Metodología de desarrollo de software para plataformas educativas robóticas usando ROS-XP

Diego León Ramírez-Bedoya[1], **John Willian Branch-Bedoya**[2], **Jovani Alberto Jiménez-Builes**[3]

[1] Ing. de sistemas, Grupo de Investigación MaterMob, Centro Tecnológico del Mobiliario, Servicio Nacional de Aprendizaje SENA, diegoleonr@misena.edu.co
[2] Ph. D. en ingeniería – sistemas, Profesor Titular, Grupo de Investigación y Desarrollo en Inteligencia Artificial GIDIA, Departamento de Ciencias de la Computación y de la Decisión, Facultad de Minas, Universidad Nacional de Colombia, jwbranch@unal.edu.co
[3] Ph. D. en ingeniería – sistemas, Profesor Titular, Grupo de Investigación y Desarrollo en Inteligencia Artificial GIDIA, Departamento de Ciencias de la Computación y de la Decisión, Facultad de Minas, Universidad Nacional de Colombia, jajimen1@unal.edu.co

RESUMEN

Este artículo presenta una metodología basada en el proceso ágil de XP para el desarrollo de software orientado a robots en plataformas educativas usando *middleware* ROS. El resultado final fue un conjunto de evidencias tales como historias de usuarios, diagramas basados en UML y líneas de código en el lenguaje Python que demuestran buenas prácticas de calidad de código en el desarrollo de software con el uso del paradigma orientado a objetos. Se obtiene una metodología ágil modificada con un diseño evolutivo e incremental, pero con una documentación rigurosa útil para proyectos educativos y de investigación.

Palabras clave: ROS, programación extrema XP, internet de las cosas, robótica, inteligencia artificial.

Recibido: 10 de Agosto de 2019. Aceptado: 27 de Noviembre de 2019
Received: August 10, 2019. Accepted: November 27, 2019

Methodology of software development for robotic educational platforms using ROS-XP

ABSTRACT

This article presents a methodology based on the agile XP process for the development of robot-oriented software in educational platforms using the ROS middleware. The final result was a set of evidences such as user stories, UML-based diagrams and lines of code in the Python language that demonstrate good code quality practices in software development with the use of the object-oriented paradigm. A modified agile methodology is obtained with an evolutionary and incremental design but with a rigorous documentation useful for educational and research projects.

Keywords: *ROS, XP, Internet of Things, Robotics, Artificial Intelligence***.**

Cómo citar este artículo: D. Ramírez, J. Branch, J. Jiménez. "Metodología de desarrollo de software para plataformas educativas robóticas usando ROS-XP.", Revista Politécnica, vol. 15, no.30, pp.55-69, 2019. DOI: 10.33571/rpolitec.v15n30a6

1. INTRODUCCIÓN

Debido a que los avances tecnológicos para la elaboración de software para robots cambian continuamente, se hace necesario rediseñar las estrategias e incluso adquirir nuevas habilidades o conocimientos que permitan asumir los nuevos retos. Por ejemplo, se evidenciaba el uso intensivo de compuertas digitales, las cuales se encontraban incorporadas en circuitos integrados. Esta tecnología hacía uso de la lógica digital para la implementación los algoritmos que permite interpretar los datos de los sensores y dirigir los diferentes actuadores en las plataformas robóticas. Estos desarrollos, aunque cumplen su objetivo de manera eficiente, ocupan un espacio físico considerable [1]. Es por ello que se empieza a experimentar el sistema de control de los prototipos con procesadores, con la particularidad de que el desarrollador debe incorporar el código en una memoria externa. El ingreso de instrucciones a estos dispositivos es complejo, debido a que está basado en códigos hexadecimales, que representan una instrucción dentro de la tabla de comandos y, además, para una correcta implementación es necesario conocimientos en electrónica [1].

En aras de reducir el tamaño de los sistemas de control llega la era de los microcontroladores, los cuales son dispositivos integrados que poseen internamente la arquitectura de un mini computador y, por consiguiente, todos los elementos que antes se encontraban separados (como en el caso de los procesadores) ahora están incorporados en un chip, por lo que se convierten una alternativa viable para el desarrollo de tarjetas electrónicas.

En la actualidad, existe un creciente interés por los computadores monoplaca que -a diferencia de sus antecesores- permiten la instalación de un sistema operativo. Esta característica los hace más versátil para la programación y, por ende, más idóneos para proyectos de investigación orientados a la robótica.

Cabe resaltar que la posibilidad de usar sistemas operativos embebidos en los robots demanda la creación de diferentes librerías y *frameworks* para facilitar la implementación de algoritmos más elaborados en las plataformas robóticas [2]. Cada uno de estos cambios proponen retos en el desarrollo de software y, por tanto, diferentes formas de hacer las cosas. Es de esperarse que se generen cambios en las metodologías que se ajusten a los avances científicos, tecnológicos y técnicos de la robótica.

Existen diferentes *frameworks* que suministran librerías para el desarrollo de software orientados a las plataformas robóticas. Entre ellos el Sistemas Operativo para Robots (ROS por sus siglas en inglés), el cual maneja librerías de lógica difusa, inteligencia artificial, visión por computador entre otros. Es decir, aparece la oportunidad y conveniencia de diferentes paradigmas que facilitan la construcción de diseños sofisticados para implementar soluciones a problemas de movilidad, visión y toma de decisiones entre otros [3].

Así mismo, la disposición a investigar sobre el middleware ROS, prevé que los usuarios requieran una metodología de trabajo enfocada a la validación y experimentación de los algoritmos diseñados. Algunos de estos usuarios poseen poco o ningún conocimiento de ingeniería de software, programación orientada a objetos y aplicaciones distribuidas. Por lo tanto, se debe definir una metodología de trabajo práctico, sencillo y fácil de aprender.

El aporte de este artículo es el diseño de una metodología basada en el proceso ágil XP, que además incorpora en el diseño diagramas propios del UML para el desarrollo de software utilizando ROS en plataformas robóticas con fines educativos o de investigación. Lo anterior permite realizar aplicaciones de manera metódicas en sistemas embebidos.

El artículo se divide en los siguientes capítulos: en el primero, se presenta la introducción luego, se exteriorizan los materiales y métodos empleados. En el capítulo tres se muestra la metodología para luego exhibir, en el capítulo cuatro los resultados y la discusión. Finalmente se enseñan las conclusiones y la bibliografía.

2. MATERIALES Y METODO

2.1 Robot Operating System (ROS)

ROS es un *framework* para robots que permite configurar una amplia gama de plataformas robóticas, que poseen una colección de herramientas, librerías, y convenciones con el objetivo de simplificar la programación. Por estas

características es unos de los más usados a nivel mundial [4]. Incorpora tópicos en diferentes campos como la visión artificial, mapeo de ambientes internos, reconocimiento de objetos, entre otros. ROS está diseñado para ejecutarse como un *middleware* multiprocesos. El hecho de ejecutar la programación de manera paralela, hace que mejore su desempeño, y evita pausas innecesarias. Por ejemplo, un nodo puede obtener información de un sistema de percepción visual a medida que un robot se va desplazando. Lo anterior, impide que se interrumpa la trayectoria mientras se obtienen imágenes de su entorno.

El *middleware* ROS puede funcionar con diferentes lenguajes de programación e implementa varios recursos para permitir que los usuarios desarrollen sus propios nodos. Por ejemplo, tenemos: Rosjava sirve para aplicaciones en Java, Roscpp que son librerías que permiten programar en C++. y, también Rospy que usa todas las ventajas del lenguaje Python.

El funcionamiento de ROS se puede visualizar en el siguiente ejemplo: en la figura 1 se aprecian dos nodos, uno para sensor y otro para el motor. Inicialmente, el nodo sensor notifica al master (servidor de registros y nombres para otros nodos) que quiere publicar imágenes sobre el *topic* "Distancia":

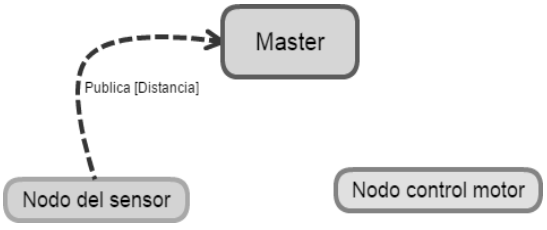

Fig. 1. Publicador informa al Master

Después, el nodo sensor publica los datos en el *topic* distancia, y queda a la espera de los procesos que se quieran inscribir. Luego, control motor solicita suscribirse al *topic* distancia con el fin de recuperar los datos capturados por el sensor figura 2.

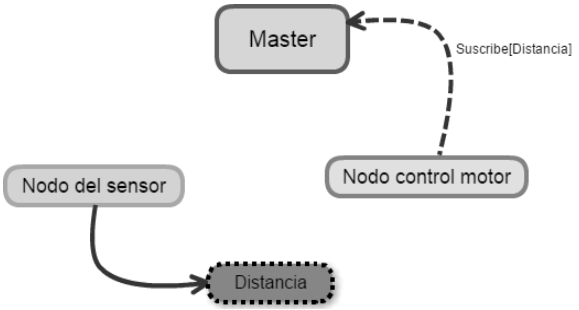

Fig. 2. Suscriptor informa al Master y publicador conecta al *topic*.

Ahora que el tema distancia tiene tanto un publicador como un suscriptor, el nodo maestro notifica al nodo del sensor y control motor sobre la existencia de cada uno de ellos para que puedan comenzar a transferir datos entre sí, como lo ilustra la Fig. 3:

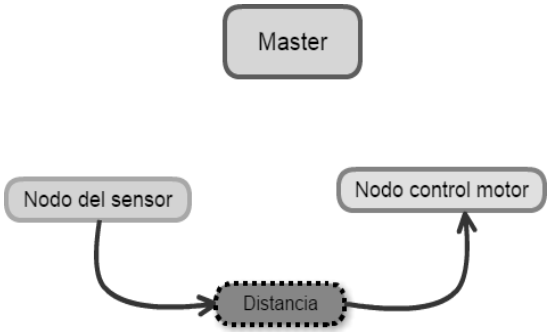

Fig. 3. Publicador y suscriptor conectados por medio de un *topic*.

Con el anterior ejemplo se puede visualizar el comportamiento de los *topic*, el master y los nodos implementados en una plataforma robótica.

2.2 Programación extrema

La programación extrema (XP por las siglas en inglés) es un proceso ágil de desarrollo de software, enfocada a las buenas prácticas de codificación, una clara comunicación y al trabajo en equipo. Está concebida para proyectos medianos y pequeños donde los requisitos son cambiantes. Por lo tanto, tiene una serie de reglas y recomendaciones que se pueden dividir en planeación y gestión, diseño, codificación, y pruebas para producir un software.

En la planeación y gestión se utilizan historias de usuario, en vez de los casos de uso, para definir el

cronograma de entrega de los productos funcionales del software [6].

El diseño debe ser simple para lo cual se usan tarjetas Clase-Responsabilidad-Colaborador (CRC). También se implementan metáforas que permitan explicar la estructura del sistema a los nuevos integrantes del equipo.

La codificación se realiza en parejas, es estandarizada por el equipo de trabajo. Además, se realizan liberaciones frecuentes de versiones. Por último, están las pruebas funcionales, donde se evalúa si la historia de usuario fue implementada correctamente (*Acceptance tests*). También, se realizan las pruebas unitarias que deben ser verificadas para todo el código del proyecto.

2.3 Comparación de metodologías Programación extrema

La Tabla I (obtenida de [8]) hace un cuadro comparativo de diferentes metodologías agiles, como SCRUM, Crystal Methodologies, Dynamic Systems Development Method (DSDM), Feature-Driven Development (FDD) y Lean Development (LD). Por lo tanto, se evalúan parámetros como resultados, adaptabilidad, colaboración, entre otros. Los valores más altos los obtienen XP y Agile Software ´ Development (ASD). Pero XP sobresale de ASD en la excelencia técnica, lo cual es un factor importante para este proyecto y, por lo tanto, un factor diferenciador de XP entre los otros procesos ágiles.

Tabla 1. Comparación de metodologías

	CMM	ASD	Crystal	DSM	FDD	LD	SCRUM	XP
Sistema como algo cambiante	1	5	4	3	3	4	5	5
Colaboración	2	5	5	4	4	4	5	5
Características metodología (CM)								
Resultados	2	5	5	4	4	4	5	5
Simplicidad	1	4	4	3	5	3	5	5
Adaptabilidad	2	5	5	3	3	4	4	3
Excelencia Técnica	4	3	3	4	4	4	3	4
Prácticas de colaboración	2	5	5	4	3	3	4	5
Media CM	2.2	4.4	4.4	3.6	3.8	3.6	4.2	4.4
Media Total	1.7	4.8	4.5	3.6	3.6	3.9	4.7	4.8

2.4 Plataforma robótica educativa Carlitos

Una plataforma robótica educativa es un conjunto de elementos que hacen parte de un sistema autónomo que tiene ciertas habilidades como, por ejemplo, mover objetos, realizar desplazamientos, vigilancia, entre otros. Estas plataformas son aprovechadas en la parte académica para enseñar ciertos principios propios de la robótica o de currículos, especialmente de ingeniería. Para validar la metodología propuesta se utilizó el robot Carlitos (Ver Fig. 4).

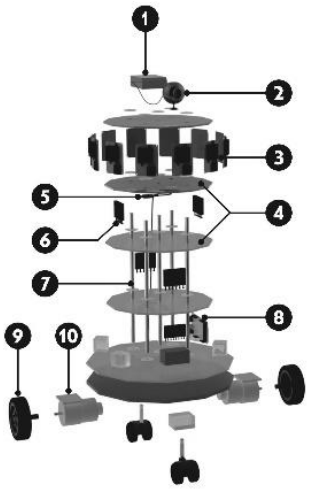

Fig. 4. Componentes de la plataforma robótica educativa Carlitos

La Fig. 4 muestra los componentes del robot Carlitos, a saber: 1) Computador monoplaca. 2) Cámara de video. 3) Sensores ultrasónicos. 4) Bases de la estructura. 5) Cable USB de comunicaciones. 6) Conectores de energía y datos. 7) Soporte cilíndrico para las bases. 8) Arduino Uno. 9) Ruedas de desplazamiento. 10) Motores de corriente continua. 11) Batería de corriente directa. 12) Ruedas para giro.

Los aspectos claves de la plataforma robótica educativa Carlitos pueden enumerarse de la siguiente forma: primero, para el desplazamiento del robot se maneja un control de bajo nivel, el cual se encarga de controlar los motores. Segundo, el uso de un mini computador para correr algoritmos de visión por computador para la navegación autónoma. Tercero, un computador personal para procesar los datos con alto costo computacional y, por último, el middleware ROS que brinda altos niveles de abstracción de hardware y variadas

funcionalidades de uso frecuente en robótica y servicios (Ver Fig. 5).

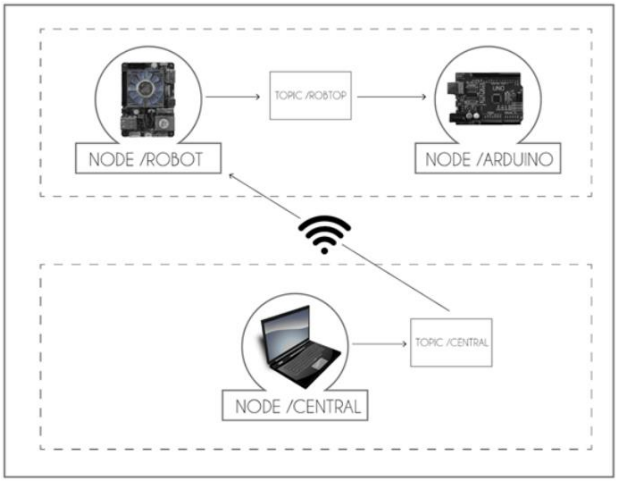

Fig. 5. Ejemplo de diagrama de flujo de la iteración en Carlitos

La tecnología electrónica open source embebida en Carlitos ofrece una serie de módulos de diseño libre, en donde las especificaciones y diagramas esquemáticos son de acceso público. Lo anterior permite disminuir los costos y la elaboración de las tarjetas por los mismos estudiantes. Se usaron principalmente sistemas embebidos de bajo costo para el control de los efectores (motores *DC*) que soportan la navegación [3] del móvil en su espacio de trabajo. Esta parte del proyecto se encarga de interpretar y ejecutar las instrucciones enviadas por el mini computador implementado en la estructura. Aunque el dispositivo puede con una carga operacional más alta, se busca desarrollar una estructura distribuida por nodos, donde cada componente se especialice en una sección en particular.

La arquitectura de control de navegación del robot Carlitos, está compuesta por lo menos de tres nodos principales: captura de imagen con el *Single Board Computer*, control de bajo nivel de los motores por medio del ROS serial y visualización del video en el computador. Estos nodos corren en diferentes unidades de procesamiento, para constituir un sistema distribuido concurrente con transferencia de datos de control y percepción en lo que se conoce como un grafo de computación ROS.

El robot Carlitos es un robot diferencial diseñado y construido teniendo en cuenta criterios de modularidad y escalabilidad del hardware.

3. METODOLOGÍA ROS-XP

El diseño de la metodología se caracterizó por ser experimental, exploratoria e instrumental para la validación de un conjunto de métodos que contemplan la ejecución de una serie de actividades conexas a los objetivos. La metodología se fundamenta en tres pilares, a saber (Ver Figura 6): el primero es una correcta planeación del proyecto para un diseño estructural y de comportamiento basado en las historias de usuario registradas por el docente o investigador. El segundo pilar, está constituido por iteraciones: 1) Una planeación de la iteración. 2) Un diseño basado en algunos diagramas del lenguaje UML (flujo y clases) combinado con diagramas [10] de nodos de ROS y tarjetas CRC modificadas (CRCMAN). 3) El desarrollo de entregas parciales por medio de componentes funcionales en los nodos de desarrollo. 4) Una propuesta de pruebas que incluye la verificación de buenas prácticas para la generación de código del proyecto en la programación orientada a objetos o servicios de los nodos ROS. El tercer pilar, se denomina de producción y se encarga de entregar el diagrama completo del producto desarrollado, junto con el código respectivo, la aplicación con los ajustes necesarios y su puesta en marcha (Ver Fig. 6).

Fig. 6. Pilares de la metodología ROS-XP.

A continuación, se presentan los componentes que poseen cada uno de los tres pilares de la metodología (Ver Fig. 9).

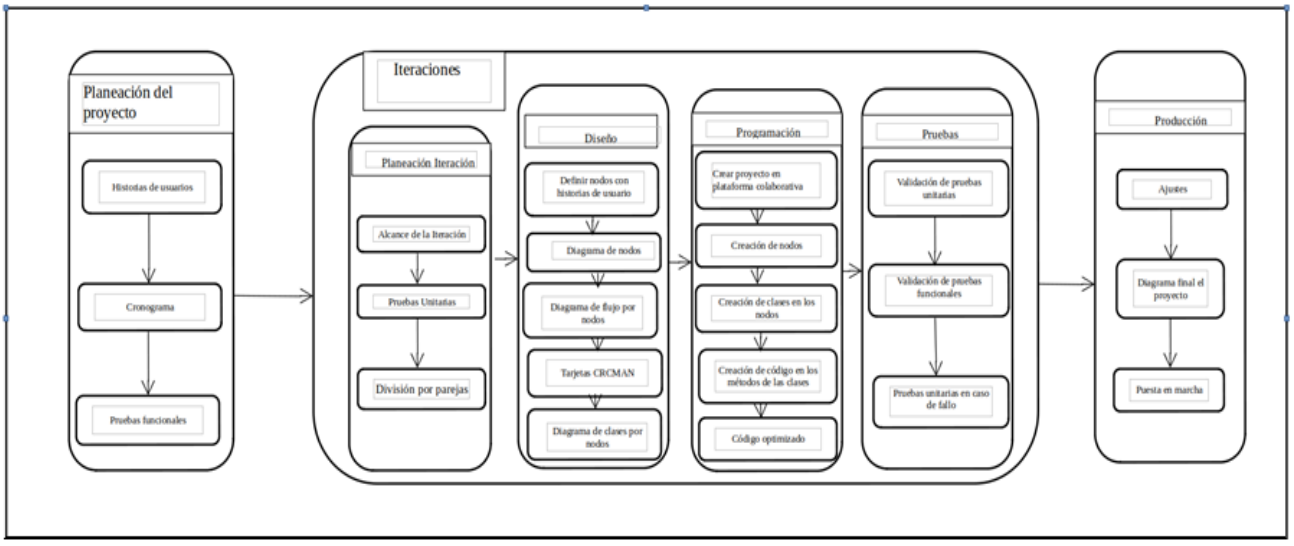

Fig. 7. Fases de la metodología ROS-XP

3.1 Planeación del proyecto

Se compone a su vez de tres etapas, a saber:

3.1.1 Historias de usuario. Son definidas por el líder con la participación de los equipos de desarrollo, las cuales pueden ser modificadas en cualquier momento del ciclo de vida del proyecto, si el equipo de trabajo lo consideraba necesario.

3.1.2 Cronograma. Se diseña el cronograma para definir el número de iteraciones que se realizaron y las fechas de entrega de los módulos funcionales correspondientes a esas fechas.

3.1.3 Pruebas funcionales. Se define las pruebas funcionales. Por ejemplo, determinar el comportamiento que va a tener el robot cuando enfrenta algunos retos.

3.2 Iteraciones

Se compone a su vez, de cuatro etapas, a saber: Planeación de la iteración, Diseño, Programación y Pruebas.

3.2.1 Planeación de la iteración. Como su nombre lo indica, en esta etapa se definen los tres pasos a seguir en un rango determinado de tiempo, a saber:

- Alcance de la iteración. La función principal es dividir el equipo de trabajo en parejas.

- Pruebas unitarias. El equipo de trabajo propone el alcance de la iteración, pero el investigador líder o profesor decide finalmente las prácticas a realizar.

- División del equipo de trabajo. Se definen las pruebas unitarias que se realizan en este proceso de desarrollo, es decir, desde un principio de la iteración, los programadores tienen claro los resultados que se deben evidenciar en cada nodo.

3.2.2 Diseño. La metodología propuesta se diferencia de XP en que se propone un diseño obligatorio pero sencillo, el cual maneja un grupo de diagramas como el de clases, secuencia y computacionales de grafos de ROS. Recordemos que esta metodología fue diseñada para programadores no muy expertos en ingeniería o arquitectura de software; por lo tanto, los diseños deben ser sencillos, pero de gran utilidad.

La metodología se basa en el paradigma de la programación orientada a objetos (POO), por lo tanto, se propone el diagrama de clases para definir la estructura de cada nodo y el de secuencia para contemplar el comportamiento de las clases de dicho nodo. Además, un diagrama computacional de grafos para definir el alcance de cada iteración.

El diagrama de clases está constituido por los objetos que afecten directamente al nodo, en los cuales se tienen en cuenta los atributos del objeto y los métodos.

En el diagrama de secuencia se muestra el comportamiento de las entidades diseñadas en el

diagrama de clases y el número de objetos es limitado por la necesidad del nodo. Es decir, que un nodo debe tener el menor número posible de objetos, pero se debe garantizar que exista al menos uno.

El diseño debe ser evolutivo, lo que quiere decir que se hace para los nodos que se van a entregar en cada iteración, pero estos pueden ser modificados en caso que la práctica lo requiera. La etapa de Diseño se compone de cinco pasos, a saber:

- Definición de los nodos. Se divide la historia de usuario en nodos de entrada, procesamiento y salida (Ver Figura 8).

- Diagrama de nodos. Se realiza el diagrama de nodos de ROS correspondiente a esa iteración.

- Diagrama de flujo por nodos. Por cada nodo se debe realizar un diagrama de flujo que explique el comportamiento de cada nodo relacionado a cada iteración.

- Tarjetas Clase Responsabilidad Colaborador Método Acción Nodo (CRCMAN). Se elabora la tarjeta CRCMAN la cual representa una clase donde se define el nombre de la clase, las responsabilidades, el nombre del nodo donde se encuentra, el nombre de los métodos y con qué clase está relacionada Tablas IV, V, VI.

- Diagrama de clase por nodos. Al final de la iteración se debe entregar el diagrama de secuencias y de clases describiendo lo realizado en la iteración. Para la creación de nodos se debe de tener en cuenta que los supernodos en ROS son aplicaciones que manejan muchos eventos, con sus suscriptores, publicadores y tópicos. Lo que hace que, si alguna parte del proceso falla, se corre el riesgo de que el flujo de todo el sistema se detenga. Además, al reutilizar el código se hace una labor difícil, debido a que la lectura del programa se hace una labor compleja. Por lo tanto, ROS-XP considera que los supernodos no son una buena idea y propone dividir los supernodos para suplir las necesidades de todo el sistema.

ROS-XP propone subdividir las historias de usuarios en partes que diferencien sí son datos de entrada, salida o de procesamiento, para luego crear nodos que se encarguen de esas partes. Por lo tanto, la metodología se adapta perfectamente a la naturaleza evolutiva de la metodología donde la compatibilidad con los cambios es importante en caso de que algo en el proyecto deba ser replanteado.

3.2.3 Programación. La codificación de ROS-XP no cambia de los estándares sugeridos de XP. Esta se basa en el paradigma de la programación orientada a objetos y las buenas prácticas. Una de las propuestas que se sigue es la programación en pareja lo cual permite realizar una codificación de mejor calidad, dando a que se compartan ideas, se aprendan técnicas de programación unos de otros y se corre menos riesgo de perder tiempo.

ROS-XP sugiere que los miembros de la pareja o equipo estén cerca físicamente, pero trabajan en diferentes computadores sobre el mismo código, haciendo uso de una plataforma de desarrollo colaborativo de software para alojar proyectos utilizando el sistema de control de versiones como GitHub.

Hacer uso de plataformas colaborativas con control de versiones facilita el trabajo en parejas porque se pueden hacer pruebas sobre el mismo código y verificar cual es la mejor propuesta. Además, la implementación del proyecto se facilita para cualquier entorno de desarrollo, debido a que se puede descargar un proyecto con sus archivos y estructura.

Antes de dar inicio a las actividades de programación se deben estipular las pruebas unitarias, la cuales deben diseñarse entre el equipo y el líder.

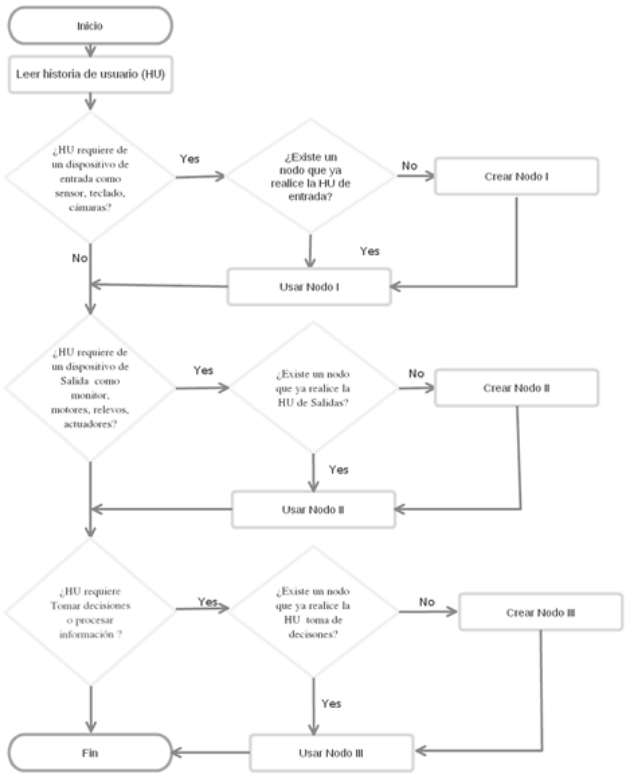

Fig. 8. Propuesta para la creación de nodos en ROS-XP

Los estándares de calidad en el desarrollo de códigos son definidos por el equipo de trabajo. No obstante, se sugiere que debe ir incorporado paradigma orientado a objetos y que la documentación sea una guía para la reutilización de los componentes existentes. La etapa de la Programación comprende a su vez, cinco pasos a saber:

- Creación del proyecto en la plataforma colaborativa. Se implementa un proyecto para la iteración en una plataforma colaborativa.

- Creación de los nodos. Se crea los nodos y las relaciones según el diagrama de nodos diseñado en la iteración.

- Clases con métodos y atributos. Se crea la estructura del código por medio de las tarjetas CRCMAN, diagramas de clase, de nodos y de secuencia.

- Código de los métodos de las clases. Se codifica lo solicitado en la planeación de la iteración.

- Código optimizado. Se optimiza el código como último paso y antes de hacer la entrega.

3.2.4 Pruebas ROS-XP. Aquí se valida el código que se implementó en cada iteración. Esta etapa se divide en tres pasos a saber:

- Validación de pruebas unitarias. Se implementa una unidad de prueba para todo método del código.

- Validación de pruebas funcionales. Se valida el código con las unidades de prueba antes de ser implementado.

- Pruebas unitarias en caso de fallo. Se crea una unidad de prueba Ante un fallo.

3.3 Producción

En este tercer pilar (Ver Fig. 7) se encarga de todo lo referente a la entrega final del proyecto. Se divide en tres etapas a saber:

3.3.1 Ajustes. Se realizan las adecuaciones necesarias del software entregado.

3.3.2 Diagrama final. Se entrega la documentación del proyecto terminado.

3.3.3 Puesta en marcha. Se pone en marcha el producto final.

4. RESULTADOS Y DISCUSIÓN

A continuación, se enumeran y explican las diferentes evidencias que se desarrollan en cada fase de la metodología ROS-XP, aplicadas al caso de estudio de teleoperación de la plataforma robótica llamada Carlitos (Ver Fig. 6).

4.1 Planeación del proyecto

4.1.1 Historias de usuario. En esta parte se recopila los requerimientos que se requieren implementar en la plataforma Carlitos.

- Historia de usuario I - Iteración I. El líder informa a los "*noders*" del requerimiento de un nodo que permita visualizar por medio de un terminal la dirección en que el usuario le ha ordenado avanzar al robot.

- Historia de usuario I - Iteración II: El líder informa a los "*noders*" del requerimiento de una clase que permita crear los nodos y enviar los datos a los *topics* que el usuario le ha ordenado avanzar al robot.

- Historia de usuario III - Iteración III: El líder informa a los "*noders*" que se requiere una clase que permita capturar los datos el teclado y enviarlos a la clase comunicaciones.

4.1.2 Cronograma. Para este caso, indica el rango de fechas en las que trabajará cada historia de usuario (ver Tabla 2)

Tabla 2. Representación cronograma

WBS	Name	Start	End	Time	%
1	El líder informa a los "*noders*" del requerimiento de un nodo que permita visualizar, por medio de una terminal, la dirección en zque el usuario le ha ordenado avanzar al robot.	Jul 11	Jul 28	15d	10
2	El líder informa a los "*noders*" del requerimiento de una clase que permita crear los nodos y enviar los datos a los topics.	Jul 31	Ago 15	15d	5
3	El líder informa a los "*noders*" que se requiere una clase que permita capturar los datos el teclado y enviarlos a la clase comunicaciones.	Sep 1	Sep 15	15d	2
4	Se requiere manejar una señal de video *stream* del robot hacia el computador, teniendo en cuenta que es posible que en un futuro se solicite algún procesamiento con la imagen.	Sep 16	Sep 30	15d	0

4.1.3 Pruebas funcionales. Muestra sí el software se comporta según lo esperado en la planeación, mediante dos pruebas:

- Prueba 1. Se comprueba que al enviar ciertos comandos el sistema reacciona respondiendo con mensajes predeterminados. Por ejemplo, al ingresar el número 8 aparece en la terminal adelante, al ingresar 4 izquierda y al ingresar 6 aparece derecha.

- Prueba 2. Se comprueba el uso de una clase para la construcción de nodos en el sistema que constituye la plataforma Carlitos. Por ejemplo, el cliente debe escoger un nombre del tópico y del nodo, luego ejecutarlo y deben aparecer esos nodos en ejecución con la herramienta Rosgraph.

4.2 Iteraciones

4.2.1 Planeación de la iteración. En esta parte se define el alcance, las pruebas y los equipos de trabajo de cada iteración.

- Alcance de la iteración. Se define qué historia de usuario se realiza en cada iteración:
Historia de usuario I - Iteración I.
Historia de usuario I - Iteración II.
Historia de usuario III - Iteración III.

- Pruebas unitarias. Se diseña las pruebas que permitirán comprobar la idoneidad del software desarrollado. Por ejemplo, para este caso se envía un número al objeto Control y con base a dicho número, el robot debe indicar en la pantalla si va a avanzar, retroceder o girar.

Tabla 3. Test de la clase control

```
class TesttsBasicos(unittest.TestCase):

    def test_control(self):
            control=Controls()
            control.ingresarOrden(8)
            print(self.assertEqual("Arriba",
control.avanceRobot()))

if __name__ == '__main__':
    # begin the unittest.main()
    unittest.main()
```

- División del equipo de trabajo. Se muestra la división de trabajos por parejas de diferentes asuntos que debe de resolver los desarrolladores (Ver Fig. 9)

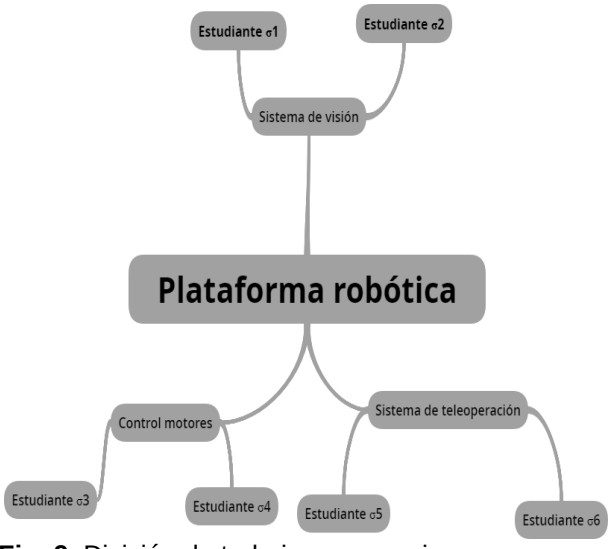

Fig. 9. División de trabajos por parejas

4.2.2 Diseño. Se implementan algunos de los diagramas necesarios para implementar el desarrollo de software de teleoperación.

- Definición de los nodos. Se divide la historia de usuario en nodos de entrada, procesamiento y salida, para luego crear nodos que se encarguen de esas responsabilidades.

- Diagrama de nodos. Se diseñan los nodos específicos de cada iteración para implementar el software de la plataforma robótica educativa (Ver Fig. 10 y Fig. 11).

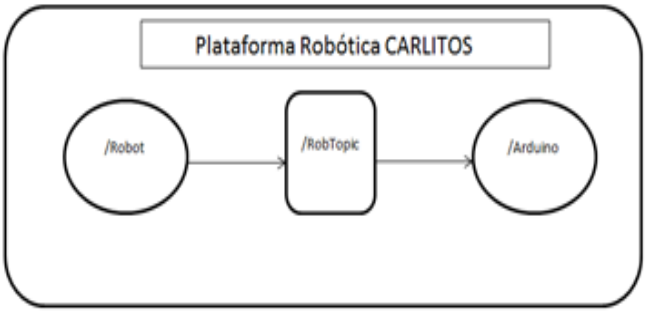

Fig. 10. Grafo computacional de ROS primera iteración

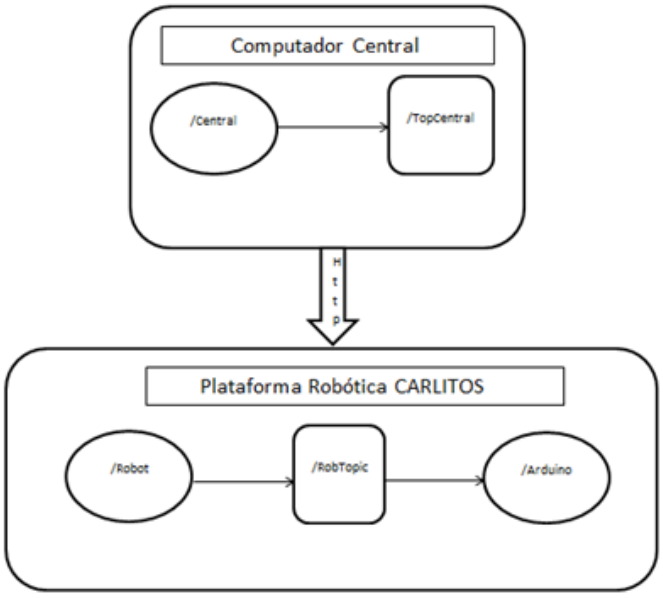

Fig. 11. Grafo computacional de ROS segunda iteración

- Diagrama de flujo por nodos. Representa los pasos a seguir en el algoritmo para implementarlo en cada nodo.

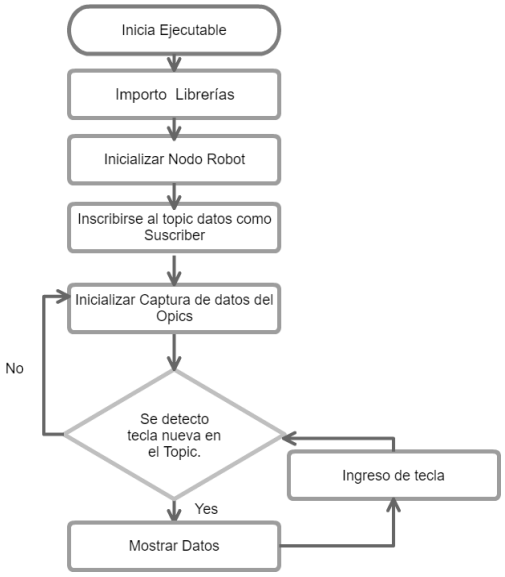

Fig. 12. Captura de datos del teclado y mostrar en pantalla

Se muestra un ejemplo sencillo del diagrama de flujo, en el nodo /Central, en el cual donde se muestran los datos ingresados por teclado para luego ser mostrados en una pantalla (Ver Fig. 12).

Luego, Se visualiza el mismo algoritmo, pero los datos son enviados a un nodo diferente para ser mostrados en la pantalla del nodo / robot (Ver Fig. 13), posteriormente este será el principio de la teleoperación.

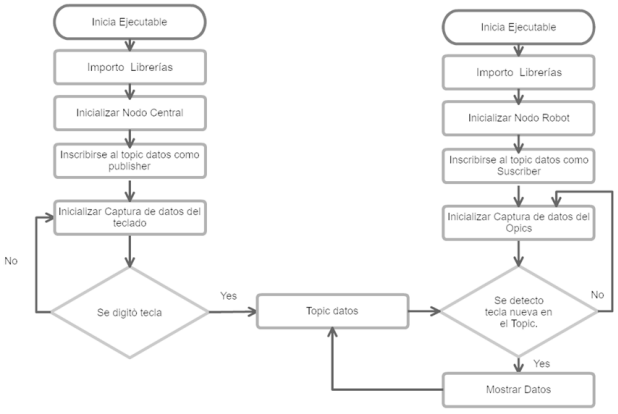

Fig. 13. Enviar datos del nodo / central y enviarlos al nodo / robot

- Tarjetas CRCMAN. En las tablas (Ver Tablas 4 y 5) se representan las clases con tarjetas que contienen la información de los nodos, métodos, relaciones y responsabilidades de la clase central y teclado que estarían en el computador con la clase Robot (que estarían implementada en el robot)

En la Tabla 4 se muestra la tarjeta CRCMAN de la iteración I la cual describe la clase Control ubicada en el nodo /Robot que se encarga de interpretar los datos que vienen de la clase de Comunicaciones para que el usuario pueda monitorear los datos del computador, con los métodos ingresarOrden() y avanceRobot()

Tabla 4. Tarjeta CRCMAN Iteración I

Nombre del nodo	Robot
Clase	Control
Colaboradores	Comunicaciones
Responsabilidad I (RI)	Interpretar los datos que vienen de la clase de comunicaciones para que el usuario pueda monitorear los datos del computador
Método I- Responsabilidad 1 (1RI) Ingreso de los datos recibidos por comunicaciones	ingresarOrden(data.data)
Método 2- Responsabilidad 2 (2RI) muestra un mensaje dependiendo de la instrucción enviada(8- arriba,2 -abajo,4-izquierda,6-derecha, cualquier otro alto)	avanceRobot()

En la Tabla 5 se muestra la tarjeta CRCMAN de la iteración II la cual modela la clase ConfigurarNodo ubicada en el nodo /Central que se encarga de facilitar la creación de nodos y el nombramiento de los Topics, con el método nombrarNodoPublisher(self,Nombrepublisher,Nombrenodo).

Tabla 5. Tarjeta CRCMAN - Iteración II

Nombre del nodo	Central
Clase	ConfiguraNodo
Colaboradores	Teclado
Responsabilidad I (RI)	Esta Clase debe facilitar la creación de nodos y el nombramiento de los Topics
Responsabilidad II (RII)	Debe permitir enviar letras tipo String por medio de esta clase al Topic
Método I- Responsabilidad 1 (1RI) inicializar y nombrar el nodo con su *topic*	nombrarNodoPublisher(self,Nombrepublisher,Nombrenodo):

A continuación, se incorpora la tarjeta CRCMAN de la iteración III la cual muestra la clase Teclado ubicada en el nodo /Central que se encarga de enviar datos capturados a la clase ConfigurarNodo, para luego ser enviados al Nodo /Robot, con el método def getchar(self).

Tabla 6. Tarjeta CRCMAN - Iteración III

Nombre del nodo	Central
Clase	Teclado
Colaboradores	ConfiguraNodo
Responsabilidad I (RI)	Capturar los datos del teclado
Responsabilidad II (RII)	Enviar datos capturados a la clase Configurar Nodo para ser enviado
Método I- Responsabilidad 1 (1RI) inicializar y nombrar el nodo con su *topic*	def getchar(self)

- Diagrama de clase por nodos. Se representan las clases control y comunicaciones en la (Ver Fig. 16) en el nodo Robot y el nodo central (Ver Fig. 18) con su clase Teclado y Configurarnodo

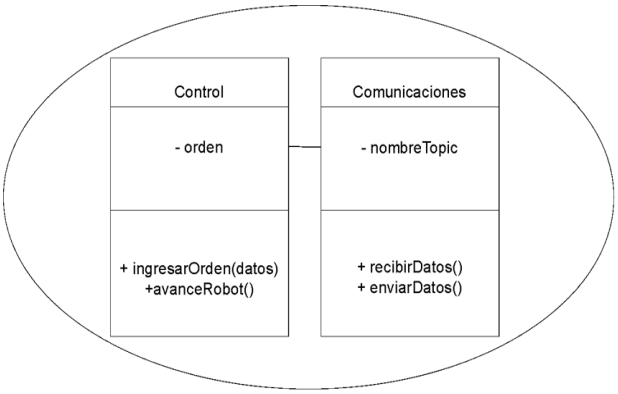

Fig. 16. Diagrama de clases de la primera Iteración – Nodo Central.

La Fig. 16 representa al nodo implementado en la plataforma robótica, encargado de recuperar los datos enviados del computador por medio de la clase Comunicaciones.

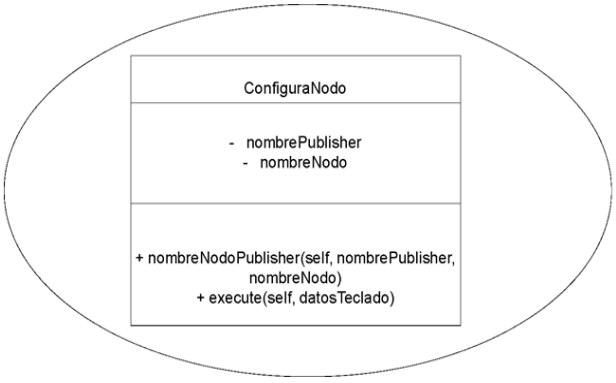

Fig.17. Diagrama de clases de la segunda Iteración – Nodo Central.

En la Fig. 17 se muestra el nodo implementado en el computador donde se crea la clase Configurarnodo que se encarga de crear el nodo / central para enviar los datos al robot.

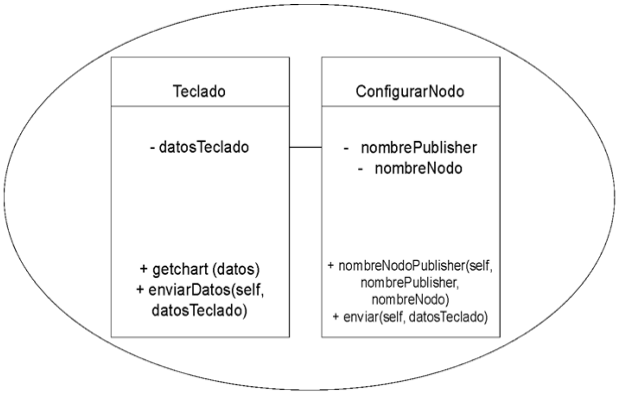

Fig. 18. Diagrama de clases de la tercera Iteración – Nodo Central.

En la Fig. 18, se agrega la clase “Teclado” al nodo / central, la cual se encarga de enviar los datos a la clase ConfigurarNodo que se encarga de enviar los datos al robot.

- Diagrama de secuencias. Se representan las relaciones entre los objetos Control y Comunicaciones del nodo Robot (Ver Fig. 19), y el objeto de la clase teclado y configurarNodo del nodo /Central (Ver Fig. 20).

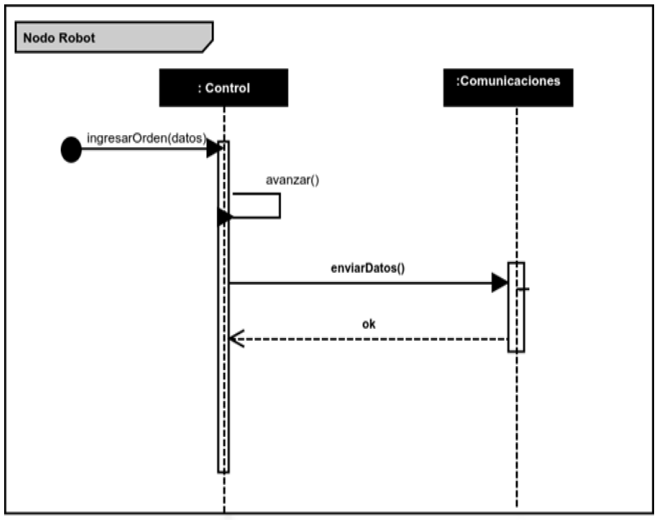

Fig. 19. Diagrama de secuencia para desarrollar la primera Iteración.

En la Fig. 19, se aprecia el comportamiento de la clase Control que se encarga de recibir los datos a la clase Comunicaciones, que, a su vez, se encarga de recuperar los datos del topic.

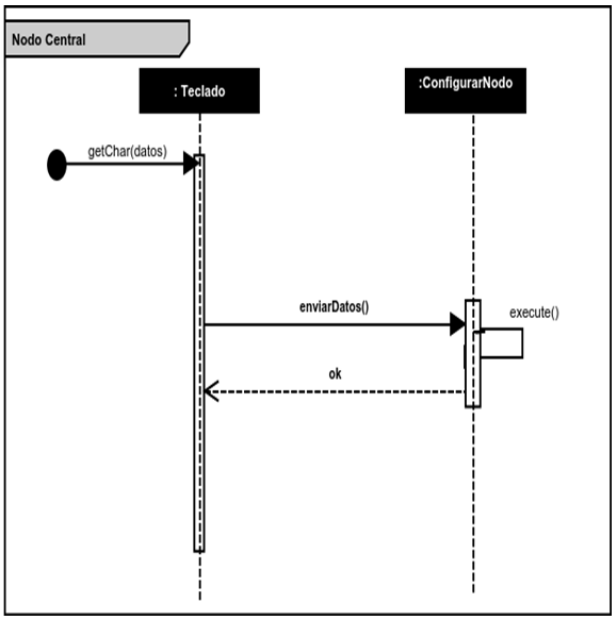

Fig 20. Diagrama de secuencias de la tercera Iteración.

En la Fig. 20, se muestra el comportamiento del nodo /Central que se encarga de enviar datos al topic.

4.3 Programación. La codificación para este caso es implementada en la plataforma colaborativa GitHUB

4.3.1 Creación del proyecto en la plataforma colaborativa

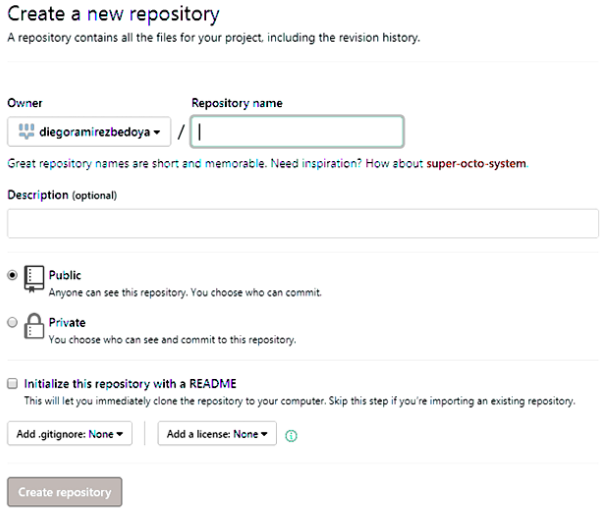

Fig. 21. Formulario para la creación del repositorio en GitHub.

Se utiliza la plataforma GitHub para la elaboración del proyecto donde se implementa el desarrolla el software para la teleoperación de la plataforma robótica Carlitos.

4.3.2 Creación de los nodos. Código en Python para crear nodos publicadores (Ver Tabla 5)

En la Tabla 5 se muestra el código necesario para crear un nodo sin incorporarlo en una clase (Ver Tabla 5)

Tabla 5. Código Python para crear nodos Publisher

```
if not rospy.is_shutdown():
rospy.loginfo(datoDelTeclado)
self.pub.publish(datoDelTeclado)
self.rate.sleep()
return True
else:
return False
```

4.3.3 Clases con métodos y atributos. Codificación básica de las clases diseñadas con sus métodos y atributos (Ver Tablas 6, 7 y 8).

En la Tabla 6 se muestra el código de la clase Control con sus atributos y métodos.

Tabla 6. Código Python de la clase Control

```
class Controls(object):
orden='5'
def ingresarOrden(self,orden):
def avanceRobot(self):
```

En la Tabla 7 se presenta el código de la clase ConfiguraNodo con sus atributos y métodos, la cual se encarga de publicar el nodo.

Tabla 7. Código Python de la clase configuranodo

```
class ConfiguraNodo:
def nombrarNodoPublisher(self,nombrPublisher,nombreNodo):

def execute(self,datoDelTeclado):
```

En la Tabla 8 se muestra código de la clase Teclado con sus atributos y métodos, la cual se encarga de publicar el nodo.

Tabla 8. Código Python de la clase Teclado

```
class Teclado(ConfiguraNodo):
def __init__(self, publisher,nodo):

def enviarDatos(self, datosTeclado):
```

4.3.4 Código de los métodos de las clases. Se visualiza la clase Control desarrollada para enviar mensajes según la captura de números específicos (Ver Tabla 9).

Tabla 9. Código Python de la clase Control

```
class Control(object):
    orden='5'
def ingresarOrden(self,orden):
    self.orden=orden
    print 'orden',self.orden
def avanceRobot(self):
    if self.orden =='8':
        print 'Arriba'
    elif self.orden=='2':
        print 'Abajo'
    elif  self.orden=='4':
        print 'Izquierda'
    else:
        print 'Derecha'
    print 'avance',self.orden
```

A continuación, se presenta el código necesario para la clase CreacionNodo que se encarga de implementar el nodo / central ubicada en el computador (Ver Tabla 10).

Tabla 10. Código Python de la clase configuranodo

```
class ConfiguraNodo:
def nombrarNodoPublisher(self,nombrPublisher,nombreNodo):
    self.pub = rospy.Publisher(publisher, String,
        queue_size=10)
    rospy.init_node(nodo, anonymous=True)
    self.rate = rospy.Rate(10) # 10hz

def execute(self,datoDelTeclado):

    if not rospy.is_shutdown():
    rospy.loginfo(datoDelTeclado)
    self.pub.publish(datoDelTeclado)
        self.rate.sleep()
            return True
        else:
            return False
```

Código completo de la clase KeyBoard.

Tabla 11. Código Python de la clase *Key*

```
class KeyBoard(ConfiguraNodo):
        def __init__(self, publisher,nodo):
        ConfiguraNodo.nombrarNodoPublisher(self, publisher,nodo)
        def enviarDatos(self, datosTeclado):
        ConfiguraNodo.execute(self, hello_str)

        def getchar(self):

          fd = sys.stdin.fileno()
          old_settings = termios.tcgetattr(fd)
          try:
            tty.setraw(sys.stdin.fileno())
            hello_str = sys.stdin.read(1)
          finally:
            termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
          return hello_str
```

4.4 Pruebas ROS-XP

4.4.1 Validación de pruebas unitarias

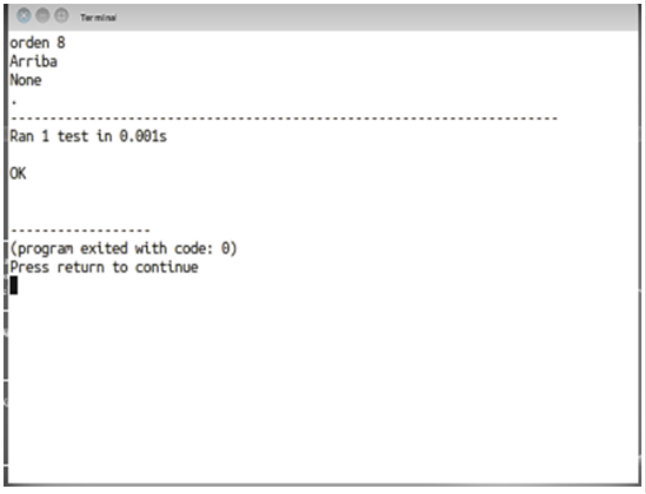

Fig. 22. Validación de que la clase Teclado funciona correctamente.

4.4.2 Pruebas unitarias en caso de fallo

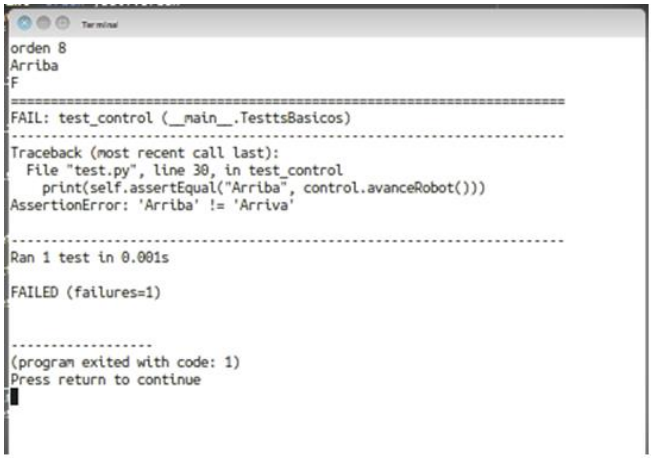

Fig. 23. Pantallazo que muestra la prueba fallida de la clase Teclado.

4.5 Producción

En esta parte se entrega el proyecto completo con el diagrama final y la aplicación funcionando.

4.5.1 Ajustes. Se cambia el tiempo entre fotograma para mejoramiento del video.

4.5.2 Diagrama final. Finalmente se plasma el gráfico (Ver Fig. 24)

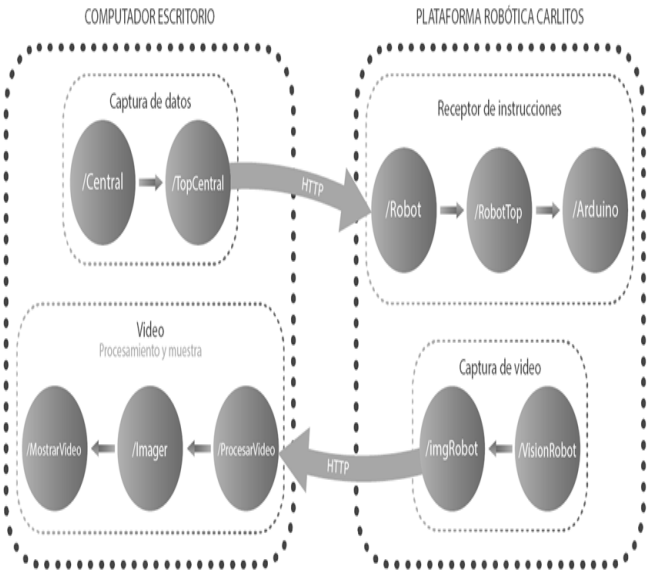

Fig. 24. Diagrama final con todos los nodos necesarios del proyecto

4.5.3 Puesta en marcha. Lista de verificación del software donde se puede apreciar, entre otras cosas, la captura de video desde la plataforma robótica y enviada al computador por medio de ROS (Ver Fig. 25).

Fig. 25. Video enviado desde el nodo / robot de la plataforma robótica a al nodo central

Los resultados de la metodología inician en la iteración, con una historia de usuario, para luego implementar el diseño. La arquitectura es sencilla, dado que predomina un grupo de diagramas como el de clases, secuencia y los grafos computacionales de ROS. El uso de las tarjetas CRCMAN incorpora el nombre de los nodos y métodos con sus responsabilidades.

Se aprecia una notable diferencia a otras metodologías como es el caso de la MDASR (Metodología de Desarrollo de Arquitectura de Software para Robots) debido a que utiliza el proceso unificado de desarrollo de software (PU), el cual usa el modelo predictivo para la construcción de aplicativos informáticos [11]. Por lo tanto, exige una rigurosidad en la planeación y el diseño porque busca obtener una predicción del comportamiento robótico, diferente a lo propuesto al trabajo presentado en este artículo dado que el diseño puede cambiar a medida que avanza el proyecto.

También se compara la metodología propuesta con la aproximación basada en UML para el diseño y codificación automática de plataformas robóticas manipuladoras de la Universidad de Jaen [12], que a diferencia de lo que plantea los autores de ROS-XP, reemplaza el grafo computacional por un diagrama de componentes, lo cual es aconsejable para desarrolladores en ingeniería de software, pero puede confundir a ingenieros o técnicos acostumbrados a trabajar con los diagramas o conceptos propios del ROS. Además, su enfoque no está orientado a un marco de trabajo para el desarrollo de software.

El resultado final del procedimiento sugerido por ROS-XP es un código en lenguaje Python implementado en los métodos de una clase. Lo anterior es aprovechando el diagrama de clases que define la estructura de cada nodo y el de secuencia, que contempla el comportamiento de las clases. Vale destacar, que los diagramas están constituidos por las entidades que afecten directamente cada nodo.

5. CONCLUSIONES

La metodología presentada en este artículo se diferencia de XP en que el diseño es obligatorio, pero sencillo el cual maneja un grupo de diagramas como el diagrama de clases, secuencia y diagramas computacionales de grafos de ROS. La modificación de la metodología XP (principalmente en la fase de diseño) permite adicionar documentación necesaria para darle rigurosidad a los desarrollos para la investigación.

El diagrama de clases está constituido por los objetos que afecten directamente al nodo, en los cuales se tienen en cuenta los atributos del objeto, y los métodos. El diseño es evolutivo, es decir que

se hace para los nodos que se van a entregar en cada iteración, pero estos pueden ser modificados en caso que la práctica lo requiera. La modularidad del código permite hacer cambios más rápidamente y concuerda totalmente con la modalidad iterativa e incremental que se ha sugerido durante todo el proyecto. Además, se ha adaptado eficientemente según el modelo de línea de la investigación en robótica. La teleoperación permitió entender cómo configurar ROS para la comunicación entre nodos, el envío de mensajes en diferente máquina, el manejo de aplicaciones distribuidas con ROS, configuración de la red, captura, procesamiento y el envío y visualización de imágenes.

El hecho de dividir las historias de usuario en entrada, procesamiento y salida genera un aumento de los números de nodos, lo cual es necesario para mejorar la cohesión y extensibilidad en la comunicación de los mismos. Por tanto, el desarrollador se ve obligado a que cada clase tenga relación directa con la función para la cual fue diseñada.

El caso de estudio de teleoperación permitió entender cómo configurar ROS para la comunicación entre nodos, el envío de mensajes en diferente máquina, el manejo de aplicaciones distribuidas con ROS, configuración de la red, captura, procesamiento y envío y visualización de imágenes.

A futuro se espera contemplar una segunda fase donde se aplique la metodología ROS-XP para el desarrollo software de muebles robóticos, que puedan dar soporte a pacientes con limitaciones de movilidad y, así, mejorar la calidad de vida de las personas que se identifican con la forma de los muebles, a fin de disminuir el posible trauma que pueda causar la incorporación de estos equipos en el hogar.

6. AGRADECIMIENTOS

Agradecemos al Doctor Carlos Mario Sierra Duque de la Universidad de Antioquia por su asesoría metodológica para la realización de este proyecto. También expresamos los agradecimientos al profesor Gustavo Alonso Acosta Amaya por sus valiosos aportes y al Sistema de Innovación e Investigación del Servicio Nacional de Aprendizaje (SENA), así como al grupo de articulación de la media técnica del Centro Tecnológico del Mobiliario de la misma institución.

7. REFERENCIAS BIBLIOGRÁFICAS

[1] T. Acharya y R. Ajoy, Image Processing-Principles and Applications, Arizona: John Wiley & Sons, 2005.

[2] A. Martínez y E. Fernández, Learning ROS for Robotics Programming, Birmingham - Mumbai: Packt, 2013.

[3] J. Lentin, Learning Robotics Using Python, Birmingham: Packt Publishing Ltd, 2015.

[4] J. Rapado, Diseño e implementación de una interfaz gráfica de usuario para mapeado de entornos y navegación en ROS. Valencia, 2016.

[5] S.R. Chidamber, D.P. Darcy, C.F. Kemerer, "Managerial use of metrics for object-oriented software: an exploratory analysis", Software Engineering IEEE Transactions on, vol. 24, no. 8, pp. 629-639, 1998.

[7] K. Beck, Extreme programming eXplained: Embrace change. Reading, MA: Addison-Wesley. Extreme programming eXplained: Embrace change. Reading, MA: Addison-Wesley, 2000.

[8] G. Martin, "UML for embedded systems specification and design: motivation and overview", Design Automation and Test in Europe Conference and Exhibition 2002. Proceedings, pp. 773-775, 2002.

[9] Stewart Baird, "Teach Yourself Extreme Programming in 24 Hours[M]", SAMS, 2003.

[10] P. Letelier, "Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP)," Técnica administrativa, vol. 5, pp. 26, June2006[Online]. Available:https://dialnet.unirioja.es/servlet/articulo?codigo=1983605

[11] N. Ospina, Arquitectura software para robots móviles aplicando la metodología MDASR. Avances en Sistemas e Informática, vol. 6, no 3, p. 133-144,2009.

[12] A. Sánchez, J. Gámez, J. Gómez, An UML based approach for designing and coding automatically robotic arm platforms, Revista Iberoamericana de Automática e Informática industrial, vol. 14, no. 1, pp. 629-639, 2017.