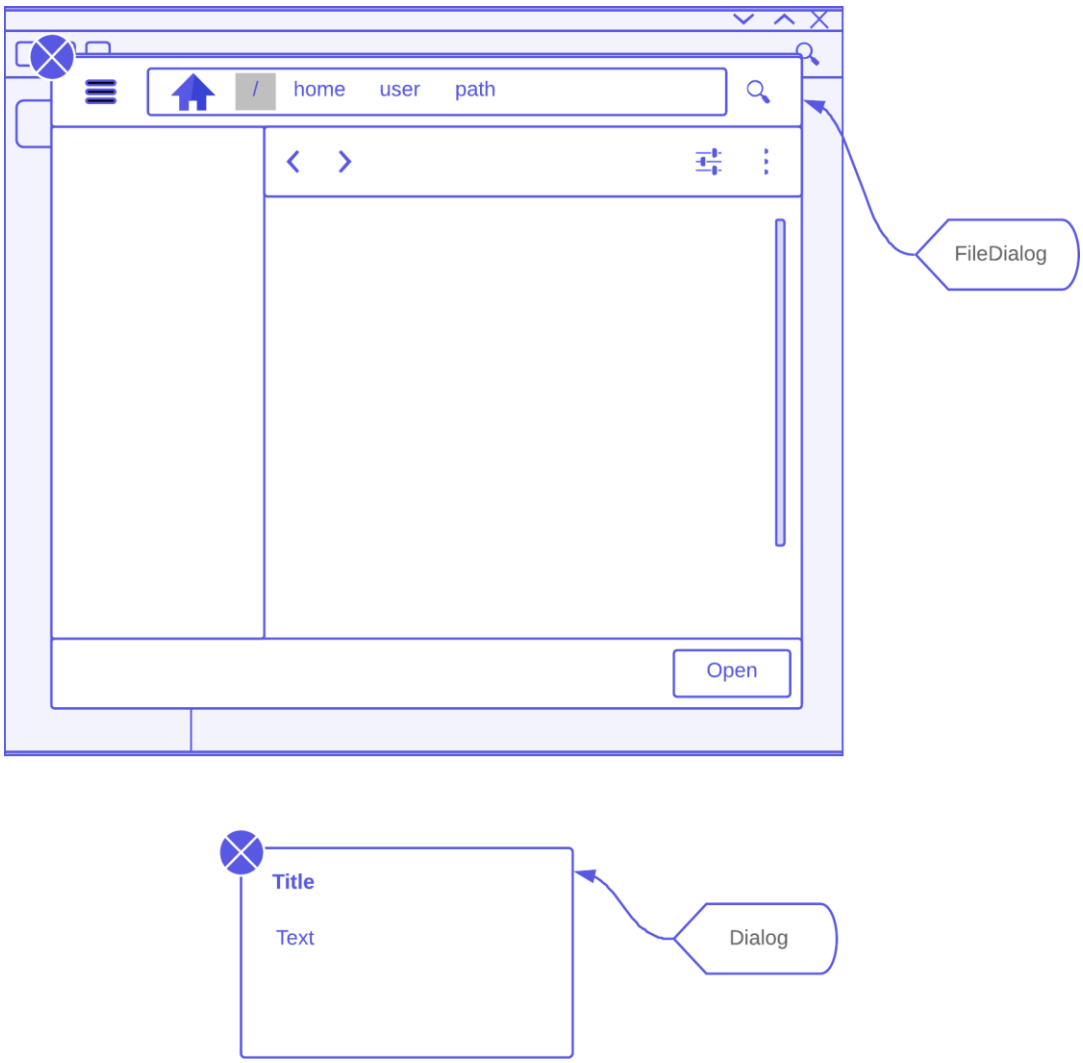

Figura 14. Mockup componentes secundarios

Para el diseño y desarrollo del HIG del framework, se siguieron los delineamientos otorgados por KDE HIG [62]□, en los aspectos HIG del framework en su mayoría fueron construidos a partir de mi colaboración en el proyecto libre Maui Project, presentados ante la comunidad KDE en el KDE Akademy 2019-Milán y que se pueden encontrar en los repositorios de incubación de KDE, en la cual se adaptaron los lineamientos de UI/UX para ser empleados en la perspectiva del framework, además de poder ser usada a partir de un modelo pre establecido.

5.3.1.1. Diagrama de actividades

La interacción de las posibles actividades se encuentran en la Figura 15. Diagrama de actividades, especificando el flujo de trabajo básico implementado para el usuario. En donde la actividad off-page llamada acción, indica el posible proceso que se implemente dentro del framework.

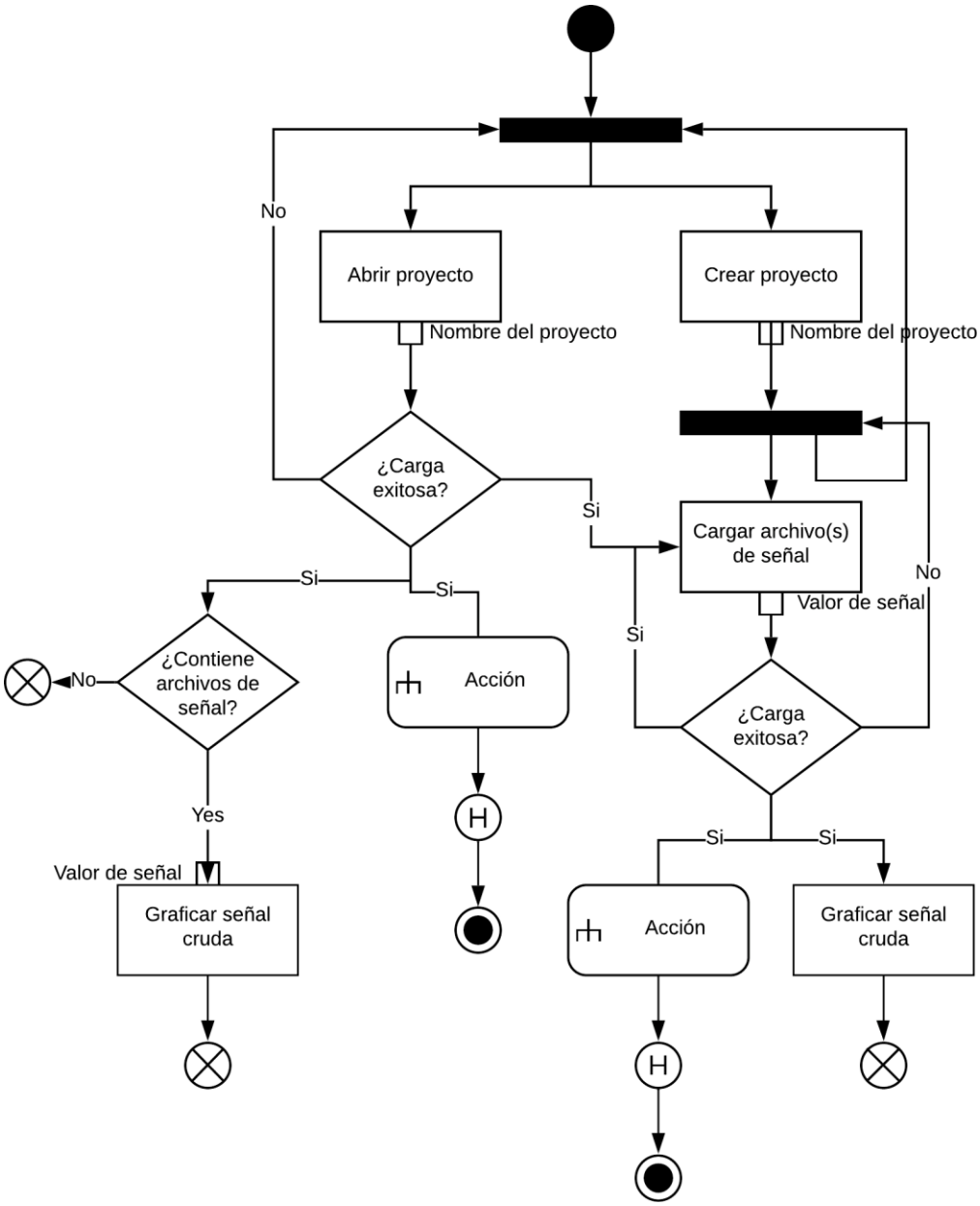

Figura 15. Diagrama de actividades

5.4. Controles y herramientas

5.4.1. Puente de enlace

Las principales librerías que permitieron la creación de todo el contexto del frontend pertenecen a la librería de Qt, especialmente el modulo de QML que permite el desarrollo de las aplicaciones en el lenguaje QML, éste modulo proporciona una API para permitir a los desarrolladores extender el lenguaje QML con tipos personalizados y la integración con JavaScript y C++. Los usos de QML crecen cada vez más, solicitando una integración con otros lenguajes, especialmente los lenguajes de alto nivel, una de las extensiones que se encuentran en estabilización para este fin es con Python. Esta comunicación entre Python y C++ es realizada por el generador de enlace Shiboken2, la cual es el generador de código de enlace basado en CPython para bibliotecas C++, y es la usada para exponer la API Qt C++ a Python mediande PySide2.

Shiboken2 analiza los encabezados Qt y obtiene la información de todas las clases de Qt, además Shiboken2 tiene un sistema de tipos basado en XML que permite modificar la información obtenida para representar y manipular adecuadamente las clases de C++ en Python, procesos que permiten una fácil compilación, este proceso es observable en la Figura 16. Relación entre Qt, Shiboken2 y PySide2[63]□.

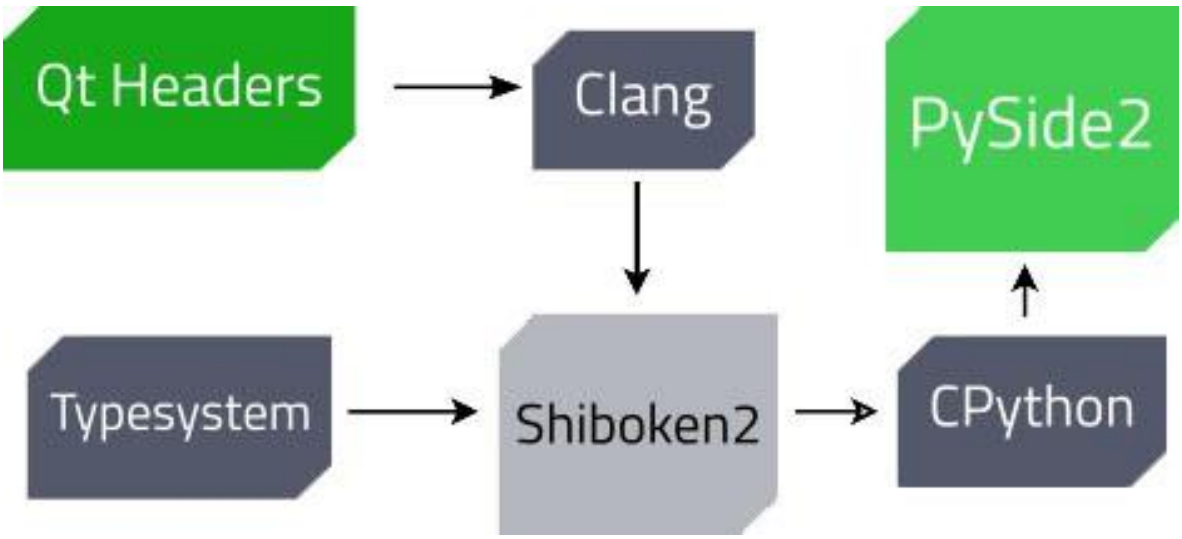

Figura 16. Relación entre Qt, Shiboken2 y PySide2

QML es el lenguaje declarativo de UI del proyecto Qt. A diferencia de los QWidgets, está diseñado teniendo en cuenta los sistemas táctiles (integrados) y, por lo tanto, es ideal para aplicaciones móviles. Kirigami y Maui Kit es un conjunto de componentes QtQuick diseñados para crear aplicaciones convergentes. A diferencia del resto de las aplicaciones construidas con estas herramientas en este proyecto no se empleó C++ para la lógica, en cambio, se empleó Python 3.x, que mediante PySide2 es compatible con Qt. Para el uso de Qt QML se generó un archivo de compilación con Shiboken2 para la correcta comunicación e integración, la cual ya se encuentra indexa dentro de los archivos activos de PySide2 (submodulo de QqmlApplicationEngine() dentro de la librería de PySide2).

Ya teniendo la posibilidad de integración entre librerías escritas en C++ con Qt y Python 3.x, se procedió a la construcción de componentes de la parte del frontend y el backend.

5.4.2. Modelo de componentes

En este numeral se describe la realización de la funcionalidad requerida en el sistema en términos de componentes y que además sirve como una abstracción del código fuente.

La funcionalidad de los diferentes componentes fueron construidas mediante encapsulamiento en objetos. Los controles fueron agrupados en 3 principales clases, conectadas al frontend mediante el motor de comunicación de contexto, Figura 17. Modelo de clase

- **LoadProject**: Realiza la creación y carga de los proyectos, así como la coordinación del archivo de configuración.
- **LoadFile**: Realiza la lectura de los arhivos de señal, así como la gráfica de la señal cruda.
- **Process**: Realiza la creación de procesos, reportes y gráficas para ser incorporadas al sistema.

La empaquetación de estas clases permite la comunicación con el frontend, significando que solo se instancian una vez por compilación.

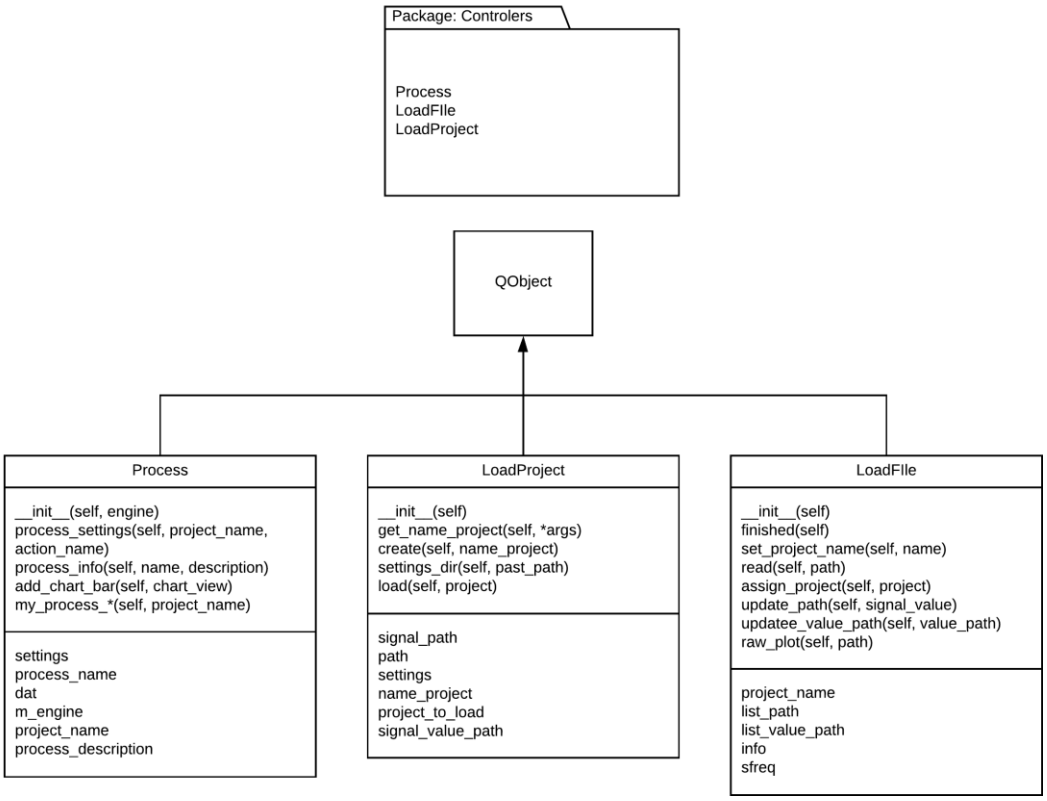

Figura 17. Modelo de clase

En la construcción de los componentes de interfaz se hace uso de la empaquetación del backend, siendo así la capa lógica del modelo de capas en el frontend, Figura 18. Modelo de interfaz, mostrando los métodos disponibles de cada interfaz.

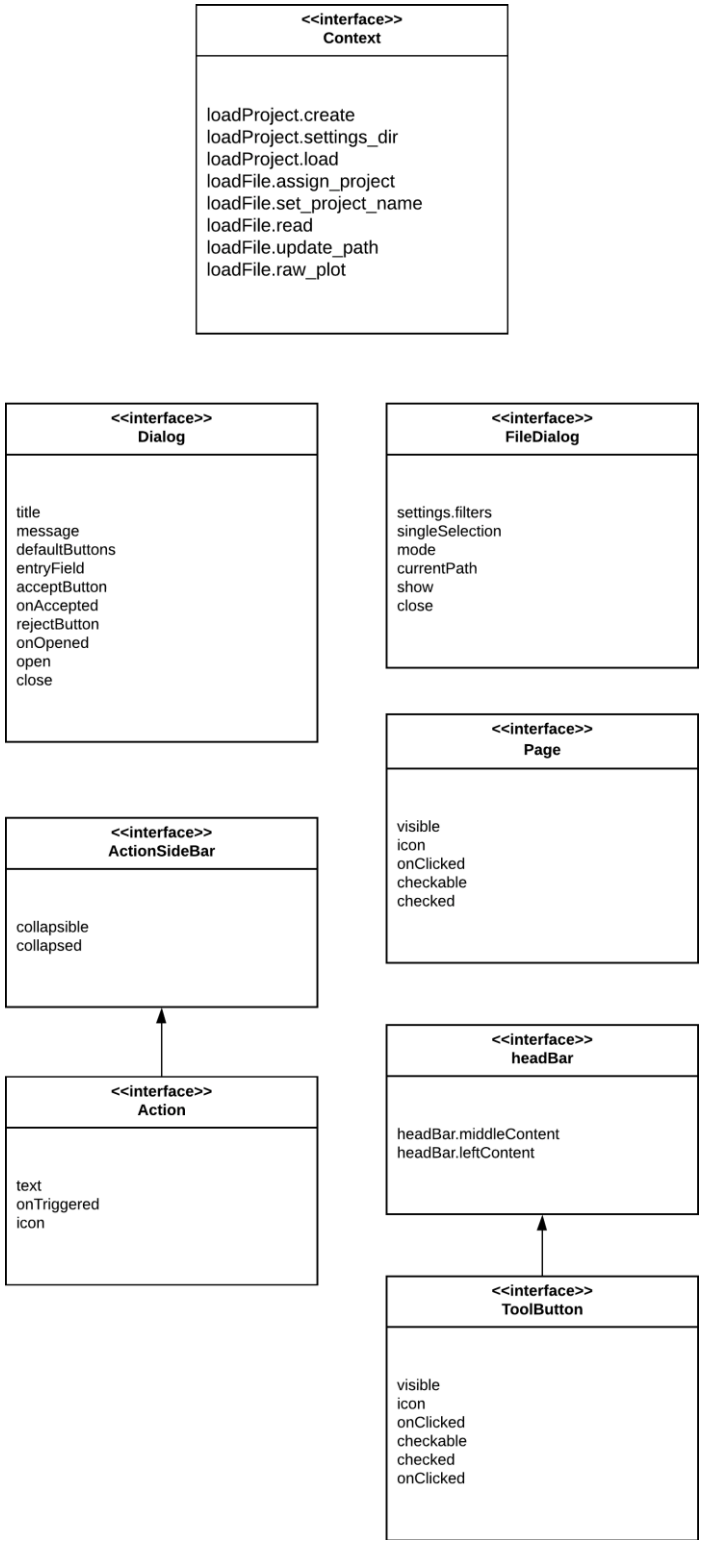

Figura 18. Modelo de interfaz

5.4.3. Herramientas

Los controles definidos anteriormente dan paso a la formación de las herramientas. Dentro del framework se definieron algunas herramientas que realizan ciertos procesos importantes dentro de un flujo de trabajo de usuario final, establecidas dentro de las HU, como:

I. Creación y carga proyectos
II. Generación de archivo de configuración con la información completa del proyecto
III. Decodifcación y carga de una o varia señales EEG en formato *.cnt* y *.set*
IV. Gráfica de la señal completa previamente cargada, en crudo
V. Generación de un informe de la información del proyecto, incluyendo los procesos aplicados, procesos de los cuales se anexan externamente

Estas herramientas son las características que se cuentan por defecto en el framework, por su encapsulación y atomicidad pueden ser modificadas en cuanto cambie la historia de usuario respectiva de cada herramienta.

5.4.4. Integración externa

Para el uso del framework en la adición de acciones, componentes o vistas/pantallas se es necesario adicionar en la sección respectiva el complemento requerido, la estructura actual facilita su integración con bajo acondicionamiento.

Las adiciones se emplean mediante la construcción de métodos/funciones si es parte del backend o componentes si es parte del frontend. Interactuando con métodos ya establecidos para la entrega y envío de información (líneas comentadas), la cual son de relevancia para el funcionamiento.

Un ejemplo de adición de una acción en el *actionSideBar*, como lo sería la extracción de alguna métrica de la señal EEG:

Backend:

/Nebula/src/controlers/process_actions.py

```
class Process(QObject):

        …

        @Slot(str, result = "QVariantList")
        def my_process_1(self, project_name):
                import statistics
                self.process_info("process_name",   "process  description")  #     set the process name and
description
                data = get_data_path(project_name) # get the EEG data values, return a list with all the data
values
                print(data, data[0])
                data = get_data(data[0]) # get only de first data value
                print(data.shape)
                var = statistics.variance(data[0, :])
                mean = statistics.mean(data[0, :])
                std = statistics.stdev(data[0, :])
                value = [std * 1000, float(var) * 1000, float(mean) * 1000]
                self.dat["process_1"] = value # set the result in the dictionary
                print(value)
                return value # send the result to the frontend
```

Frontend:

/Nebula/src/views/ActionsSideBar.qml

```
Maui.ActionSideBar
{
   id: slideView
   preferredWidth: Kirigami.Units.gridUnit * 10
   collapsible: true
   collapsed: !root.isWide

          ...

   Action {
      id: action1
      text: "Button2" // name in the button
      onTriggered:
      {
         var result_process1 = loadProcess.my_process_1(mainPath) // call my function from the backend
         var process1 = loadProcess.process_settings(mainPath, action1.text) // load the settings in the log file, always
have to go
         secondChart = true // true if you want to plot the result, only for an 2D-array values
         swipeView.currentIndex = 2; // Set the view number, different to 1
      }
      icon.name: "anchor" // icon name
   }
```

```
        // Another action button

    Action {
        id: action2
        text: "Button3"
        onTriggered: console.log("button3 slideview")
        icon.name: "anchor"
    }

}
```

Como se observa en los dos extractos de códigos, es sólo adicionar/modificar las líneas comentadas, tomando menos de 10 líneas para generar un acople.

Con esto ya se obtiene un proceso a realizar y como resultado una gráfica a partir de un vector de valores.

6. Conclusiones

Las metodologías de desarrollo de software tanto tradicionales como ágiles presentan sus debilidades, para complementar estas debilidades, otras metodologías pueden usarse juntas para alcanzar el éxito. El modelo híbrido en neuroingeniería propuesto es un paradigma de gestión de proyectos y tiene la capacidad de producir productos de software de calidad alineados con los requisitos y objetivos del cliente, además de poder ser empleado en la academia e investigación, ya que su naturaleza de creación es el enfoque evolutivo de investigación. Los resultados de su aplicación demostraron que es posible aumentar la productividad y retroespectiva en cada iteración en los estudiantes probados. Un trabajo futuro es probar el modelo propuesto utilizando entornos reactivos con más estudiantes y observar el proceso en su aplicación en otros trabajos investigativos de pocos integrantes.

Al igual que otras aplicaciones multiplataforma y convergente de la comunidad Linux / KDE, se empleó tecnologías declarativas de UI, que a diferencia de los Widgets, se permitió el diseño teniendo en cuenta los sistemas táctiles y la sensibilidad de los tamaños, ideal para aplicaciones de escritorio / móviles (basadas en linux, como Plasma mobile).

Un modelo frontend establecido permitió la simplificación y el avance de los procesos de desarrollo, proporcionando flexibilidad para otros logros, en donde el cliente recibe excelentes UI y UX. En el presente framework se encuentra la construcción de toda la batería del frontend teniendo en cuenta el flujo de trabajo con información neurofisiológica, otorgando estandarización en las aplicaciones construidas, permitiendo el desarrollo continuo colaborativo y reduciendo el tiempo en la estructuración de aplicaciones finales de usuario. En cuanto al lado de la colaboración por parte del desarrollo la curva de aprendizaje no es extensiva y permite un rápido acople de mantenimiento y pruebas.

Los controles y herramientas básicas para un flujo estándar de trabajo fueron incorporados, dando acceso al rápido desarrollo continuo y de forma limpia, logrando construir aplicaciones sin necesidad de reinventar la rueda, permitiendo el enfoque en las acciones extra como un core. La arquitectura implementada es una base estable e importante para el crecimiento del framework y el escalamiento con futuras colaboraciones. Se espera que su colaboración permita que el framework pueda ser pragmático, mejorando la estabilidad en las herramientas existentes y adición de nuevas.

<Este trabajo fué desarrollado y escrito completamente con herramientas FOSS>

7. Bibliografía

[1] F. M. Unturbe, E. P. de Pablo, and F. del P. Guerrero, *Conectividad funcional y anatómica en el cerebro humano: Análisis de señales y aplicaciones en ciencias de la salud*. 2015.

[2] A. Gevins, "Electrophysiological Imaging of Brain Function," *Brain Mapp. Methods*, pp. 175–188, Jan. 2002.

[3] "The role of biomarkers and imaging in the clinical diagnosis of dementia.," *Age Ageing*.

[4] R. S. Khandpur, "Biomedical Recorders," in *HANDBOOK OF BIOMEDICAL INSTRUMENTATION*, Third., Acces engineering, Ed. McGraw-Hill Professional, 2014.

[5] F. S. F. Inc, "¿Qué es el software libre?," *Free Software Foundation, Inc*, 2012. [Online]. Available: https://www.gnu.org/philosophy/free-sw.html. [Accessed: 14-Oct-2018].

[6] A. Delorme and S. Makeig, "EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis," *J. Neurosci. Methods*, vol. 134, no. 1, pp. 9–21, Mar. 2004.

[7] F. Tadel, S. Baillet, J. C. Mosher, D. Pantazis, and R. M. Leahy, "Brainstorm: a user-friendly application for MEG/EEG analysis.," *Comput. Intell. Neurosci.*, vol. 2011, p. 879716, Apr. 2011.

[8] MITSAR, "WinEEG:: QEEG and ERP Analysis Software," 2018. [Online]. Available: http://www.mitsar-medical.com/eeg-software/qeeg-software/. [Accessed: 14-Oct-2018].

[9] F. S. Bao, X. Liu, and C. Zhang, "PyEEG: an open source Python module for EEG/MEG feature extraction.," *Comput. Intell. Neurosci.*, vol. 2011, p. 406391, Mar. 2011.

[10] BrainProducts, "Brain Products GmbH / Products & Applications / Analyzer 2," 1997. [Online]. Available: https://www.brainproducts.com/productdetails.php?id=17. [Accessed: 14-Oct-2018].

[11] M. Scherg, P. Berg, and BESA, "BESA® | Brain Electrical Source Analysis," 2018. [Online]. Available: http://www.besa.de/products/besa-research/besa-research-overview/. [Accessed: 14-Oct-2018].

[12] brainwaves.io, "Brainwave Analysis," 2018. [Online]. Available: http://brainwaves.io/wp/. [Accessed: 14-Oct-2018].

[13] MARTINOS and MNE DEVELOPERS, "MNE — MNE 0.16.1," 2018. [Online]. Available: https://martinos.org/mne/stable/index.html. [Accessed: 14-Oct-2018].

[14] GNU and UNESCO, "Free and Open Source Software (FOSS)," 2017. [Online]. Available: https://en.unesco.org/foss. [Accessed: 14-Oct-2018].

[15] GNU, “Free Open Source Software -FOSS,” 2016. [Online]. Available: http://freeopensourcesoftware.org/index.php/Main_Page. [Accessed: 14-Oct-2018].

[16] ISO 2500, “ISO 25010, Calidad del producto software.” [Online]. Available: https://iso25000.com/index.php/normas-iso-25000/iso-25010. [Accessed: 14-Oct-2018].

[17] O. M. Group, “OMG Unified Modeling Language TM (OMG UML), Superstructure v.2.3,” *InformatikSpektrum*, 2010.

[18] R. S. Pressman, *Software Engineering A Practitioner’s Approach 7th Ed - Roger S. Pressman*. 2009.

[19] L. R. Vijayasarathy and C. W. Butler, “Choice of Software Development Methodologies: Do Organizational, Project, and Team Characteristics Matter?,” *IEEE Softw.*, vol. 33, no. 5, pp. 86–94, Sep. 2016.

[20] L. Lindstrom and R. Jeffries, “Extreme Programming and Agile Software Development Methodologies,” *Inf. Syst. Manag.*, vol. 21, no. 3, pp. 41–52, Jun. 2004.

[21] K. Beck *et al.*, “Manifesto for Agile Software Development,” *The Agile Alliance*, 2001. .

[22] pmi, “Success Rates Rise 2017 9th Global Project Management Survey,” USA, 2017.

[23] “Agile Project Delivery Confidence,” 2017.

[24] T. Friend, “Agile Project Success and Failure (The Story of the FBI Sentinel Program),” in *Software Solutions Symposium*, 2017.

[25] D. Murugaiyan, “International Journal of Information Technology and Business Management WATEERFALLVs V-MODEL Vs AGILE: A COMPARATIVE STUDY ON SDLC,” vol. 2, no. 1, 2012.

[26] M. A. Awad, “A Comparison between Agile and Traditional Software Development Methodologies.”

[27] M. Flower, “The New Methodology,” 2005. [Online]. Available: https://www.martinfowler.com/articles/newMethodology.html. [Accessed: 18-Jun-2019].

[28] J. Erickson, K. Lyytinen, and K. Siau, “Agile Modeling, Agile Software Development, and Extreme Programming,” *J. Database Manag.*, vol. 16, no. 4, pp. 88–100, Oct. 2005.

[29] K. L. Beck, “Extreme programming explained - embrace change,” *undefined*, 1990.

[30] S. Wood, G. Michaelides, and C. Thomson, “Successful extreme programming: Fidelity to the methodology or good teamworking?,” *Inf. Softw. Technol.*, vol. 55, no. 4, pp. 660–672, Apr. 2013.

[31] L. Williams, L. Williams, W. Krebs, and L. Layman, “Extreme Programming Evaluation Framework for Object-Oriented Languages -- Version 1.4,” 2004.

[32] K. Beck, “Ver el código de «Programacion extrema» - EcuRed,” *EcuRed*, 1996. [Online]. Available: https://www.ecured.cu/index.php?title=Programacion_extrema&action=edit. [Accessed: 21-Jun-2019].

[33] SCRUMstudy.com, *A Guide to the Scrum Body Of Knowledge (SBOKTMGuide) – 3rd Edition*, 3rd ed. Arizona: SCRUMstudy, 2017.

[34] K. Schwaber, “Improving the Profession of Software Delivery,” *Scrum.org*, 2005. [Online]. Available: https://www.scrum.org/index.php/about. [Accessed: 23-Jun-2019].

[35] M. Cohn, *User Stories Applied From the Library of www.wowebook.com*. Boston: Addison-Wesley, 2009.

[36] R. Davies, “The Power of Stories.”

[37] G. Lucassen, F. Dalpiaz, J. M. E. M. Van Der Werf, and S. Brinkkemper, “Forging high-quality User Stories: Towards a discipline for Agile Requirements,” in *2015 IEEE 23rd International Requirements Engineering Conference, RE 2015 - Proceedings*, 2015, pp. 126–135.

[38] A. M. Moreno and A. Yagüe, “Agile user stories enriched with usability,” in *Lecture Notes in Business Information Processing*, 2012, vol. 111 LNBIP, pp. 168–176.

[39] M. Shamsur Rahim, A. Ehtesham Chowdhury, D. Nandi, M. Rahman, and S. Hakim, “ScrumFall: A Hybrid Software Process Model Software Requirements Prioritization in Agile: A Proposed Generic Approach View project articles View project ScrumFall: A Hybrid Software Process Model,” *Artic. Int. J. Inf. Technol. Comput. Sci.*, vol. 12, pp. 41–48, 2018.

[40] B. Mitchell, “U.S. CIO: Agile IT remains ‘immature’ in federal government,” New York, 2015.

[41] Open source community, B. Perens, and E. S. Raymond, “Open Source Initiative,” 2018. [Online]. Available: https://opensource.org/about. [Accessed: 14-Oct-2018].

[42] L. Teixeira, A. R. Xambre, H. Alvelos, N. Filipe, and A. L. Ramos, “Selecting an Open-source Framework: A Practical Case Based on Software Development for Sensory Analysis,” *Procedia Comput. Sci.*, vol. 64, pp. 1057–1064, Jan. 2015.

[43] D. M. Nichols and M. B. Twidale, “Usability processes in open source projects,” *Softw. Process Improv. Pract.*, vol. 11, no. 2, pp. 149–162, Mar. 2006.

[44] L. Bauer, C. Bravo-Lillo, L. Cranor, and E. Fragkaki, “Warning Design Guidelines,” 2013.

[45] C. Benson, M. Müller-Prove, and J. Mzourek, “Professional usability in open source projects: GNOME, OpenOffice.org, NetBeans,” in *Conference on Human Factors in Computing Systems - Proceedings*, 2004, pp. 1083–1084.

[46] N. Viorres, P. Xenofon, M. Stavrakis, E. Vlachogiannis, P. Koutsabasis, and J. Darzentas, “Major HCI challenges for open source software adoption and development,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2007, vol. 4564 LNCS, pp. 455–464.

[47] K. Community, “KDE Human Interface Guidelines — Human Interface Guidelines documentation.” [Online]. Available: https://hig.kde.org/. [Accessed: 24-Oct-2019].

[48] L. Jan, “The fundamental difference between User Experience and User Interface design,” *Medium*, 2019. [Online]. Available: https://uxdesign.cc/the-fundamental-difference-between-user-experience-and-user-interface-design-d634373e6413. [Accessed: 24-Oct-2019].

[49] A. Pedroni, A. Bahreini, and N. Langer, “Automagic: Standardized preprocessing of big EEG data,” *Neuroimage*.

[50] K. J. Gorgolewski *et al.*, “BIDS apps: Improving ease of use, accessibility, and reproducibility of neuroimaging data analysis methods,” *PLoS Comput. Biol.*, vol. 13, no. 3, Mar. 2017.

[51] “Brain Imaging Data Structure.” [Online]. Available: https://bids.neuroimaging.io/. [Accessed: 17-Oct-2019].

[52] A. Spector, P. Norvig, and S. Petrov, “Google’s Hybrid Approach to Research.”

[53] M. Kuhrmann *et al.*, “Hybrid Software Development Approaches in Practice: A European Perspective,” *IEEE Softw.*, vol. 36, no. 4, pp. 20–31, Jul. 2019.

[54] A. Fryrear, “2nd Annual State of Agile Marketing Report - AgileSherpas,” 2019.

[55] A. Smoczyńska, M. Pawlak, and A. Poniszewska-Marańda, “Hybrid Agile Method for Management of Software Creation,” Springer, Cham, 2019, pp. 101–115.

[56] E. S. Hidalgo, “Adapting the scrum framework for agile project management in science: case study of a distributed research initiative.,” *Heliyon*, vol. 5, no. 3, p. e01447, Mar. 2019.

[57] A. Ehtesham Chowdhury, D. Nandi, M. Rahman, M. Shamsur Rahim, and S. Hakim, “ScrumFall: A Hybrid Software Process Model,” *Artic. Int. J. Inf. Technol. Comput. Sci.*, vol. 12, pp. 41–48, 2018.

[58] S. Michael, Stal; Peter, “2.4 Interactive Systems - Pattern-Oriented Software Architecture, Volume 1, A System of Patterns [Book],” in *Pattern-Oriented Software Architecture*, John Wiley & Sons, Inc.

[59] R. Mark, “1. Layered Architecture - Software Architecture Patterns [Book],” in *Software Architecture Patterns*, O’Reilly Media, Inc.

[60] “A Multilayered Architecture for Qt Quick | ICS.” [Online]. Available: https://www.ics.com/blog/multilayered-architecture-qt-quick. [Accessed: 23-Oct-2019].

[61] M. F. MONTENEGRO CARVALHO DA FONTOURA, “A SYSTEMATIC APPROACH TO FRAMEWORK DEVELOPMENT,” Pontifical Catholic University of Rio de Janeiro, 1999.

[62] “About — Human Interface Guidelines documentation.” [Online]. Available: https://hig.kde.org/resources/about.html. [Accessed: 24-Oct-2019].

[63] “Qt for Python/Shiboken - Qt Wiki.” [Online]. Available: https://wiki.qt.io/Qt_for_Python/Shiboken. [Accessed: 12-Nov-2019].

ANEXOS

8. Anexo 1 – licenses_compatibility

Nebula framework is expected to be licensed under the LGPL v3 license, for which the compatibility characteristics among the tools to be used for the construction of Nebula were taken into account.

The following table compares various features of each license and is a general guide to the terms and conditions of each license, in addition to the compatibility of the licenses of the technologies with the Nebula framework. The table lists the permissions and limitations regarding the following subjects:

1. **Linking**: Linking of the licensed code with code licensed under a different license (e.g. when the code is provided as a library).
2. **Distribution** : Distribution of the code to third parties.
3. **Modification**: Modification of the code by a licensee.
4. **Patent grant**: Protection of licensees from patent claims made by code contributors: Regarding their contribution, and protection of contributors from patent claims made by licensees.
5. **Private use**: Whether modification to the code must be shared with the community or may be used privately (e.g. internal use by a corporation).
6. **Sublicensing**: Whether modified code may be licensed under a different license (for example a copyright) or must retain the same license under which it was provided.
7. **Trademark grant**: Use of trademarks associated with the licensed code or its contributors by a licensee.
8. **Compatibility with Nebula framework**: Its use is possible for the creation of the framework.

Table 1. List of compatibility

Technologies name	License name	Linking	Distribution	Modification	Patent grant	Private use	Sublicensing	™ Grant	Compatibility with Nebula Framework
Python	PSFL	Permissive	?	Permissive	?	?	?	?	Yes
Pyside2	LGPL v2.1	With restrictions	Copylefted	Copylefted	Yes	Yes	Copylefted	Yes	Yes
QML	LGPL v3 *	With restrictions	Copylefted	Copylefted	Yes	Yes	Copylefted	Yes	Yes
MNE Python	BSD - 3 clause	Permissive	Permissive	Permissive	Manually	Yes	Permissive	Manually	Yes
Kirigami	LGPL v3	With restrictions	Copylefted	Copylefted	Yes	Yes	Copylefted	Yes	Yes
MAUIKit	LGPL v3	With restrictions	Copylefted	Copylefted	Yes	Yes	Copylefted	Yes	Yes

The following “license slide” figure makes it easy to see when common licenses can be combined:

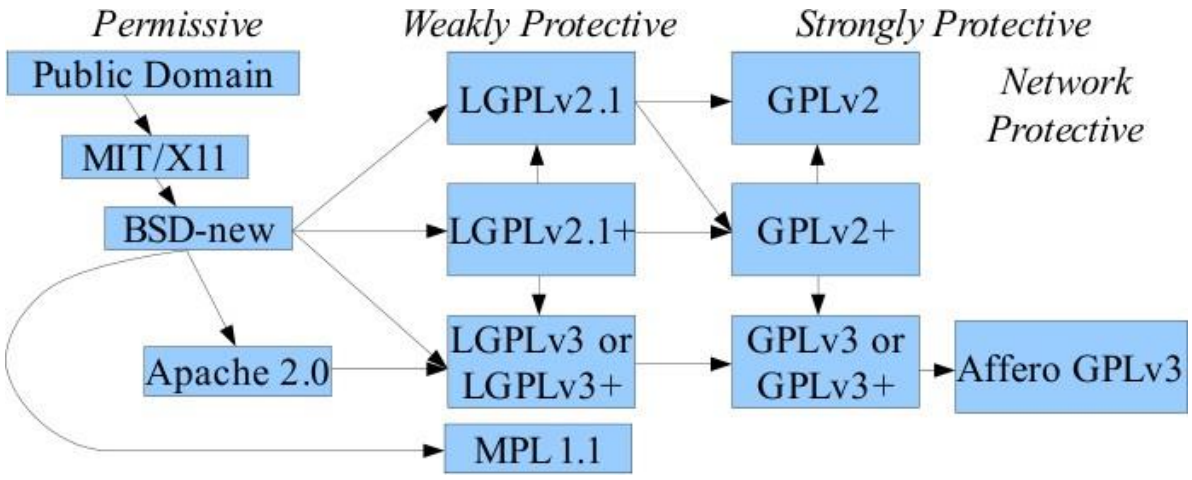

Figura 1. Combination of licenses.

Quick summary

LGPL V2.1: This license mainly applies to libraries. You may copy, distribute and modify the software provided that you state modifications and license them under LGPL-2.1. Anything statically linked to the library can only be redistributed under LGPL, but applications that use the library doesn't have to be. You must allow reverse engineering of your application as necessary to debug and relink the library.

LGPL V3: This license is mainly applied to libraries. You may copy, distribute and modify the software provided that modifications are described and licensed for free under LGPL. Derivatives works (including modifications or anything statically linked to the library) can only be redistributed under LGPL, but applications that use the library don't have to be.

BSD 3-clause: The BSD 3-clause license allows you almost unlimited freedom with the software so long as you include the BSD copyright and license notice in it.

PSFL: The license under which Python 2.0 and Python 3.* was distributed; includes the licenses from previous versions of python.

All of them can be used for:

- Commercial use
- Modify
- Distribute

With QML there is a variation, it is LGPL v3 if everything remains in LGPL v3, if you intend to use copyright for the whole project then it would change to commercial license provided by Qt Company.

But Nebula framework will become private software (with copyright)?, I hope not.

Can we then commercialize our developments?

Yes, of course you can. I will briefly explain how can be under commercial licenses would develop.

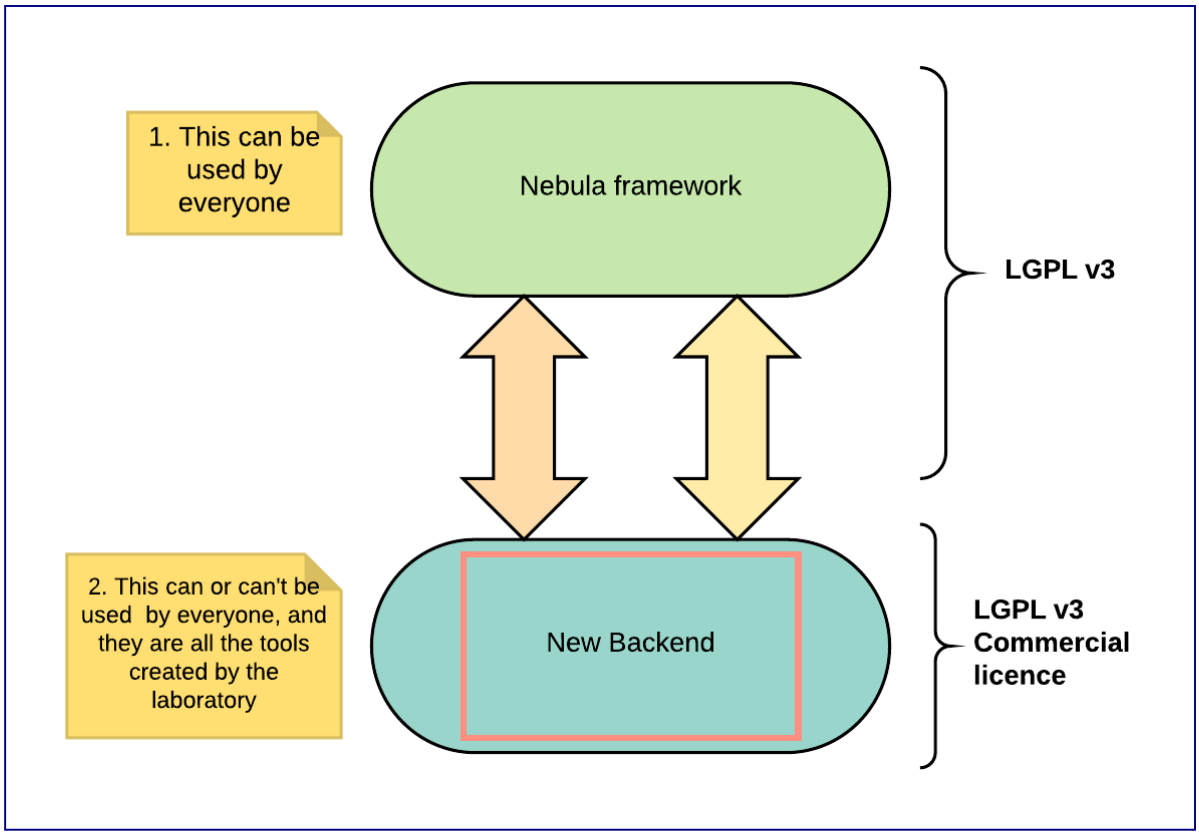

Figure 2. Illustration of the components in Nebula framework and their possibles licenses.

This means that all developments created by the laboratory may or may not have a commercial license, that will depend on the intentions and rewards expected.

On the other hand, the Nebula framework can be modified, distributed and used by anyone, but it can not have a commercial license due to the limitations of using some wheels like pyside2 and QML, unless it is wanted to pay for the commercial license requested by Qt Company.

Then, **What can we sell?**

Everything inside the red box. Then whoever buys that what is being bought is what is in the red box, and everything comes together. It's like saying "*Me vino premiado*".

9. Anexo 2 - Style_Guideline_Nebula

Visitar:

https://gitlab.com/gruponeuropsicologia/style-guideline-nebula