Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

# PROPUESTA DE UN PLAN OPERATIVO PARA IMPLEMENTACIÓN DE LA ADMINISTRACIÓN DEL CAMBIO E INTEGRACIÓN CONTINUA, EN EL DESARROLLO DEL SISTEMA OPERATIVO DE LA ASOCIACIÓN CIVIL EDULIBRE

**Julio Fernando Osorio Escobar**

Asesorado por la Inga. Mirna Ivonne Aldana Larrazábal

Guatemala, marzo de 2017

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA


FACULTAD DE INGENIERÍA

# PROPUESTA DE UN PLAN OPERATIVO PARA IMPLEMENTACIÓN DE LA ADMINISTRACIÓN DEL CAMBIO E INTEGRACIÓN CONTINUA, EN EL DESARROLLO DEL SISTEMA OPERATIVO DE LA ASOCIACIÓN CIVIL EDULIBRE

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

**JULIO FERNANDO OSORIO ESCOBAR**
ASESORADO POR LA INGA. MIRNA IVONNE ALDANA LARRAZÁBAL

AL CONFERÍRSELE EL TÍTULO DE

**INGENIERO EN CIENCIAS Y SISTEMAS**

GUATEMALA, MARZO 2017

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA


## NÓMINA DE JUNTA DIRECTIVA

| | |
|---|---|
| DECANO | Ing. Pedro Antonio Aguilar Polanco |
| VOCAL I | Ing. Ángel Roberto Sic García |
| VOCAL II | Ing. Pablo Christian de León Rodríguez |
| VOCAL III | Ing. José Milton de León Bran |
| VOCAL IV | Br. Jurgen Andoni Ramírez Ramírez |
| VOCAL V | Br. Oscar Humberto Galicia Nuñez |
| SECRETARIA | Inga. Lesbia Magalí Herrera López |

## TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

| | |
|---|---|
| DECANO | Ing. Pedro Antonio Aguilar Polanco |
| EXAMINADOR | Ing. Sergio Arnaldo Méndez Aguilar |
| EXAMINADOR | Ing. William Samuel Guevara Orellana |
| EXAMINADOR | Ing. José Alfredo González Díaz |
| SECRETARIA | Inga. Lesbia Magalí Herrera López |

# HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

**PROPUESTA DE UN PLAN OPERATIVO PARA IMPLEMENTACIÓN DE LA ADMINISTRACIÓN DEL CAMBIO E INTEGRACIÓN CONTINUA, EN EL DESARROLLO DEL SISTEMA OPERATIVO DE LA ASOCIACIÓN CIVIL EDULIBRE**

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha marzo de 2016.

**Julio Fernando Osorio Escobar**

Guatemala 04 de octubre de 2016

Ingeniero
Carlos Alfredo Azurdia
Coordinador de Privados y Revisor de Trabajos de Graduación
Escuela de Ciencias y Sistemas
Facultad de Ingeniería

Respetable Ing. Azurdia:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **JULIO FERNANDO OSORIO ESCOBAR**, con número de carnet **200714856**, titulado: **"PROPUESTA DE UN PLAN OPERATIVO PARA IMPLEMETACIÓN DE LA ADMINISTRACIÓN DEL CAMBIO E INTEGRACION CONTINUA EN EL DESARROLLO DEL SISTEMA OPERATIVO DE LA ASOCIACIÓN CIVIL EDULIBRE"**, y a mi criterio el mismo cumple con los objetivos propuestos al inicio del trabajo, por lo que procedo a dar mi aprobación.

Sin otro particular me suscribo de usted.

Atentamente,

Mirna Ivonne Aldana Larr
INGENIERA EN CIENCIAS Y SI
Colegiada No. 9567

Mirna Ivonne Aldana Larrazabal
Ingeniera en Ciencias y Sistemas
Colegiado No. 9567
Asesor de Trabajo de Graduación

Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 19 de Octubre de 2016

Ingeniero
**Marlon Antonio Pérez Türk**
**Director de la Escuela de Ingeniería**
**En Ciencias y Sistemas**

Respetable Ingeniero Pérez:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **JULIO FERNANDO OSORIO ESCOBAR** con carné **200714856,** titulado: **"PROPUESTA DE UN PLAN OPERATIVO PARA IMPLEMENTACIÓN DE LA ADMINISTRACIÓN DEL CAMBIO E INTEGRACIÓN CONTINUA EN EL DESARROLLO DEL SISTEMA OPERATIVO DE LA ASOCIACIÓN CIVIL EDULIBRE",** y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

**Ing. Carlos Alfredo Azurdia**
Coordinador de Privados
y Revisión de Trabajos de Graduación

ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÌA
ESCUELA DE INGENIERÍA EN
CIENCIAS Y SISTEMAS
TEL: 24767644

*El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del asesor con el visto bueno del revisor y del Licenciado en Letras, del trabajo de graduación* ***"PROPUESTA DE UN PLAN OPERATIVO PARA IMPLEMENTACIÓN DE LA ADMINISTRACIÓN DEL CAMBIO E INTEGRACIÓN CONTINUA, EN EL DESARROLLO DEL SISTEMA OPERATIVO DE LA ASOCIACIÓN CIVIL EDULIBRE",*** *realizado por el estudiante JULIO FERNANDO OSORIO ESCOBAR aprueba el presente trabajo y solicita la autorización del mismo.*

***"ID Y ENSEÑAD A TODOS"***

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION DE INGENIERIA EN CIENCIAS Y SISTEMAS

*Ing. Murlon Antonio Pérez Türk*
***Director***
***Escuela de Ingeniería en Ciencias y Sistemas***

*Guatemala, 13 de marzo de 2017*

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

DTG. 140.2017

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al Trabajo de Graduación titulado: **PROPUESTA DE UN PLAN OPERATIVO PARA IMPLEMENTACIÓN DE LA ADMINISTRACIÓN DEL CAMBIO E INTEGRACIÓN CONTINUA, EN EL DESARROLLO DEL SISTEMA OPERATIVO DE LA ASOCIACIÓN CIVIL EDULIBRE,** presentado por el estudiante universitario: **Julio Fernando Osorio Escobar,** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

Ing. Pedro Antonio Aguilar Polanco
Decano

Guatemala, marzo de 2017

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

/gdech

## ACTO QUE DEDICO A:

| | |
|---|---|
| **Dios** | Por ser quien me dio la vida, porque me dio salud, porque me dio fuerzas, porque me dio sabiduría para seguir luchando. |
| **Mi padre** | Julio Osorio, quien aún en su enfermedad mantuvo mi carrera, y por proporcionarme las herramientas para mi camino de aprendizaje. |
| **Mi madre** | Sara Escobar de Osorio, por quererme, por cuidarme, por orar por mí, por apoyarme incluso con sus problemas y dificultades, y porque siempre estuvo para mí. |
| **Mis hermanos** | Julio Alejandro, por enseñarme a ayudar a los demás; Lesly, por enseñarme a seguir lo que quieres; Josué, por enseñarme a colaborar cuando nace de corazón; y Natalia, por enseñarme a luchar por los estudios. |
| **Mi novia** | Mi amada Marlyn González, por enseñarme a seguir luchando a pesar de los problemas, por ser un gran apoyo en la lucha que tuve para realizar mi trabajo de graduación, e incluso en mi vida profesional y personal. |

# AGRADECIMIENTOS A:


**Mis abuelas** Evangelina Chávez y Julia Sierra, por el amor que me dieron, por los ánimos que recibí y porque me enseñaron a ser una persona trabajadora aun con la gran cantidad de problemas que tenga uno en la vida.

**Mis amigos** Con quienes estuve en el camino de aprendizaje, superando juntos las dificultades, apoyándonos, animándonos para seguir la lucha del aprendizaje.

**Universidad** La Universidad de San Carlos de Guatemala, por ser el lugar donde logré mi formación como profesional.

**Mi asesora** Inga. Invonne Aldana, por ser un gran apoyo en la realización de mi trabajo de graduación y compartir su gran experiencia como ingeniera en ciencias y sistemas.

**Pueblo de Guatemala** Porque gracias al pago de sus impuestos, un porcentaje es destinado para la formación de profesionales egresados de la Universidad de San Carlos de Guatemala, a quienes me estoy incluyendo.

# ÍNDICE GENERAL

# ÍNDICE DE ILUSTRACIONES

## FIGURAS



## TABLAS

# LISTA DE SÍMBOLOS

| Símbolo | Significado |
| --- | --- |
| **IDE** | Entorno de desarrollo |
| **CI** | Integración continua |

# GLOSARIO

**Ambiente** Es el entorno de ejecución de un programa.

**Base de datos** Sistema computarizado para administrar y gestionar información.

***Build*** La construcción de programa utilizando archivos fuente y generando archivos ejecutables.

**Código fuente** Conjunto de archivos que contienen instrucciones de código en un lenguaje de programación.

**Componente** Elemento computarizado que contiene una funcionalidad especifica.

**Conectividad** Se refiere a la comunicación de un ambiente hacia otros sistemas y ambientes.

***Firewall*** Elemento de red o computarizado que permite o deniega accesos al ambiente.

***Framework*** Es un entorno de trabajo que provee componentes, módulos y librerías con funcionalidades específicas.

**Parámetro** Es una variable de entrada a la que se le puede especificar un valor.

| | |
|---|---|
| ***Proxy*** | Es un elemento físico o computarizado que filtra y permite el acceso al contenido *web*. |
| **Puerto** | Es una interfaz para recibir y enviar datos en diferentes protocolos de enlace de datos. |
| **Red** | Conjunto de dispositivos de red interconectados. |
| ***Release*** | Es el conjunto de archivos binarios construidos con la compilación del código fuente, para ser instalados en un ambiente de producción con un versionamiento especifico. |
| ***Roll back*** | Revertir los cambios realizados en un ambiente después de una instalación. |
| **Servicio** | Sistema computarizado que provee una funcionalidad especifica. |
| **Sistema operativo** | Conjunto de programas, controladores y recursos que controlan un ordenador. |
| ***Software*** | Conjunto de programas que permite al ordenador realizar una tarea específica. |
| ***Tester*** | Rol cuya tarea principal es realizar una rutina o plan de pruebas. |

| | |
|---|---|
| **Usuario** | Persona que utiliza el sistema y/o se identifica para que se le permita acceso. |
| **Versionamiento** | Es etiquetar con una nomenclatura definida los cambios realizados en la línea de tiempo de un elemento. |

# RESUMEN

El presente trabajo de graduación describe el proyecto de creación de la propuesta de un plan operativo para implementación de la administración del cambio e integración continua, en el desarrollo del sistema operativo de Edulibre. Esta es una asociación civil sin fines de lucro que busca llevar una mejor educación a los niños de primaria, a través del uso de tecnología en el aula, proporcionando laboratorios de computación en diferentes escuelas públicas del país, utilizando *software* libre en los equipos.

La propuesta del plan operativo realizado para esta asociación busca oportunidades de mejora, reducción de tiempo en la detección de errores y buenas prácticas en el desarrollo de su sistema operativo: EdulibreOS, implementando integración continua y administración del cambio. El plan operativo contiene:

- Identificación de oportunidades de mejora.
- Definición de roles del equipo de trabajo.
- Definición de un proceso de pruebas automatizadas.
- Definición del proceso de administración de configuración.
- Identificación de herramientas para pruebas automatizadas e integración continua.

# OBJETIVOS

## General

Elaborar un plan operativo para la Asociación Civil Edulibre, que sirva de guía para implementar una correcta administración del cambio e integración continua en el proceso de desarrollo de su sistema operativo.

## Específicos

1. Elaborar un documento de la metodología utilizada actualmente en el proceso de desarrollo del sistema operativo de Edulibre, identificando sus fases y una breve descripción de cada una.

2. Identificar al menos cinco oportunidades de mejora por medio de la administración del cambio e integración continua, basadas en el documento antes mencionado.

3. Establecer al menos seis puntos de control para que el desarrollo del sistema operativo de Edulibre se pueda basar en la administración del cambio e integración continua, especificando las condiciones que se deberán cumplir en cada punto de control y la acción que se deberá tomar.

4. Definir al menos cuatro roles del equipo de trabajo del desarrollo del sistema operativo de Edulibre. A cada rol se le definirán al

menos cinco funciones que podrá realizar, y al menos cuatro responsabilidades que deberá cumplir.

5. Identificar la cantidad de tiempo que toma encontrar errores en el proceso de desarrollo actual, realizando encuestas del tiempo tomado en estos procesos por el equipo en el presente.

6. Establecer un proceso de pruebas automatizadas para reducir la cantidad de tiempo que toma encontrar errores en el proceso de desarrollo actual al menos en un 20 %.

7. Establecer un proceso de control de configuración para reducir los problemas de configuraciones de servicios, puertos, accesos y versiones al menos en un 30 %, comparando con el proceso actual.

8. Identificar al menos dos herramientas para el manejo de integración continua, realizando un cuadro comparativo con las características de cada herramienta, para que Edulibre pueda utilizar una, seleccionando la que mejor se adecue a la asociación.

9. Identificar al menos tres herramientas para realizar pruebas automatizadas, realizando un cuadro comparativo con las características de cada herramienta, para que Edulibre pueda utilizar una, seleccionando la que mejor se adecue a la asociación.

# INTRODUCCIÓN

Edulibre es una asociación civil sin fines de lucro que proporciona laboratorios con equipo de computación en distintas escuelas públicas del país. Lleva educación a los niños mediante el uso de su sistema operativo EdulibreOS, basado en la distribución GNU/Linux Ubuntu. Este sistema operativo contiene programas y videojuegos educativos que proporcionan a los niños la oportunidad de aprender jugando. La distribución del sistema operativo EdulibreOS es propiamente guatemalteca e, igual que el sistema operativo en que está basada, tiene una licencia general pública, para que cualquier persona pueda tomarla y modificarla.

El presente trabajo de graduación propone un plan operativo de administración del cambio e integración continua para el desarrollo del sistema operativo EdulibreOS, en el cual están contenidos los programas y juegos educativos con enfoque para niños de educación primaria. El plan operativo especificará el proceso para control de configuraciones, proceso de pruebas automatizadas, buenas prácticas para la implementación de la administración del cambio e integración continua en el desarrollo del sistema operativo, e identificación de herramientas útiles para pruebas automatizadas e integración continua.

# 1. MARCO TEÓRICO

## 1.1. Control de versiones

Un sistema de control de versiones es una herramienta que permite manejar los cambios realizados sobre cualquier documento o archivo, especialmente al código fuente de un proyecto. Estos sistemas brindan a los desarrolladores la facilidad de seleccionar un punto en el tiempo y obtener versiones anteriores del código, permitiendo regresar a versiones funcionales, detectar errores, recuperar archivos e incluso un proyecto que haya sido eliminado por error.

Estos sistemas brindan una fuente centralizada, en donde cualquier persona que tenga acceso puede obtener el proyecto, lo que ayuda a los desarrolladores a trabajar en conjunto pero desde distintos lugares, y observar los cambios realizados por otros miembros del equipo de trabajo, además de controlar y resolver cualquier conflicto que surja por dichos cambios. También, por cada cambio, el sistema mantiene un registro que incluye la fecha y hora, el nombre del usuario y un comentario que brinda más información sobre el cambio realizado.

Es común que en los equipos de trabajo se maneje un estándar para nombrar cada versión, y en general es habitual utilizar una numeración decimal como estándar. En dicha numeración cada cifra indica la importancia del cambio, quedando a discreción del equipo nombrar la cantidad de decimales a utilizar y los cambios que entran en cada cifra. Generalmente, un cambio en la primera cifra indica una versión nueva, y las otras cifras pueden indicar la

introducción de una nueva funcionalidad a cierta versión, el arreglo de un error, entre otras cosas.

Existen diversas herramientas para el control de versiones, pero en general para hacer posible todo lo mencionado se hace uso de lo que comúnmente se conoce como repositorio y copia de trabajo. El repositorio es la fuente principal y centralizada que se encuentra en un servidor y en donde se almacena el proyecto o cualquier archivo del cual se pretende llevar un control. Mientras que una copia de trabajo es una copia del repositorio, que puede obtener cualquier miembro del equipo para trabajar sobre ella. Así es como cada persona puede trabajar desde cualquier lugar y luego subir los cambios al repositorio para que el resto del equipo los pueda ver.

En su estructura se encuentra la línea principal conocida como *trunk*, y a partir de esta línea es posible obtener copias que permitan trabajar de forma aislada y sin arruinar el resto del proyecto; estas copias se conocen como *branch* o ramas. Esto es útil para el manejo de versiones, tanto como para no arruinar algo de la versión anterior como para diferenciarlas, ya que al completar la nueva versión el controlador brinda una función para unir la o las ramas con la línea principal. Al trabajar sobre una rama los cambios se realizan solo sobre esta, por lo que existe independencia entre ramas y la línea principal, hasta que se unen.

Para el manejo de los archivos, los controladores de versiones ofrecen ciertas operaciones básicas, las cuales son:

- *Create*
- *Checkout*
- *Commit*

- *Update*
- *Add*
- *Edit*
- *Delete*
- *Rename*
- *Move*
- *Status*
- *Diff*
- *Revert*
- *Log*
- *Tag*
- *Branch*
- *Merge*
- *Resolve*
- *Lock*
- *Clone*
- *Push*
- *Pull*

La integración continua corre sobre la línea principal del controlador de versiones, por lo que es fundamental implementar un sistema de control de versiones para generar integración continua.

## 1.2. Integración continua

Martin Fowler la define en el libro *Continuous integration* como: “una práctica de desarrollo de *software* donde los miembros de un equipo deben integrar su trabajo con frecuencia, por lo general cada persona integra al

menos una vez al día, lo que conduce a múltiples integraciones por día. Cada integración es verificada por una construcción automatizada (incluyendo pruebas) para detectar los errores de integración lo más rápidamente posible. Muchos equipos encuentran que este enfoque conduce a una reducción significativa de problemas de integración y permite a un equipo desarrollar el *software* de cohesión más rápidamente"[1].

La integración continua se originó con las prácticas de desarrollo de *Extreme Programming*. La integración continua principalmente provee la reducción de riesgos, *software* de calidad y comunicación entre los desarrolladores del equipo de trabajo. Es una práctica de desarrollo donde los desarrolladores deberán realizar un *commit* de sus cambios al repositorio principal, de una forma frecuente para que los desarrolladores del equipo de trabajo siempre tengan disponibles los últimos cambios realizados al proyecto. Antes de realizar un *commit* al repositorio principal del proyecto, el desarrollador deberá descargar la última versión del repositorio principal y si tiene cambios a la versión actual de la que dispone, deberá realizar los cambios necesarios, ajustando sus cambios a la última versión del repositorio. Deberá realizar una compilación sin errores y realizar pruebas que se deberán cumplir al 100 %, si no contiene errores el desarrollador podrá realizar un *commit* al repositorio principal para agregar sus cambios en el código fuente. Para que el desarrollador pueda terminar con su trabajo deberá compilar el código desde el repositorio principal hacia el servidor de integración donde se proporcionará la funcionalidad de los cambios realizados.

Una práctica muy importante en la integración continua es mantener un único repositorio de origen, donde todos los desarrolladores podrán realizar

[1] FOWLER, Martin. *Continuous integration*. http://martinfowler.com/articles/continuousIntegration.html. Consulta: 20 de febrero de 2016.

sus *commit* de los cambios realizados a los códigos fuentes del proyecto. Deberán agregar al *script* de cambios todo lo que utilizaron desde fuentes, librerías, *scripts* de pruebas, configuraciones de IDE. Esto para que cualquier desarrollador que desea realizar un *check out* de repositorio principal hacia su ambiente local, lo pueda ejecutar sin ningún problema. La regla definida por Martin Flower en el libro de *Continuous integration* dice: “La regla básica es que debe ser capaz de caminar hasta el proyecto con una máquina virgen, hacer obtención y ser capaces de construir completamente el sistema”[2]. Lo que indica que aun teniendo una máquina con únicamente los requerimientos mínimos de ejecución del sistema, al obtener el código fuente del repositorio principal, pueda ser posible construirlo completamente en el sistema sin tener errores de falta de dependencias.

La automatización de construcción es una práctica importante, debido a que se realizarán muchos cambios al día, de los distintos *commit* de los desarrolladores del equipo de trabajo, por lo que se deberá facilitar la construcción de una forma automatizada, haciendo uso de una herramienta con la que, al notificarse un cambio exitoso en el repositorio principal, se construya automáticamente. La realización de un auto *testing* permitirá encontrar errores más rápidamente cuando se realiza un cambio en el código fuente y se construye. Se podrán realizar pruebas permitiendo encontrar errores de una forma más fácil y rápida.

Una práctica muy importante donde todo mundo se compromete con el *commit* en el repositorio principal, se refiere a que cada miembro del equipo de trabajo realice pequeños cambios frecuentemente al repositorio principal para que todos los desarrolladores se encuentren comunicados y enterados de los

[2] FOWLER, Martin. *Continuous integration*. http://martinfowler.com/articles/continuousIntegration.html. Consulta: 20 de febrero de 2016.

cambios del proyecto y sea mucho más fácil la integración. Esto se realiza al menos una vez por día.

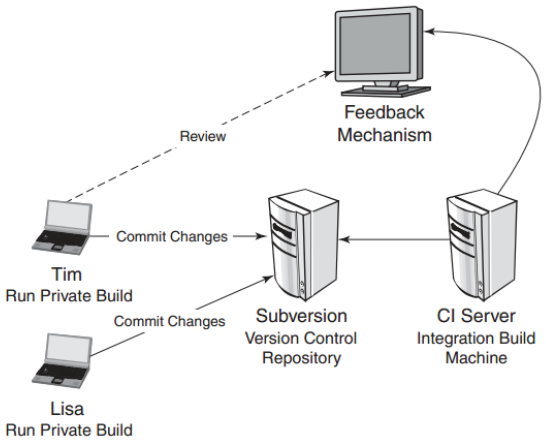
Cada *commit* deberá construir la línea principal a la máquina integradora, es decir que cada *commit* de cada desarrollador notificará al servidor de integración que hay cambios disponibles, y podrá realizar una construcción automática con los cambios o notificar al encargado de la construcción que existe un nuevo cambio. De tal forma que al construir el sistema en la máquina integradora, se puedan ejecutar las pruebas automáticamente para detectar errores más rápidamente.

Cuando ocurre un error en la línea principal todos deberían estar involucrados para resolverlo. Este problema puede ocurrir pero debe resolverse a la brevedad; es la prioridad de todos los desarrolladores del equipo de trabajo. No significa que todo el equipo de trabajo detenga sus actividades y resuelvan el problema de la línea principal, sino que a cualquier desarrollador del equipo de trabajo que le ocurra este error en la línea principal con la ayuda del supervisor de la construcción de la línea principal deberán resolver el problema como su prioridad. Como primer paso deberán bajar el proyecto a un ambiente local y, sobre ese ambiente, solucionarlo y regresar la línea principal a la última versión estable, mientras corrigen el error y realizan el *commit* nuevamente.

Es muy importante que nadie realice un *commit* con sus cambios cuando el estado de la línea principal sea erróneo, por la simple razón de que corregir los errores sería mucho más complicado y requeriría de un mayor esfuerzo. Tampoco nadie debería obtener las fuentes del proyecto de la línea principal cuando está en un estado erróneo.

La integración continua es una práctica que todos los miembros del equipo de trabajo deberán estar comprometidos a realizar con los puntos anteriormente mencionados y el resumen de su flujo está representado en la figura 1.

Figura 1. **Integración continua**




Fuente: DUVALL, Paul M.; MATYAS, Steven; GLOVER, Andrew. *Continuous integration.* P. 26

## 1.3. Pruebas automatizadas

Las pruebas automatizadas son aquellas que se programan para ser realizadas sobre el código a través de un *software* dedicado, no dejando de lado las pruebas manuales, ya que es necesario que los desarrolladores o encargados de las pruebas las codifiquen. En el artículo *¿Cómo enfoco el testing de forma ágil?*, se menciona que: “automatizar esas pruebas va a facilitarle el trabajo a los *testers*, en el sentido de que pueden dedicarse al *testing* de verdad, nuevo, que vaya aportando valor en cada iteración: *testeo* de nuevas funcionalidades, búsqueda de errores más difíciles de encontrar, probar

nuevas características. En definitiva, actividades que requieran más de su experiencia"[3].

Las pruebas en general deben ser realizadas cada vez que se realiza un cambio en la estructura del proyecto, no importando lo pequeño que haya sido dicho cambio, ya que podría afectar a otros componentes. La detección temprana de errores permite al equipo de desarrollo actuar rápidamente y prevenir catástrofes; además de realizar pruebas brinda confianza a los interesados en el proyecto. Sin embargo, las pruebas implican un costo monetario y una inversión de tiempo.

Existen metodologías, como el desarrollo basado en pruebas, en las que las pruebas son tan fundamentales que deben escribirse antes de codificar la aplicación a desarrollar. Y es que, como se menciona en el documento *Symfony 1.4, la guía definitiva*: "En ocasiones, las pruebas automatizadas pueden reemplazar la documentación técnica de la aplicación, ya que ilustran de forma clara el funcionamiento de la aplicación. Un buen conjunto de pruebas muestra la salida que produce la aplicación para una serie de entradas de prueba, por lo que es suficiente para entender el propósito de cada método"[4].

Existen diversos tipos de pruebas que pueden realizarse a diferentes niveles y ambientes, como por ejemplo las pruebas unitarias, de componentes, de sistema, de integración, de carga/rendimiento, de seguridad, entre otras. En todo código es necesario siempre probar las pequeñas funcionalidades que se implementan y validar que aún sigan funcionando las que ya estaban implementadas, y es por ello que existen dos tipos de pruebas importantes que

[3] GARCÍA, Ana. *¿Cómo enfoco el testing de forma ágil?* http://www.javiergarzas.com/2015/01/testing-agil.html.

[4] FRANCOIS, Fabien. *Symfony 1.4, la guía definitiva.* http://librosweb.es/libro/symfony_1_4/.

deben realizarse constantemente, y al ser realizadas periódicamente es importante automatizarlas. Estos tipos de pruebas son:

- Pruebas unitarias: se realizan sobre pequeñas porciones de código para probar una función en particular y bajo ciertas circunstancias. En general, el código a probar debe ser pequeño e independiente de otros componentes. Sirven para probar métodos/funciones con entradas específicas y verificar que devuelvan salidas específicas, por lo que para un método/función pueden realizarse distintas pruebas unitarias.

- Pruebas funcionales: cuando una unidad de código está ligada a otras y es necesario probar todo un bloque, deja de ser una prueba unitaria y se convierte en una prueba funcional. Estas pruebas validan procesos y escenarios completos. Sin embargo, para escenarios más complejos las pruebas unitarias y funcionales se quedan cortas.

## 1.4. Control de configuración

El estándar IEEE, mencionado en el libro *Configuration management principles and practices*, define de manera completa y resumida lo que el control de configuración es: “una disciplina que aplica una dirección técnica y administrativa y un control para: identificar y documentar las características funcionales y físicas de los *ítems* de configuración, controlar los cambios de estas características, grabar y reportar cambios de estado procesado e implementado, y verificar la complacencia con los requerimientos especificados”[5].

[5] JONASSEN, Anne. *Configuration management principles and practices.* Boston, Pearson Education Inc., 2003. 432 p.

El control de configuración debe asegurarse de comprender cada uno de los aspectos que maneja la empresa, tanto así que debe volverse parte de la cultura general de la compañía. El control de la configuración debe abarcar distintas perspectivas para su completa implementación, y para ello el libro recién citado menciona seis perspectivas a tomar en cuenta:

- Perspectiva de las personas: implica la creación y asignación de roles.

- Perspectiva del producto: es necesario entender el producto que la compañía vende o genera y de esta forma adaptar el control.

- Perspectiva del proyecto: se debe comprender el ciclo de vida de las actividades que se llevan a cabo en la compañía.

- Perspectiva de toda la organización: es importante tomar en cuenta los objetivos, metas, misión y visión de la empresa.

- Perspectiva de procesos: cada uno de los procesos que se manejan y que se planean implementar deben ser entendidos para su mantenimiento y mejora continua.

- Perspectiva de herramientas: el control de configuración necesita de herramientas para su implementación, las cuales pueden ser creadas desde cero o utilizarse unas que ya existan.

Es importante para la integración continua la utilización de un sistema de control de configuraciones, ya que este sistema se encarga de manejar el versionamiento del código fuente, lo que incluye su almacenamiento y proveer a los desarrolladores del equipo. Se apoya del control de versiones, pero el

control de configuraciones maneja no solo el código fuente, sino todas aquellas configuraciones, archivos, librerías y demás objetos necesarios para construir un ambiente de trabajo estándar, el cual puede ser obtenido por los desarrolladores de tal manera que todos posean un mismo ambiente y con las últimas actualizaciones.

Ya que el control de configuración varía conforme la empresa y su cultura, no hay una forma específica de implementarlo, sin embargo, en el documento *Software configuration management best practices*, se mencionan cinco pasos bastante generales a tomar en cuenta para su implementación:

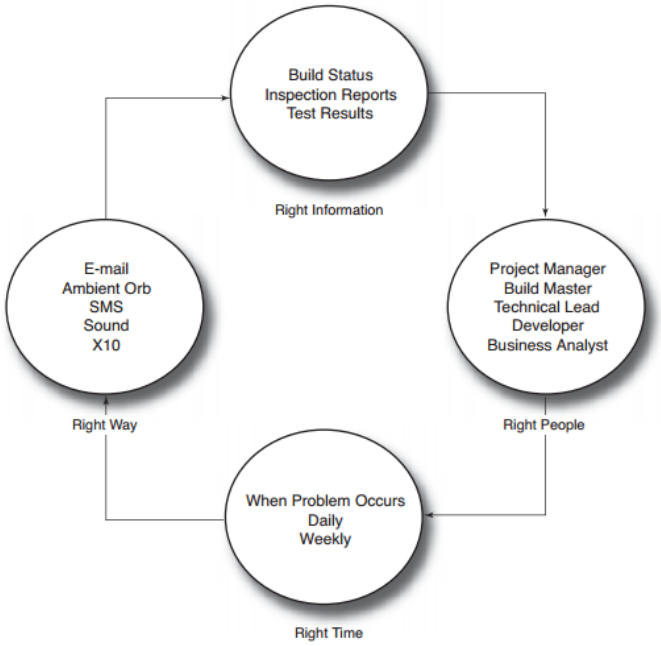
- Crear un plan
- Establecer un repositorio
- Desarrollar un proceso
- Definir los *ítems* de configuración correctos
- Ir más despacio[6]

## 1.5. Retroalimentación

La retroalimentación es muy importante en la implementación de integración continua, ya que informa el estado de una *build* a las personas correctas. Una retroalimentación rápida y constante es saludable en la integración continua porque informa el estado satisfactorio o fallido de un *build*, una construcción de una versión del proyecto, a las personas correctas, utilizando el medio correcto, proporcionando la información correcta en el tiempo correcto.

[6] Accurev. *Software configuration management best practices.* http://resources.idgenterprise.com/original/AST-0062469_AccuRev-SCM-Best-Practices.pdf.

Figura 2. **Elementos correctos de retroalimentación continua**



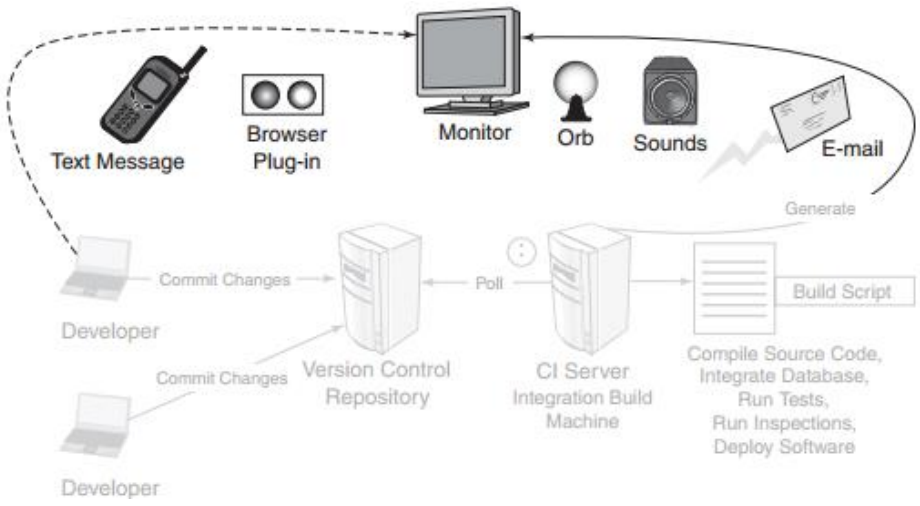

Fuente: DUVALL, Paul M.; MATYAS, Steven; GLOVER, Andrew. *Continuous integration.* P. 206

Las personas correctas son todas aquellas relacionadas al proyecto, personas interesadas en el proyecto como administradores, desarrolladores, expertos del negocio, arquitectos de *software* y *tester*.

La información correcta proporciona el estado del *build*, de la construcción del proyecto, resultados de instalación y de pruebas, que pueden ser satisfactorias o fallidas, indicando el detalle de cada una de ellas.

El medio correcto son todos los mecanismos disponibles para notificar la información de la construcción del proyecto, utilizando correo electrónico, mensajes a través de teléfono móvil, entre otros.

Figura 3. **Mecanismos para retroalimentación continua**




Fuente: DUVALL, Paul M.; MATYAS, Steven; GLOVER, Andrew. *Continuous integration.* P.19

El tiempo correcto es inmediatamente después de que ocurrió el evento que puede ser: construcción, instalación, ejecución de pruebas. Cuando finaliza algún evento se deberá notificar inmediatamente a todas las personas involucradas con los medios disponibles.

La retroalimentación proporcionará buenas prácticas en el caso de que todo haya sido satisfactorio, así como rápida detección de problemas para su corrección, si fue insatisfactorio.

# 2. EDULIBRE

## 2.1. Antecedentes de la empresa

Actualmente el sector educativo en Guatemala cuenta con muchas deficiencias, entre las cuales se encuentra la falta de tecnología en las escuelas e institutos. En 2014, la página periodística La Hora publicó un artículo en línea en donde se informaba sobre las estadísticas del Ministerio de Educación en cuanto al área de tecnología, y se mencionaba que solamente el 5 % de las escuelas e instituciones del país contaban con laboratorios de computación.

Edulibre es una asociación civil sin fines de lucro, que nació de la iniciativa de un grupo de estudiantes de Ingeniería en Ciencias y Sistemas de la Universidad de San Carlos de Guatemala, quienes vieron la necesidad de las escuelas del país en cuanto al sector de tecnología. Estos estudiantes no solo detectaron una necesidad sino que además vieron una oportunidad para aplicar sus conocimientos en favor del país y de reutilizar *hardware*.

Edulibre obtiene *hardware* a partir de donaciones de terceros, que van desde personas particulares hasta empresas, que poseen *hardware* que ya no utilizan pero que puede servirle a otras personas. Esta asociación se encarga de recoger el material, darle mantenimiento, limpiarlo y repararlo si fuera necesario. Luego instalan el *software* en el equipo y ellos mismos se encargan de pintar el lugar en donde se realizará la instalación; así también, colocan todo el equipo y cableado necesario para el funcionamiento del laboratorio. Por último, brindan capacitación al personal de la escuela en donde se realizó la instalación.

Hoy en día Edulibre brinda la oportunidad de que las personas puedan realizar lo que ellos llaman un apadrinamiento individual, a través del cual se brinda ayuda económica cada mes, a cambio de mantenerse informados sobre los avances del *software* y de los laboratorios que se inauguran gracias a su aportación. Además existe un programa de voluntariado, en el cual las personas pueden avocarse a la asociación y ayudarlos en su labor. Este programa ha sido fomentado mayormente en distintas universidades como la Universidad de San Carlos, Universidad del Valle y Universidad Mariano Gálvez.

Figura 4. **Paso a paso Edulibre**


Fuente: Edulibre. *Paso a paso*. http://edulibre.net/index.php/paso-a-paso/. Consulta: 06 de abril de 2016.

## 2.2. Historia

Para hablar de la historia de esta asociación, es necesario observar algunos de los aspectos que conforman su perspectiva.

### 2.2.1. Misión

"Desarrollamos soluciones de tecnología de información de código abierto y asesoramos en su uso en las escuelas de nivel primario, integrándolas a su práctica pedagógica."[7]

### 2.2.2. Visión

"Brindamos la oportunidad a las niñas y niños de Latino América de tener acceso a una Educación de calidad a través de tecnologías de información, guiados por los principios de código abierto."[8]

### 2.2.3. Objetivos

La asociación Edulibre define los siguientes objetivos:

- General
  - "Promover el uso de un Sistema Operativo con programas y herramientas educativas totalmente libres por medio de EdulibreOS a los niños y estudiantes guatemaltecos (as) o de

[7] EDULIBRE. *Página oficial*. http://edulibre.net/index.php/about-us/. Consulta: 20 de febrero de 2016.
[8] Ibíd.

cualquier parte de Latinoamérica, mediante el aprendizaje tecnológico”[9].

- Específicos
  - “Crear una plataforma totalmente educativa.
  - Que los usuarios sean capaces de utilizar las herramientas educativas que se les ofrece.
  - Promover el aprendizaje de los niños o de los estudiantes.
  - Facilitar herramientas tecnológicas libres”[10].

## 2.3. Sistema operativo

Es útil detenerse a observar el desarrollo de este sistema operativo.

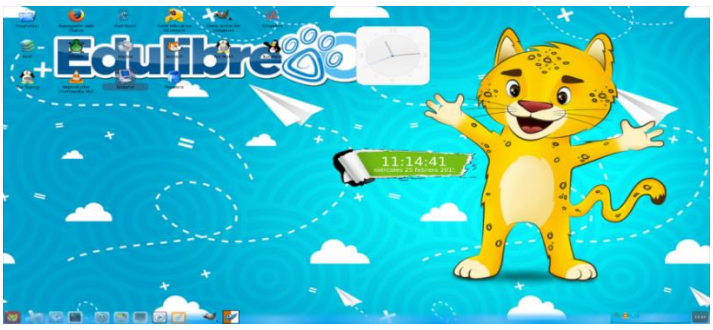
### 2.3.1. Historia

La idea de Edulibre no fue solo la de brindar *hardware*, sino que de igual manera, para contribuir a la educación de los niños, era necesario crear *software* que se pudiera instalar en las computadoras y que permitiera a aquellos reforzar sus conocimientos. Las personas de Edulibre se dieron cuenta de cómo, a pesar de que para ellos utilizar un ordenador era una tarea fácil, para los niños no, ya que muchos de ellos, sino es que su totalidad, jamás habían utilizado una computadora. Entonces vieron la necesidad de crear *software* que permitiera a los niños aprender a utilizar una computadora de una manera divertida y también de crear *software* que permitiera a los niños aprender sobre otras materias.

[9] EDULIBRE. *Página oficial.* http://edulibre.net/index.php/objetivos/. Consulta 20 de febrero de 2016.
[10] Ibíd.

Es así como nace EdulibreOS, el cual ellos describen en su página web como: “un sistema operativo basado en Ubuntu GNU/Linux creado en Guatemala por Herberth Guzmán, diseñado y construido pensando en los centros educativos de nivel primario, básico y diversificado, introduciendo al usuario al mundo de la computación y tecnología. Además brinda una gran cantidad de herramientas que servirán de apoyo en el desarrollo de las habilidades intelectuales y creativas de los usuarios. El sistema también está pensado para el uso hogareño, para que toda aquella persona que esté en busca de una herramienta educativa de calidad para sus hijos lo encuentre en EdulibreOS.”[11] La versión 1.0 de EdulibreOS se lanzó en junio de 2008 en Guatemala y en la actualidad se ha extendido a Centro América. Cuenta con más de 120 aplicaciones educativas y 400 actividades para el desarrollo de las capacidades tecnológicas de los niños, tal y como lo indican en su página web. Las personas que integran Edulibre dedican su tiempo a desarrollar *software* educativo de código abierto, y además a obtener *hardware* para realizar las instalaciones en las escuelas. La última versión lanzada a la fecha es la de EdulibreOS-V7.2, basada en Ubuntu 14.04 LTS.

Figura 5. **Escritorio de EdulibreOS 7.2**




Fuente: Edulibre. *Versiones*. http://edulibre.net/index.php/versiones/. Consulta: 10 de abril de 2016.

[11] EDULIBRE. *Página oficial.* http://edulibre.net/index.php/inicio/. Consulta: 20 de febrero de 2016.

### 2.3.2. Características

Los integrantes de Edulibre pensaron en todo, ya que para alcanzar su meta ellos sabían que era probable que el equipo donado careciera de recursos, e inclusive que fuese necesario aprovechar al máximo cualquier tipo de *hardware* que se les ofreciere. Por eso era necesario tener un sistema operativo que pudiese funcionar bien con una cantidad mínima de recursos, por lo que se eligió Ubuntu GNU/Linux, que es el sistema operativo en el que está basado EdulibreOS. EdulibreOS 7.2 cuenta con tres presentaciones, las cuales demandan distintos requisitos mínimos pero que se siguen adaptando al hardware de bajos recursos:

- EdulibreOs 7.2 Liviana
  - Procesador 1 Ghz
  - Memoria RAM de 512 MB
  - Disco duro de 6 GB (con *swap* incluido)

- EdulibreOS 7.2 Completa x86
  - Procesador 1.8 Ghz
  - Memoria RAM de 512 MB
  - Disco duro de 10 GB (con *swap* incluido)

- EdulibreOS 7.2 Completa x64
  - Procesador 1.8 Mhz
  - Memoria RAM de 1 GB
  - Disco duro de 10 GB (con *swap* incluido)

# 3. FASE DE INVESTIGACION

## 3.1. Metodología de desarrollo

En esta sección se pretende identificar y documentar la forma en que los miembros del equipo de trabajo de la asociación Edulibre realizan el desarrollo de su *software*. Actualmente no se cuenta con una metodología de desarrollo claramente definida y documentada, pero se han logrado identificar algunas de las características más importantes sobre la forma en que se maneja el equipo y su desarrollo.

### 3.1.1. Fases

La asociación Edulibre posee dos desarrollos importantes, su Sistema Operativo EdulibreOS y el escritorio Innova. Las fases del ciclo de vida de desarrollo identificadas se presentan para cada uno a continuación.

- Sistema Operativo EdulibreOS

Las fases identificadas para el desarrollo del sistema operativo EdulibreOS son:

  - Fase de selección de la distribución: consiste en identificar la distribución que se utilizará como base para la creación del sistema operativo EdulibreOS, donde el coordinador del desarrollo

del sistema operativo identificará qué archivos se utilizarán para construir EdulibreOS.

- Fase de configuración de archivos: consiste en configurar y personalizar los archivos identificados en la fase anterior, que cumplan con las características y requerimientos para la creación de una nueva versión de EdulibreOs.

- Fase de ejecución y depuración: consiste en tomar todos los archivos que formarán el sistema operativo EdulibreOS. En estos se incluyen los configurados o personalizados por los desarrolladores de EdulibreOS, que se ejecutarán sobre un sistema operativo anfitrión basado en la distribución Debian o derivados, para que sea ejecutado pudiendo detectar errores y corregirlos.

- Fase de construcción: consiste en tomar todos los archivos que formarán al sistema operativo EdulibreOS y procesarlos para crear la ISO, que contendrá el sistema operativo para que pueda ser instalado posteriormente.

- Fase de instalación y pruebas: consiste en tomar la ISO construida e instalarla en una máquina física o virtual, para validar primero su correcta instalación. Luego se realizarán pruebas funcionales para validar su correcto funcionamiento.

- Escritorio Innova

Las fases identificadas para el desarrollo del escritorio Innova se listan a continuación definiendo una breve descripción de cada una de ellas:

  - Identificación de requerimientos: consiste en identificar nuevas funcionalidades para su escritorio, mejoras o corrección de errores. Se realiza un listado de los requerimientos identificados y se identifica una prioridad.

  - Fase de análisis: consiste en analizar el requerimiento para identificar claramente lo que se tiene que hacer, definiendo la funcionalidad del requerimiento.

  - Fase de diseño: consiste en realizar un breve diseño de lo que se va a desarrollar para lograr cumplir con la funcionalidad exitosa del requerimiento identificado.

  - Fase de desarrollo: consiste en desarrollar la funcionalidad en un lenguaje de programación, realizando en esta fase pruebas unitarias para ir validando la correcta programación.

  - Fase de pruebas: consiste en probar la funcionalidad desarrollada realizando pruebas funcionales, para verificar su correcto funcionamiento o realizar detección de errores.

El escritorio de Innova se incluye en el sistema operativo EdulibreOS, cuando ambos pasaron por todas sus fases. Se tiene un

sistema operativo EdulibreOS con el escritorio Innova y se realiza la fase de pruebas de usuario:

- Fase de pruebas de usuario: consiste en instalar el sistema operativo EdulibreOS con el escritorio Innova en el laboratorio de una escuela. Los usuarios lo utilizarán y validarán el correcto funcionamiento, o detectarán algunos errores.

- Fase de lanzamiento: consiste en publicar una nueva distribución estable para que pueda ser descargada por las personas que la desean utilizar.

### 3.1.2. Entregables

Edulibre tiene dos grandes proyectos, su sistema operativo EdulibreOS y su escritorio Innova, cada uno de estos tiene una serie de entregables que se presentan a continuación.

- Sistema Operativo EdulibreOS

El sistema operativo cuenta con varias fases dentro de las cuales se manejan distintos entregables, los cuales se presentan a continuación:

- Fase de distribución: en esta fase se crea un jaula *chroot* que contiene el sistema de archivos (incluye carpetas y archivos) necesarios para el sistema operativo EdulibreOS. El entregable es la jaula chroot.

- Fase de configuración: en esta fase se modifican y personalizan los archivos de la jaula *chroot*; el entregable es la jaula *chroot* con los archivos configurados.

- Fase de instalación y pruebas: en esta fase se ejecuta y depura el sistema operativo, se realizan las correcciones y el entregable es la jaula *chroot* con los archivos corregidos, si aplican para alguna corrección.

- Fase de construcción: en esta fase se crea la ISO del sistema operativo; el entregable es la ISO que contiene el sistema operativo.

- Fase de instalación y pruebas: se instala el sistema operativo EdulibreOS; el entregable es una confirmación de pruebas satisfactoria o un reporte de errores.

- Escritorio Innova

El escritorio Innova posee distintos entregables que son resultado de cada fase de su ciclo de vida, y se presentan a continuación:

- Fase de identificación de requerimientos: en esta fase el entregable es un listado de requerimientos a ser desarrollados.

- Fase de análisis: en esta fase el entregable es una breve descripción del requerimiento, indicando claramente la funcionalidad.

- Fase de diseño: en esta fase el entregable es una breve descripción del diseño, indicando claramente cómo se pretende realizar el desarrollo.

- Fase de desarrollo: en esta fase el entregable son los archivos fuentes del lenguaje de programación para que cualquier otro miembro del equipo lo pueda ejecutar.

- Fase de pruebas: en la fase de pruebas, el entregable es el instalador del escritorio Innova con extensión *deb*.

### 3.1.3. Ambientes

Los ambientes son los entornos utilizados por los miembros del equipo de Edulibre para el ciclo de vida del desarrollo de su sistema operativo EdulibreOs y de su escritorio Innova. Existen ambientes utilizados para desarrollo y para la realización de pruebas, los cuales se presentan a continuación.

- Desarrollo

El ambiente de desarrollo es caracterizado por tener la versión de desarrollo de la distribución base seleccionada que se actualiza una vez por semana, para identificar los módulos nuevos, cambios en los módulos anteriores y en la eliminación de módulos, para que los desarrolladores del sistema operativo adapten EdulibreOS a los cambios del sistema operativo en que está basado.

- Pruebas funcionales

El ambiente de pruebas generalmente lo forman máquinas virtuales creadas por los integrantes de Edulibre. En cada máquina virtual se instala la última versión en desarrollo del sistema operativo EdulibreOS.

- Pruebas de usuario

El ambiente de pruebas de usuario es el laboratorio de una escuela a la que se le instala la última versión estable del sistema operativo EdulibreOS que estaba en desarrollo.

### 3.1.4. Roles

Los roles definen la función de cada miembro del equipo de trabajo de Edulibre, un miembro puede tener más de un rol de acuerdo a los miembros disponibles o experiencia que posean. A continuación se presentan los roles que poseen actualmente con una breve descripción de su función.

- Administrador de proyectos

Se encarga de identificar los requerimientos para el proyecto de sistema Operativo EdulibreOS y para el proyecto Innova, coordinando la planificación en el desarrollo y ejecución de pruebas, hasta su lanzamiento.

- Coordinador de desarrollo

Se encarga de identificar la distribución base del sistema operativo EdulibreOS, así como de identificar los archivos necesarios para su correcto

funcionamiento y la identificación de los archivos que se deben configurar. Además integra el desarrollo del sistema operativo EdulibreOS y su escritorio Innova.

- Desarrollador

Se encarga de configurar los archivos del sistema operativo EdulibreOS identificados por el coordinador y del desarrollo de módulos del escritorio Innova.

- Encargado de pruebas

Se encarga de realizar pruebas funcionales del sistema operativo EdulibreOS y de su escritorio Innova, aprobando su correcto funcionamiento o notificando sus errores.

- Usuarios

Personas fuera de los miembros de Edulibre, que se encargan de utilizar el sistema normalmente, indicando si encontraron o no alguna funcionalidad incorrecta.

## 3.2. Equipo de trabajo

Son los activos que posee Edulibre para el desarrollo del sistema operativo EdulibreOS y el escritorio Innova. Dichos activos son representados por los miembros del equipo de trabajo que desempeñan uno o más roles y las herramientas que utilizan para su desarrollo, como computadoras, servidor, Internet, entre otros.

### 3.2.1. Personal

El personal se refiere al capital humano que forma parte de la asociación Edulibre, incluyendo a voluntarios que se involucran en el desarrollo de *software*. A continuación se listan los miembros que forman parte del equipo presentados por rol.

- Administrador de proyectos

Actualmente es solo una persona la que se encarga de administrar los proyectos de Edulibre, estos incluyen el desarrollo de sistema operativo EdulibreOS y el desarrollo de su escritorio Innova. También maneja más proyectos, como la instalación de laboratorios en las escuelas y el desarrollo de programas o videojuegos educativos.

- Coordinador de desarrollo

Actualmente es solo una persona la encargada de coordinar el desarrollo del sistema Operativo EdulibreOs y del escritorio Innova. Realiza el desarrollo mayor en ambos proyectos.

- Desarrollador

Actualmente ellos tienen voluntarios que los apoyan en el desarrollo del sistema operativo EdulibreOS y del escritorio Innova, generalmente son 1 ó 2 personas más que apoyan en el desarrollo.

- Encargado de pruebas

Actualmente son cuatro miembros del grupo de trabajo de Edulibre, las pruebas son realizadas por estos cuatro miembros, donde se incluye al administrador de proyectos y coordinador de desarrollo.

### 3.2.2. Infraestructura y equipo

Debido a que es una asociación sin fines de lucro, su infraestructura y equipo son limitados. Se especifica a continuación el equipo que actualmente poseen:

- Computadoras

Disponen de cuatro computadoras, cada computadora es para cada uno de los miembros del equipo de trabajo de Edulibre.

- Servidor

Disponen de un servidor que contiene la página *web* con la información de Edulibre y permite la descarga de las versiones estables lanzadas de su sistema operativo.

- Internet

Disponen de una conexión a Internet para uso de los miembros de EdulibreOs y de acceso al repositorio de versiones de EdulibreOS.

## 3.3. Detalles técnicos del código fuente del sistema operativo

Los detalles técnicos del código fuente del sistema operativo EdulibreOS y el escritorio Innova se presentan brevemente a continuación.

### 3.3.1. Tecnologías utilizadas

Edulibre utiliza únicamente *software* libre. Entre las tecnologías con que cuentan están:

- Sistema Operativo GNU/Linux Debian o derivados
- Lenguaje de programación Gambas
- Casper para la creación de ISO

### 3.3.2. Versionamiento

Hoy en día en Edulibre no manejan un versionamiento claramente definido. Actualmente el coordinador de desarrollo, en el desarrollo del escritorio Innova, crea una nueva versión interna cuando valida un funcionamiento estable. Tanto para el sistema operativo EdulibreOs como para el escritorio Innova se crea una nueva versión cuando se realiza un lanzamiento. Estos son los únicos dos controles de versionamiento de los que se tiene seguimiento.

## 3.4. Pruebas

Las pruebas son utilizadas para detectar errores que puedan ser corregidos. Las pruebas no solo incluyen detección de errores, sino que también identificación de mejoras sobre una funcionalidad, que apoyan a la creación de un producto de calidad.

### 3.4.1. Tipos de pruebas

Dependiendo del tipo de prueba que se realice se encontrarán diferentes tipos de errores. Cada tipo de error tiene un tiempo promedio diferente para su detección.

- Pruebas unitarias

Las pruebas unitarias se realizan en la fase de ejecución y pruebas para el sistema operativo EdulibreOS, y en la fase de desarrollo del escritorio Innova. Las pruebas generalmente consisten en probar la configuración de uno o unos pocos archivos del sistema operativo para que no existan errores y probar los métodos programados para el escritorio, validando que no existan errores de compilación y que el método realice su funcionalidad correctamente.

Actualmente no existe un plan de pruebas para las pruebas unitarias del sistema operativo o escritorio, estas pruebas son las que al desarrollador se le ocurren probar después de realizar una configuración o una programación. Además, generalmente la detección de errores de estas pruebas es rápida, ya que es cuestión de unos minutos, pero no se puede constatar que prueben distintos escenarios.

- Pruebas funcionales

Las pruebas funcionales se realizan en la fase de instalación y pruebas del sistema operativo, y en la fase de pruebas del escritorio. Las pruebas consisten en utilizar los programas o videojuegos que contiene el sistema operativo para evaluar su correcta funcionalidad.

Actualmente no cuentan con un plan de pruebas funcionales, ya que cada encargado de pruebas realiza las pruebas que cree convenientes y evalúa su resultado, pudiendo este ser satisfactorio, insatisfactorio o si se debería mejorar. Generalmente la detección de errores en este tipo de pruebas es cuestión de unas horas, pero es posible que no se validen todos los escenarios ni se realicen pruebas anteriores de nuevo, si estas fueron satisfactorias.

- Pruebas de usuario

Las pruebas de usuario se realizan cuando se crea una versión de sistema operativo estable de la última versión en desarrollo y se instala en el laboratorio de una escuela. La prueba consiste en que los usuarios del sistema operativo, maestros y estudiantes, utilicen el sistema operativo con normalidad. Al pasar dos semanas con el sistema operativo se realiza la evaluación sobre su funcionalidad.

## 3.5. Oportunidades de mejora

Las oportunidades de mejora son puntos identificados por Edulibre y por la implementación del plan operativo, para identificar mejoras sobre el proceso actual en el desarrollo e implementación del sistema operativo EdulibreOS y su escritorio Innova. Las oportunidades de mejora identificadas son las siguientes:

- Implementación de un control de versiones para el desarrollo del sistema operativo EdulibreOS y el escritorio Innova.
- Implementación de un repositorio para obtener actualizaciones de los programas contenidos en su sistema operativo.
- Diseño de un plan de pruebas unitarias y funcionales para mejorar los tiempos en la detección de errores.

# 4. FASE DE DISEÑO

## 4.1. Definición del proyecto

El proyecto consiste en diseñar un plan operativo que le sirva a Edulibre como guía para el desarrollo de su sistema operativo EdulibreOS y su escritorio Innova, además de la implementación de integración continua y administración del cambio. Para ello se proponen herramientas y prácticas que deberán implementarse paulatinamente para lograr dichos objetivos, con el propósito de llevar un mejor control en el desarrollo tanto del sistema operativo EdulibreOS como del escritorio Innova, así como de contar con una detección rápida y temprana de errores. También se plantea el uso de herramientas de automatización de pruebas que permitan crear desarrollos con mayor frecuencia y que proporcionen funciones útiles al sistema operativo y a su escritorio.

## 4.2. Definición de roles

En todo equipo de trabajo es de gran importancia definir roles. Los roles especifican las funciones y responsabilidades del miembro del equipo. Es importante que cada miembro del equipo conozca cuál es su rol en el equipo de trabajo y cuáles son las funciones y responsabilidades con las que debe cumplir. La definición de roles proporciona organización, coordinación y comunicación en el equipo de trabajo. Los roles con sus funciones y responsabilidades que se proponen para la implementación de la integración continua y administración del cambio en el desarrollo del sistema operativo EdulibreOS y el escritorio Innova se definen a continuación:

Tabla I. **Definición de roles del equipo de desarrollo de Edulibre**

| | IDENTIFICADAS | PROPUESTAS | |
|---|---|---|---|
| ROL | **Funciones** | **Funciones** | **Responsabilidades** |
| Administrador de proyectos | ▪ Identificar los requerimientos para el proyecto.<br>▪ Coordinar la planificación del desarrollo y ejecución de pruebas. | ○ Administrar los requerimientos de los nuevos proyectos o mantenimientos del sistema operativo y escritorio, priorizando los requerimientos en conjunto con el coordinador de desarrollo.<br>○ Verificar el cumplimiento del desarrollo del proyecto de acuerdo a su planificación en conjunto con el coordinador del proyecto.<br>○ Coordinar el desarrollo y pruebas de los proyectos de acuerdo a la prioridad establecida en conjunto con los coordinadores de desarrollo y pruebas.<br>○ Coordinar instalaciones y actualizaciones del sistema operativo en las escuelas. | ✓ Llevar el control de la finalización de desarrollo de los proyectos con una carta de finalización firmada por el coordinador de desarrollo y el desarrollador responsable.<br>✓ Llevar el control de la finalización de pruebas de los proyectos con una carta de certificación firmada por el coordinador y encargado de pruebas. |
| Coordinador de desarrollo | ▪ Identificar la distribución base del sistema operativo.<br>▪ Identificar archivos para el funcionamiento del SO.<br>▪ Identificar archivos de configuración. | ○ Administrar los proyectos de desarrollo y mantenimiento del sistema operativo y del escritorio, validando los tiempos de desarrollo y el cumplimiento de los requerimientos. | ✓ Llevar el control de finalización de desarrollo con una carta firmada por el desarrollador responsable.<br>✓ Solicitar la ejecución de pruebas para proyectos finalizados. |

Continuación de la tabla I.

| | | | |
|---|---|---|---|
| | ▪ Integrar EdulibreOS e Innova | ○ Asignar proyectos a los desarrolladores y darles seguimiento a los avances.<br>○ Priorizar los proyectos de desarrollo en conjunto con el administrador de proyectos.<br>○ Presentar al administrador de proyectos los avances y estados de los proyectos. | . |
| Desarrollador | ▪ Configurar archivos de EdulibreOS.<br>▪ Desarrollar módulos. | ○ Realizar una breve planificación de tareas sobre el desarrollo a realizar y validarlas con el coordinador de desarrollo.<br>○ Realizar el desarrollo del proyecto de acuerdo a la planificación aprobada.<br>○ Notificar sobre los avances al coordinador de desarrollo.<br>○ Realizar pruebas unitarias y funcionales. | ✓ Notificar la finalización del proyecto.<br>✓ Firmar carta de finalización. |
| Encargado de pruebas | ▪ Realizar pruebas funcionales.<br>▪ Aprobar funcionamiento de EdulibreOS e Innova.<br>▪ Notificar errores. | | |
| Usuario | ▪ Utilizar el sistema e informar sobre cualquier error o falla. | | |

Continuación de la tabla I.

| | | | |
|---|---|---|---|
| Coordinador de pruebas | | ○ Administrar la ejecución de pruebas a proyectos finalizados.<br>○ Asignar responsables de ejecución de pruebas.<br>○ Revisar el plan de pruebas en dónde se especifican los escenarios a evaluar y pruebas a realizar. | ✓ Solicitar las cartas de finalización para ejecutar las pruebas.<br>✓ Validar el plan de prueba para cada proyecto.<br>✓ Solicitar y firmar la carta de certificación de pruebas. |
| Ejecutor de pruebas | | ○ Crear el plan de pruebas para cada proyecto.<br>○ Validar el plan de pruebas con el coordinador de pruebas.<br>○ Ejecutar las pruebas e informar sobre los errores encontrados, especificando de forma clara el error y los valores con los que este se dio. | ✓ Firmar carta de certificación de pruebas. |
| Colaborador | | ○ Identificar problemas o mejoras en el sistema operativo y el escritorio.<br>○ Presentar información sobre errores encontrados o mejoras identificadas. | ✓ Utilizar el sistema operativo con frecuencia para detectar problemas y/o mejoras. |

Fuente: elaboración propia.

## 4.3. Manejo de versiones

Para el desarrollo del sistema operativo EdulibreOS y del escritorio de Innova, se deberá implementar un control de versiones para ambos desarrollos, definiendo el proceso y la estructura para manejar el control de versiones de

ambos desarrollos y comparando herramientas que permitan llevar el control de versiones adecuadamente.

### 4.3.1. Definición del proceso de control de versiones

Para llevar un correcto control de versiones en el desarrollo del sistema operativo EdulibreOS y del escritorio Innova, se definirá la propuesta del versionamiento formada por el *trunk* del proyecto y sus *branchs*. También se definirá el proceso que se debe seguir para llevar un adecuado y organizado control de versiones.

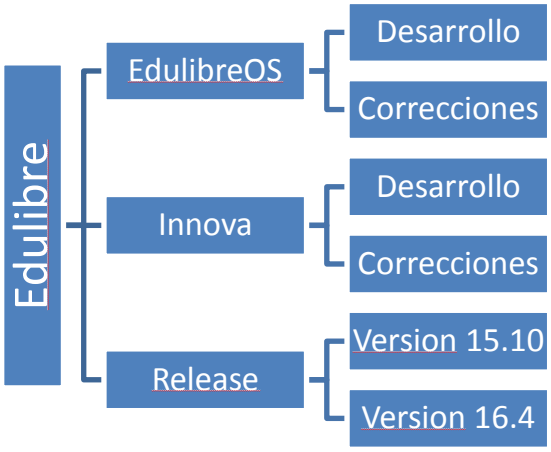
La estructura de trabajo del repositorio estará formada por el *trunk*, la base principal del proyecto. Se dividirá en tres branchs principales: el primero deberá ser utilizado para el sistema operativo EdulibreOS, el segundo para el escritorio Innova y el tercero para los *releases* que contendrán los archivos utilizados en los lanzamientos de las versiones de EdulibreOS con el escritorio Innova incluido.

El *branch* del sistema operativo EdulibreOS se dividirá en dos *branchs*, el *branch* para desarrollo y el *branch* para corrección de errores. El *branch* para desarrollo se utilizará para el control de versiones de los nuevos desarrollos asociados a EdulibreOS y el *branch* de correcciones se utilizará para llevar el control de versiones de la corrección de errores de los desarrollos finalizados asociados a EdulibreOS a los que se les realizan pruebas.

El *branch* del escritorio Innova se dividirá en dos *branchs*, el *branch* para desarrollo y el *branch* para corrección de errores. El *branch* para desarrollo se utilizará para el control de versiones de los nuevos desarrollos asociados a Innova y el *branch* de correcciones se utilizará para llevar el seguimiento del

control de versiones de la corrección de errores de los desarrollos finalizados asociados a Innova a los que se les realizan pruebas.

Figura 6. **Estructura de control de versiones para Edulibre**




Fuente: elaboración propia.

En el proceso de control de versiones los miembros del equipo de desarrollo deberán realizar una serie de prácticas, que al aplicar adecuadamente lograrán un organizado control de versiones con la mínima cantidad de problemas de integración en el desarrollo y pruebas de sistema operativo EdulibreOS y el escritorio Innova.

Primero, los miembros del equipo de trabajo deberán tener una copia local del repositorio al que le realizarán cambios en desarrollo o al que se le realizarán correcciones. El miembro que realice un cambio estable deberá realizar un *merge* a su repositorio local del código del repositorio principal agregando los cambios que pudieran existir a su repositorio local; el miembro deberá integrar los cambios correctamente, cuando el proyecto ya no contenga

errores realizando un *commit* de su repositorio local al repositorio principal, agregando las nuevas funcionalidades.

Todos los miembros del equipo de trabajo deberán realizar *commit* con la mayor frecuencia posible, mínimo una vez al día, con el fin de evitar problemas de integración con los desarrollos y correcciones realizadas por los demás miembros del equipo de trabajo.

Segundo, cuando se finalice el desarrollo de un proyecto y se pase a un ambiente de pruebas, se deberá realizar un *merge* del *branch* de desarrollo hacia el *branch* de correcciones, para tener la versión del código que está en pruebas sin que sean afectadas por los nuevos desarrollos. Si en las pruebas se encontraran errores, el miembro del equipo encargado en el desarrollo del proyecto asociado deberá corregirlas como su prioridad; si está en un desarrollo nuevo deberá detenerlo temporalmente y descargar en su repositorio local el *branch* de desarrollo del repositorio principal y realizar las correcciones. Cuando las correcciones se hayan realizado se deberá realizar el segundo paso.

Tercero, cuando las pruebas realizadas finalicen satisfactoriamente se decidirá si los cambios realizados en el *branch* de correcciones se realizan haciendo un *merge* al *branch* de *release* para agregar la nueva funcionalidad al próximo lanzamiento. Es importante que todos los miembros del equipo estén comprometidos a realizar los pasos indicados para tener un adecuado y organizado control de versiones. Si en algún momento algún *branch* principal tiene un error, ningún miembro del equipo deberá trabajar con el *branch* con problemas, es decir no podrá realizar *merge* o *commit.* El miembro del equipo de trabajo que provocó el error, en conjunto con el coordinador de desarrollo, deberán corregir el problema y estabilizar el *branch* principal a la brevedad,

para no afectar la integración del desarrollo de los demás miembros del equipo de trabajo.

### 4.3.2. Herramientas propuestas

Anteriormente se mencionó la importancia de utilizar un sistema de control de versiones en todo proyecto de desarrollo. En esta sección se plantea definir tres herramientas que se adaptan a las necesidades de Edulibre y que contienen distintas ventajas y desventajas. A partir de estas se tomará la propuesta a implementar:

- Subversion: herramienta *open source* desarrollada por Apache Software Foundation, basada en CVS, multiplataforma que cuenta con un modelo cliente-servidor. Es utilizada por diferentes proyectos, como por ejemplo Google Code y posee *plug-ins* para distintos IDEs. Cuenta con distintos clientes que se adaptan a cada plataforma y que en general ofrecen interfaces amigables al usuario, brindando una mayor facilidad de uso, ya que además es una herramienta fácil de aprender, aunque es lenta en su funcionamiento. Aún posee algunos problemas provenientes de su origen, CVS. Esta herramienta puede ser descargada de su página oficial sin ningún costo. En esta misma página se puede encontrar documentación e información sobre la comunidad que promueve la organización para que otras personas se ayuden y reporten errores e incluso mejoras sobre esta herramienta.

- Git: al igual que Subversion, es una herramienta gratuita, *open source*, multiplataforma, pero que maneja un modelo distribuido en el que cada miembro del equipo posee una copia local del proyecto sobre la que realiza cambios, implementando también un modelo de *branching*, ya

que cada copia es independiente de la otra. Al final cada miembro del equipo sube sus cambios, realizando un *merge.* En su página oficial se puede encontrar distinta documentación en forma de enlaces, vídeos y libros. Y de igual manera posee una comunidad que también brinda ayuda. Cuenta con un mayor soporte para Linux. Git es tan popular que también posee distintos clientes provistos por terceros que brindan una interfaz para el manejo de versiones. Además es una herramienta con alta velocidad, sin gran demanda de recursos, pero cuyo aprendizaje es más lento y complicado para aquellos que son principiantes en el mundo del control de versiones, aunque al final vale la pena. De igual manera existe *software* que brinda repositorios en la nube para Git, como es el caso de GitHub y GitLab. Aunque para estos existe una versión gratuita y una pagada.

- *Mercurial*: herramienta gratuita, *open source*, independiente de la plataforma y que maneja un modelo distribuido al igual que Git. Combina la velocidad de Git con la facilidad de aprendizaje de Subversion, además, brinda escalabilidad y es extensible. Posee una interfaz *web* y en su página oficial se encuentra una gran cantidad de documentación.

  De igual manera, permite la integración con *software* de terceros que brindan clientes con una interfaz agradable al usuario y que facilita el uso y aprendizaje de la herramienta. Brinda extensiones propias, pero también acepta extensiones de terceros.

### 4.3.3. Cuadro comparativo de herramientas

Con base en la investigación realizada y en la información presentada de cada herramienta, se muestra el siguiente cuadro comparativo:

Tabla II. **Cuadro comparativo de herramientas de control de versiones**

| | Subversion | Git | Mercurial |
|---|---|---|---|
| **Ventajas** | ✓ Es *open source.*<br>✓ Es gratuita.<br>✓ Fácil de descargar e instalar.<br>✓ Corre en sistemas operativos Unix, Win32, BeOS, OS/2 y MacOS X.<br>✓ Fácil de aprender y utilizar.<br>✓ Amplia documentación. | ✓ Es *open source.*<br>✓ Es gratuita.<br>✓ Bastante rápida para realizar operaciones en comparación con otras.<br>✓ Permite el manejo de múltiples ramas.<br>✓ Maneja un sistema distribuido, por lo que cada usuario tiene una copia local.<br>✓ Corre en sistemas operativos Linux, Windows, Mac OS X y Solaris.<br>✓ Alto soporte para Linux.<br>✓ Integración con múltiples herramientas que brindan repositorios en la nube.<br>✓ Diversidad de herramientas de terceros que le proporcionan interfaz.<br>✓ Amplia ayuda de la comunidad y documentación. | ✓ Es open source.<br>✓ Es gratuita.<br>✓ Fácil de aprender y utilizar.<br>✓ Amplia documentación y de buena calidad.<br>✓ Maneja un sistema distribuido.<br>✓ Corre en sistemas operativos GNU/Linux, Windows, Mac OS X y Solaris. |
| **Desventajas** | ✗ Es lenta comparada con otras herramientas.<br>✗ Está basada en CVS por lo que aún posee algunos *bugs* provenientes de su origen.<br>✗ No maneja el concepto de ramas y etiquetas, el mismo usuario debe definir estos objetos.<br>✗ Maneja una arquitectura cliente-servidor, por lo que necesita un servidor para el repositorio y a donde los | ✗ La curva de aprendizaje es alta ya que posee mayor cantidad de comandos y un flujo muy diferente al de otras herramientas. | ✗ No permite realizar *merge* entre dos ramas padre.<br>✗ Regresar a una versión anterior a la última a la que se le hizo *commit* conlleva bastante trabajo.<br>✗ Eliminar ramas conlleva un trabajo complicado. |

Continuación de la tabla II.

| | | | |
|---|---|---|---|
| | usuarios puedan conectarse. | | |

Fuente: elaboración propia.

### 4.3.4. Evaluación y selección de una herramienta

Para la selección de las 3 herramientas propuestas fue de gran importancia que estas fuesen gratuitas y que soportaran múltiples plataformas, es por ello que se observa esta similitud entre ellas. Subversion es una herramienta simple, fácil de utilizar y con una curva de aprendizaje baja. Sin embargo debido a su relación con CVS posee algunas deficiencias y al tener una arquitectura cliente-servidor es necesario contar con más recursos para su implementación, convirtiéndolo en un sistema lento. Por otro lado están Git y Mercurial, que al ser sistemas distribuidos no requieren de un servidor dedicado y distribuyen la carga y procesamiento entre los miembros del equipo, volviéndolos sistemas veloces pero un tanto desordenados, ya que cada persona puede tener una versión diferente del proyecto en cierto momento.Por su parte, Mercurial es un sistema potente pero que mantiene la simplicidad, lo que brinda un aprendizaje fácil y rápido. En cambio, Git posee una curva de aprendizaje más alta pero que a largo plazo brinda la posibilidad de realizar más acciones que con otros controladores de versiones, ya que es una herramienta aún más potente. Además diversas herramientas de integración continua poseen compatibilidad con GitHub que aloja repositorios de Git.

Con base en la información brindada y el análisis realizado, Subversion puede ser fácil de utilizar pero requiere de más recursos para funcionar bien.

Mercurial y Git brindan la facilidad de trabajar desde cualquier lugar y en cualquier momento, ya que cada miembro del equipo posee una copia propia, un factor importante para los miembros de Edulibre que no se encuentran programando juntos o en el mismo lugar todo el tiempo. Mercuarial brinda una curva de aprendizaje baja, lo que permitiría al equipo de Edulibre comprender rápidamente el uso del control de versiones. Sin embargo, Git posee mayor compatibilidad con otras herramientas necesarias para lograr el objetivo de este proyecto y, a pesar de que la curva de aprendizaje es alta, los beneficios que se obtienen en el tiempo también son grandes.

## 4.4. Manejo de integración continua

El manejo de integración continua consiste en que los miembros del equipo de trabajo y demás involucrados apliquen un conjunto de prácticas definidas en el proceso de implementación de integración continua, paulatinamente, para ir logrando una mejor integración continua que proveerá pequeñas funcionalidades con mayor frecuencia, en tiempos de desarrollo cortos para que estén disponibles para instalación en producción.

### 4.4.1. Definición del proceso de integración continua

Con base en las prácticas identificadas en el proceso de implementación de integración continua, se definen los pasos que deberán realizar los miembros del equipo de trabajo para alcanzar la implementación de integración continua en sus desarrollos de sistema operativo EdulibreOS y el escritorio Innova. Estos pasos son:

Primer paso, la definición de cada proyecto debe tener poco alcance para que el proyecto pueda ser desarrollado en un tiempo no mayor a dos semanas.

Si el proyecto tiene muchos requerimientos se deberá dividir en los proyectos que sean necesarios para que cada proyecto tenga un tiempo de desarrollo que no exceda las dos semanas.

Segundo paso, cada miembro del equipo de trabajo que esté realizando un desarrollo o corrección deberá realizar *commit* con frecuencia, como mínimo una vez por día, para evitar que la integración con el desarrollo de los demás sea complicada y lleve mucho tiempo realizar la integración.

El miembro encargado del desarrollo del proyecto deberá realizar pruebas unitarias y funcionales en paralelo con su desarrollo, para validar su correcto funcionamiento y así proporcionar una funcionalidad estable, lista para realizarle pruebas funcionales y de integración, realizadas por los miembros encargados de la ejecución de pruebas.

Tercer paso, cuando el desarrollo esté finalizado se deberá trasladar a un ambiente de pruebas, en donde los encargados en realizar las pruebas del proyecto deberán aplicar las pruebas listadas en su *script*, validando distintos escenarios y valores. Si el proyecto tuviera errores o correcciones, se deberán notificar al miembro del equipo de trabajo encargado en el desarrollo, el error y los valores del escenario en el que ocurrió para que puedan ser replicados y corregidos. Si al proyecto se le deben realizar correcciones, el miembro encargado del desarrollo deberá priorizar la corrección de errores del proyecto y realizar los pasos correspondientes establecidos en la definición del proceso de control de versiones.

Cuarto paso, cuando el proceso haya finalizado las pruebas satisfactoriamente se deberá realizar los pasos correspondientes definidos en el

proceso del control de versiones y trasladar el proyecto a un estado de disponibilidad para ser desplegado en producción cuando sea requerido.

### 4.4.2. Herramientas propuestas

Para el manejo de la integración continua se plantean en esta sección tres herramientas de las cuales se mencionan sus distintas características. Al final, a partir de estas herramientas se tomará la propuesta a implementar.

- Jenkins: herramienta gratuita y *open source* que sirve para el manejo del proceso de integración continua y realizar pruebas, pero que además puede ser utilizada para el manejo del proceso de despliegue continuo. Posee una amplia integración con distintos *plugins*, lo que le brinda extensibilidad. También soporta distintos sistemas de control de versiones. Permite la compilación de proyectos que manejan ciertos lenguajes y línea de comandos. Posee una interfaz *web* a través de la cual se realizan todas las configuraciones necesarias para el manejo de los distintos procesos. Es bastante popular en el medio, por ser una herramienta liviana pero que cuenta con todas las características necesarias para llevar a cabo los procesos de integración continua y despliegue, ofreciendo un sistema robusto y sin costo.

- TeamCity: herramienta para el manejo de construcción de proyectos (*build*, en inglés) e integración continua y despliegue continuo. Es una herramienta fácil de instalar y utilizar gracias a su interfaz, personalizable e intuitiva para los usuarios, hecha para guiarlos a lo largo del proceso de configuración y utilización de la herramienta. En su página oficial se encuentra una amplia documentación y *demos* que guían al usuario en su uso. Esta herramienta permite agregar distintas dependencias, es extensible y soporta integración con distintos IDEs y controladores de

versiones. En cuanto a su costo, esta herramienta maneja un modelo de licenciamiento, por lo que es pagada. Sin embargo, cuenta con una versión gratuita que posee límite para 20 configuraciones y 3 agentes para construir proyectos. Por otro lado, también ofrece licencias gratuitas para proyectos *open source* que deben cumplir con ciertas características, como tener al menos 3 meses de funcionamiento y que sean proyectos que se mantienen activos. Es necesario llenar un formulario para aplicar a la licencia, la cual debe ser renovada cada año.

- GitLab CI: herramienta *open source* que permite realizar pruebas, construir código y desplegarlo, como parte del proceso de integración continua. Soporta múltiples plataformas y lenguajes, incluyendo línea de comandos para correr *scripts*. Una de sus características más importantes es que ofrece paralelismo para realizar las construcciones de código. Se puede repartir entre varios servidores la construcción de código, brindándole también escalabilidad automática para agregar o quitar máquinas virtuales, según sea necesario en algún punto en el tiempo. Brinda integración con Docker y varios controladores de versiones, especialmente con GitLab. Hay dos formas de obtener esta herramienta, una de forma gratuita en su *community edition* y su versión pagada que es la Enterprise edition.

### 4.4.3. Cuadro comparativo de herramientas

Con base en la investigación realizada y la información presentada se muestra el siguiente cuadro comparativo:

Tabla III. **Cuadro comparativo de herramientas de integración continua**

| | Jenkins | TeamCity | GitLab CI |
|---|---|---|---|
| **Ventajas** | ✓ Es *open source.*<br>✓ Es gratuita.<br>✓ Corre en sistemas operativos Unix-like, Windows y Mac OS X.<br>✓ Soporta integración con distintos sistemas de control de versiones.<br>✓ Permite realizar *builds* a través de línea de comandos.<br>✓ Extensible.<br>✓ Liviana<br>✓ Tiene interfaz *web.* | ✓ Corre en sistemas Linux, Windows y Mac OS X.<br>✓ Soporta integración con distintos sistemas de control de versiones.<br>✓ Permite realizar *builds* a través de línea de comandos.<br>✓ Interfaz personalizable e intuitiva.<br>✓ Fácil de utilizar y configurar.<br>✓ Extensible.<br>✓ Tiene una versión gratuita.<br>✓ Ofrece licencias gratuitas para proyectos *open source.* | ✓ Es *open source.*<br>✓ Corre en sistemas Linux.<br>✓ Permite realizar *builds* a través de línea de comandos.<br>✓ Permite realizar múltiples *builds* en paralelo.<br>✓ Soporta integración con distintos sistemas de control de versiones.<br>✓ Escalable.<br>✓ Tiene una *community edition.* |
| **Desventajas** | × No es tan intuitiva comparada con otras herramientas, por lo que al principio es difícil de utilizar. | × La versión gratuita es bastante limitada.<br>× La versión pagada maneja altos costos.<br>× Hay que aplicar para obtener la licencia gratuita y renovarla cada año.<br>× Mayor soporte para Windows. | × Mejor integración y soporte con GitLab. |

Fuente: elaboración propia.

### 4.4.4. Evaluación y selección de una herramienta

Para la selección de las 3 herramientas propuestas fueron fundamentales aspectos como que fuesen *open source*, de preferencia gratuitas, que tuviesen integración con los sistemas de control de versiones propuestos en la sección anterior y que manejasen línea de comandos. El último aspecto fue uno de los más importantes, ya que la asociación Edulibre trabaja con el lenguaje Gambas, para el cual no existen herramientas específicas, por lo que para poder manejar este lenguaje es necesario realizar las construcciones del proyecto a través de

*scripts.* Existe diversidad de herramientas para el manejo de integración continua pero que se especializan en ciertos lenguajes, por lo que fueron descartadas.

Jenkins es una de las herramientas de código abierto más populares en la actualidad, puede ser debido a que es gratuita y a que es una herramienta bastante completa que le permite a las empresas grandes y pequeñas realizar las mismas actividades y adaptarse a sus necesidades. Con facilidad se puede integrar a esta un sistema de control de versiones si existe algún *plugin* para el mismo. Por su parte, TeamCity es una herramienta con una interfaz poderosa, muy intuitiva, que brinda facilidad de uso y aprendizaje al usuario. Por defecto posee integración con sistemas de control de versiones. Una de sus grandes desventajas es que se maneja por licenciamiento y aunque posee una versión gratuita es muy limitada. Ofrece también la posibilidad de obtener una licencia gratuita, al ser un proyecto de código abierto, pero hay que aplicar para obtenerla y se debe renovar. Realmente, TeamCity se orienta más a empresas grandes, con gran capital y que poseen sistemas que cambian constantemente. Por último, GitLab CI es otra herramienta muy intuitiva y con una interfaz agradable. Permite integrar con repositorios de control de versiones como GitHub, que se encuentra en la nube, o brindando la url del repositorio en donde sea que se encuentre. Esta herramienta cuenta con dos ediciones, una de la comunidad y otra que es Enterprise, por lo que es pagada. Realmente la versión comunitaria posee las características suficientes para crear un sistema de integración continua aceptable. GitLab CI aún se encuentra agregando y mejorando su sistema debido a que es más joven que los otros mencionados aquí.

Con base en la información brindada y el análisis realizado se puede concluir que TeamCity es una herramienta potente pero sobrepasa las

necesidades de Edulibre, por lo que se desaprovecharían muchas características, y necesita inyección de capital. GitLab CI y Jenkins poseen las características necesarias para manejar los proyectos de Edulibre y aunque existe una edición comunitaria que es gratuita de GitLab CI, Jenkins ofrece todo el paquete con toda la funcionalidad de forma gratuita, con amplia ayuda de la comunidad y documentación gracias a su popularidad. Las personas en general prefieren y utilizan Jenkins, ya que es una herramienta que cumple con su funcionalidad de manera efectiva y, al ser Edulibre una asociación pequeña, es una herramienta ideal.

## 4.5. Manejo de pruebas

El manejo de pruebas consiste en prácticas que deberán ser implementadas en la ejecución de pruebas de los desarrollos del sistema operativo EdulibreOS y del escritorio Innova, que permiten detectar errores rápidamente evaluando distintos escenarios y valores, que al ser corregidos proporcionan desarrollos de calidad.

### 4.5.1. Definición del proceso de pruebas

La definición del proceso de pruebas se establecerá dependiendo del ambiente en que se encuentre el proyecto que agregará funcionalidad sobre el sistema operativo EdulibreOS y el escritorio Innova. Para EdulibreOS se establecerán dos ambientes para la ejecución de pruebas: el ambiente de desarrollo y el ambiente de pruebas.

Las pruebas en el ambiente de desarrollo deberán ser realizadas por el miembro del equipo de trabajo encargado del desarrollo del proyecto. Esta persona ejecutará dos tipos de pruebas: pruebas unitarias y pruebas

funcionales. Las pruebas unitarias evaluarán el funcionamiento a nivel de código del desarrollo y las pruebas funcionales evaluarán el funcionamiento adecuado a la característica en desarrollo.

Las pruebas en el ambiente de pruebas deberán ser realizadas por el ejecutor de pruebas, quien deberá crear un plan de pruebas. El plan contendrá los escenarios y valores que se evaluarán para validar el correcto funcionamiento del proyecto desarrollado y deberá incluir pruebas funcionales y pruebas de sistema.

Si existieran errores en las pruebas realizadas, el ejecutor de pruebas deberá indicar el escenario y los valores utilizados que provocaron el error, para ser reportados al encargado del desarrollo para su corrección. Cada vez que el proyecto regrese a pruebas, después de realizarle correcciones, deberá ejecutarse el plan de pruebas nuevamente, para validar todos los escenarios y valores y así validar también su correcto funcionamiento.

- Unitarias

Las pruebas unitarias serán ejecutadas por la persona que esté realizando el desarrollo. Es decir que cada miembro del equipo que implemente código debe encargarse de realizar las pruebas unitarias sobre dicho código. Las unidades a probar son cada método o función implementada y por cada escenario por el que se vaya a utilizar. Es decir que si se implementa un método/función debe realizarse una prueba unitaria y si dicho método/función es utilizado en distintos casos, por cada caso deberá realizarse una prueba unitaria.

Las pruebas unitarias se encuentran dentro del proyecto como una clase más, por lo que estarán contenidas en el repositorio. Cada vez que se necesite agregar una prueba bastará con agregar un método a la clase de pruebas. Se deberá presentar ante el coordinador de desarrollo el resultado de las pruebas en donde se valida que el proyecto pasó todas las pruebas.

- Funcionales

El coordinador de pruebas debe reunirse con el desarrollador para crear un plan de pruebas a seguir para probar la funcionalidad. Al final, el coordinador asigna al responsable, que por lo general será la persona que esté realizando el desarrollo, pero también podría ser el ejecutor de pruebas. Las pruebas funcionales serán realizadas por cada nueva funcionalidad que sea implementada. Un método/función por si solo puede no tener sentido, pero al invocar varios métodos se puede realizar una tarea con un sentido para el cliente, lo que se convierte en una funcionalidad.

Es decir que, si por ejemplo, para la pantalla de un juego se han creado dos nuevos botones, en donde uno permite guardar la información del jugador actual y el otro actualizar la información de un jugador, se correrán dos pruebas, una por cada botón, ya que cada uno realiza una función diferente y tiene sentido para el usuario. Cada botón seguramente realizará una serie de acciones que le permitirán realizar la acción para la que fue programado. A través de estas pruebas se podrá evaluar el correcto funcionamiento de nuevas características añadidas a un módulo.

Las pruebas funcionales también pueden encontrarse dentro del proyecto, en una clase como las pruebas unitarias. Sin embargo, si existen pruebas en donde se necesite de la intervención del usuario, como por ejemplo presionar

un botón para ejecutar alguna otra acción como parte de un proceso, será necesario utilizar otra herramienta en donde se debe crear un archivo fuera del proyecto, y que pueda ser leído y ejecutado por la otra herramienta, o bien, contar con el apoyo de colaboradores dispuestos a realizar las pruebas manualmente y notificar sobre cualquier inconveniente.

De igual manera que con las pruebas unitarias, el desarrollador o la persona que sea asignada para realizar la prueba debe presentar al coordinador de pruebas la evidencia de que las pruebas fueron exitosas, para notificar también al coordinador de desarrollo.

- Sistema

Las pruebas de sistema buscan evaluar el efecto de implementar una funcionalidad en un sistema, es decir, cómo esta puede afectar a las demás funcionalidades existentes. Su objetivo es verificar la correcta integración de una funcionalidad al sistema, ya que estas pueden ser objetos nuevos, mejoras, correcciones de errores, entre otros.

Para llevar a cabo estas pruebas es necesario que el coordinador de pruebas se reúna con el ejecutor de pruebas e incluso podría ser hasta con el desarrollador, para crear el plan de pruebas a ejecutar, especificando los posibles escenarios y valores según el caso, sin embargo, en este punto se pueden también pasar nuevamente las pruebas unitarias y funcionales que ya existan para cada proyecto, asociado a las nuevas funcionalidades, y así validar que todo siga funcionando bien.

Al final, estas pruebas serán realizadas por el ejecutor de pruebas, o bien el coordinador de pruebas puede asignar esta tarea a los desarrolladores que

crearon funcionalidades relacionadas, para que generen nuevamente las pruebas a partir de los archivos ya existentes. De igual forma, en algún punto se puede contar con la ayuda de los colaboradores o incluso con el usuario final para detectar cualquier otro problema.

### 4.5.2. Automatización de pruebas

La automatización de pruebas consiste en optimizar el proceso de pruebas que permita a los ejecutores de pruebas evaluar estas desde su *script* de una forma más fácil, más rápida y más simple, con el fin de reducir el tiempo en la detección de errores y evaluar más escenarios. La automatización del proceso de pruebas se aplicará utilizando herramientas que permitan ejecutar las pruebas de una forma más fácil y rápida, indicando las pruebas que se deben evaluar y que estas se ejecuten la cantidad de veces necesarias, de acuerdo a las correcciones que se le realicen al desarrollo para validar que las correcciones no afecten escenarios que se habían evaluado correctamente.

### 4.5.3. Herramientas propuestas

En esta sección se presentan tres herramientas para la implementación de pruebas y su automatización. Las herramientas planteadas poseen diferentes características que les permiten realizar distintos tipos de pruebas sobre proyectos, pero que brindan funcionalidades útiles para realizar pruebas sobre los proyectos desarrollados por Edulibre.

- Eggplant Functional: herramienta que permite realizar pruebas de funcionalidad sobre entornos gráficos. El usuario debe escribir un archivo en donde especifica la forma en que se realiza la prueba y los pasos a seguir. Esta herramienta toma ese archivo y a través de un análisis de

imágenes realiza la prueba. Debido a la forma en que funciona esta herramienta, las pruebas pueden realizarse sobre distintos dispositivos, sin importar la plataforma, lenguaje o sistema operativo. Posee integración con distintas herramientas de integración continua. Es una herramienta que se adquiere bajo licencia, es decir, es pagada.

- Linux Desktop Testing Project (ldtp): herramienta que, al igual que la anterior, permite realizar pruebas de funcionalidad y automatizarlas. Es de código abierto y utiliza las librerías de accesibilidad para trabajar sobre interfaces gráficas. Permite realizar pruebas en distintos ambientes y sin importar el sistema operativo, ya que posee versiones para los mismos, e incluso de forma remota. Se pueden escribir archivos de texto, en los cuales se especifican las acciones a realizar durante la prueba, a través de las distintas convenciones que posee, incluyendo aquellas para acceder a los objetos de la interfaz. Maneja distintos lenguajes de programación, incluyendo la línea de comandos.

- Componente para Gambas (UnitForm): Gambas es un entorno y lenguaje de programación de código abierto que permite realizar distintos tipos de desarrollos. Es fácil de aprender y utilizar, ya que está basado en el lenguaje Basic. Actualmente es utilizado por la asociación Edulibre para desarrollar sus proyectos. Hoy en día no existen herramientas propias o que posean integración con Gambas para la ejecución de pruebas unitarias, sin embargo, existen componentes que permiten realizar dichas pruebas y que son brindados por la comunidad de usuarios. El componente más popular es UnitForm, el cual se descarga y agrega al código del proyecto como una librería más. Una vez agregado se crean las clases necesarias, en donde se define el código de las pruebas a

realizar y se crea la instancia de la clase principal, la cual corre las pruebas y muestra los resultados de las mismas.

### 4.5.4. Cuadro comparativo de herramientas

Con base en la investigación realizada y la información presentada se muestra el siguiente cuadro comparativo:

Tabla IV. **Cuadro comparativo de herramientas de pruebas**

| | **Eggplant Functional** | **Linux Desktop Testing Project (LDTP)** | **Componente para Gambas (UnitForm)** |
|---|---|---|---|
| **Ventajas** | ✓ Permite realizar pruebas funcionales.<br>✓ Realiza pruebas sobre entornos gráficos.<br>✓ Se maneja a través de un archivo.<br>✓ Es independiente del lenguaje de programación.<br>✓ Permite automatización de pruebas. | ✓ Es *open source.*<br>✓ Posee versiones para Linux, Windows y Mac OS X.<br>✓ Permite realizar pruebas funcionales.<br>✓ Realiza pruebas sobre entornos gráficos.<br>✓ Se maneja a través de un archivo.<br>✓ Permite automatización de pruebas.<br>✓ Permite realizar pruebas de forma remota. | ✓ Es *open source.*<br>✓ Integración completa con Gambas.<br>✓ Fácil de integrar.<br>✓ Fácil de utilizar. |
| **Desventajas** | × Es pagada. | × La documentación es escasa. | × No existe una documentación específica, ya que es un desarrollo de la comunidad. |

Fuente: elaboración propia.

### 4.5.5. Evaluación y selección de una herramienta

Uno de los aspectos más relevantes en cuanto a la selección de las herramientas propuestas fue el hecho de que la asociación Edulibre trabaja con el lenguaje Gambas. Además, con base en los tipos de desarrollo que maneja Edulibre, se buscó que las herramientas permitieran realizar pruebas unitarias y/o funcionales a través de la interfaz de usuario. Hoy en día no existen herramientas que posean entre sus lenguajes a Gambas, y por ende no poseen integración directa con el mismo.

Como se mencionó en esta sección, existen distintos tipos de pruebas que pueden realizarse sobre un proyecto y varían en cuanto al tipo de proyecto que se está desarrollando y las características que maneja el mismo. Generalmente, al realizar pruebas sobre un proyecto se busca probar su código fuente, que es la parte más crítica y que sufre la mayor cantidad de modificaciones, aunque las pruebas pueden realizarse sobre distintas partes del proyecto para medir sus distintas funcionalidades.

En el área de pruebas unitarias la mejor opción para Edulibre es utilizar los componentes de Gambas, desarrollados por miembros de la comunidad. Estos componentes son fáciles de integrar y utilizar, además brindan información certera y de una manera entendible, para que el usuario pueda detectar rápidamente los errores y corregirlos.

Adicionalmente, se puede integrar el uso de LDTP, para realizar pruebas sobre las interfaces desarrolladas por Edulibre en su escritorio. Esta herramienta tiene compatibilidad completa con Linux, facilitando su instalación. Además, la creación de sus *scripts* es bastante intuitiva, basándose en las convenciones que ya posee esta herramienta y que están bien documentadas.

Por otro lado, Eggplant es una herramienta similar a LDTP, pero que se consigue bajo licencia.

## 4.6. Manejo de configuración

El manejo de la configuración es llevar un control de la configuración que debe controlarse con el sistema operativo EdulibreOS y el escritorio Innova. Consiste en identificar y documentar la información de los sistemas como dos productos, identificando los programas que contienen, la versión con la que fueron construidos y las características de cada sistema. Este control de configuración disminuirá los errores de integración y adaptación con nuevos desarrollos, sin afectar los programas y configuraciones del sistema operativo EdulibreOS o el escritorio Innova. Además, esto permite identificar las partes que saldrían involucradas en algún cambio de *framework*, herramienta de trabajo, con el fin de tener el control de cambios y si el cambio tuviera un impacto, que este sea identificado y mitigado.

### 4.6.1. Definición del proceso de configuración

El proceso de configuración consiste en llevar el control de las configuraciones y características de los sistemas EdulibreOS e Innova, así como las herramientas utilizadas para su desarrollo, *framework*, direcciones ip de los servidores, repositorio y versión utilizada.

La primera configuración que se debe administrar es el control de proyectos. Cada desarrollo que se realice sobre el sistema operativo EdulibreOS o el escritorio Innova se manejará como proyecto. Se deberá crear un *branch* por cada proyecto, obteniéndolo del *branch* principal de desarrollo, e

identificándolo con un identificador de proyecto. Este identificador se deberá guardar en el control de la configuración.

En el control de configuración de cada proyecto, además de un identificador, deberá incluirse una breve descripción de la funcionalidad que proporcionará. También deberá incluirse información del *framework* que se utiliza, la versión de la herramienta con que se desarrolla, dirección ip, puerto, archivo de configuración y las librerías que utiliza para su funcionamiento, según aplique.

La segunda configuración que se deberá administrar es el control de herramientas. Se deberán identificar y documentar las herramientas, *framework* y librerías utilizadas, especificando las versiones utilizadas para que, al momento de actualizar o cambiar una herramienta, *framework* o librería, tener identificado el impacto que provocaría para poder mitigarlo.

La tercera configuración que se deberá administrar es el control de direcciones ip y puertos. Se deberán identificar y documentar las direcciones ip y puertos utilizados, incluyendo información de las aplicaciones que acceden o se ejecutan sobre las mismas.

La cuarta configuración que se deberá administrar es el control de ambientes. Se deberá identificar la versión del *release* instalado en el ambiente, las configuraciones de ip y puertos, *framework* y librerías requeridas para el sistema.

## 4.7. Puntos de control

En los puntos de control se establecerán y se especificarán condiciones e indicadores que deberán ocurrir en el proceso de implementación de integración continua y administración al cambio en el desarrollo del sistema operativo EdulibreOS y del escritorio Innova, así como la acción que se deberá realizar cuando ocurra o se cumplan dichas condiciones.

### 4.7.1. Definición de puntos de control

Los puntos de control ayudarán a Edulibre a integrar y relacionar las prácticas de los diferentes procesos especificados en la integración continua, especificando las condiciones que deben ocurrir y la acción que se deberá realizar para integrar los procesos de integración continua al desarrollo del sistema operativo EdulibreOS y el escritorio Innova, especificando los siguientes puntos de control:

- Punto 1. Definición de proyectos: es importante que en la especificación de sus requerimientos se considere que el tiempo de desarrollo sea como máximo de dos semanas. Si el tiempo de desarrollo supera las dos semanas el proyecto se deberá subdividir en fases, especificando los requerimientos que se deberán desarrollar en un tiempo máximo de dos semanas por cada fase. Estas fases se manejarán como un proyecto y el desarrollo de las siguientes fases se iniciará si y solo si el desarrollo de la fase anterior se encuentra certificado y listo para lanzar a producción.

- Punto 2. Inicio de desarrollo del proyecto: se deberá crear un *branch* por cada proyecto del que se inicie un desarrollo, identificando el proyecto con un identificador único y etiquetándolo en el *branch* creado para el

proyecto. El miembro del equipo de trabajo que realice el desarrollo deberá realizar sus *commit* al menos una vez al día, agregando los avances de su desarrollo. Configurando la herramienta de integración continua se podrá notificar a los involucrados por correo electrónico los *commit* realizados en un *branch* en específico, para estar informados sobre los avances del proyecto relacionado.

- Punto 3. Compromiso con un commit diario: es importante que cada miembro del equipo de trabajo que esté realizando un desarrollo realice un *commit* de sus cambios al menos una vez por día en el *branch* del proyecto, además de realizar la configuración en la herramienta de integración continua para notificar los *commit* que se realicen al *branch* del proyecto, notificando al miembro del equipo de trabajo encargado en el desarrollo y al coordinador de desarrollo, para que esté enterado de los cambios realizados en el *branch* para validar su integración continua. Es importante que antes de realizar un *commit,* el miembro del equipo de trabajo encargado en el desarrollo del proyecto revise los cambios en el *branch* del proyecto, y por si este tuviera cambios los deberá fusionar realizando un *merge* en el *branch* local y el *branch* principal del proyecto.

- Punto 4. Ejecución de pruebas en ambiente de desarrollo: el miembro del equipo de trabajo encargado del desarrollo del proyecto deberá ejecutar pruebas unitarias por cada segmento de código que desarrolló y, al lograr una funcionalidad completa, ejecutar pruebas funcionales, realizándolas de forma manual o utilizando una herramienta de ejecución de pruebas recomendadas que facilitarán el proceso de pruebas.

- Punto 5. *Commit* al *branch* de pruebas: si es el primer *commit* de un proyecto al *branch* de pruebas, se deberá solicitar la carta de finalización

de desarrollo del proyecto para que se puedan iniciar la ejecución de pruebas, configurando la herramienta de integración continua para notificar a los involucrados en la instalación del proyecto en el ambiente de pruebas y ejecución de estas, para que se coordine la instalación y la ejecución de las mismas.

- Punto 6. *Script* de pruebas: el ejecutor de pruebas deberá crear un *script* que contenga todos los valores y escenarios que se evaluarán. Este *script* deberá ser validado por el coordinador de pruebas. Cuando el *script* esté validado, el ejecutor de pruebas realizará las identificadas, utilizando las herramientas de pruebas recomendadas que le faciliten el proceso de pruebas. Si se encontraran errores, el ejecutor de pruebas deberá indicar los valores y escenarios evaluados al miembro del equipo de trabajo encargado en el desarrollo para que realice las correcciones.

- Punto 7. *Commit* al *branch* de *release*: el *commit* al *branch* de *release* se realizará si y solo si la ejecución de pruebas es satisfactoria, siendo validada con la carta de certificación de pruebas satisfactorias. Se realizará un *commit* del *branch* de pruebas del proyecto al *branch* de *release* agregando una nueva funcionalidad disponible, configurando la herramienta de integración continua para que al realizar un *commit* en el *branch* de *release* se notifique a los involucrados del desarrollo, coordinador de pruebas y administrador de proyectos sobre la nueva funcionalidad disponible para ser instalada en producción.

- Punto 8. Lanzamientos nuevos: al momento de realizar un *commit* al *branch* de *release* el proyecto está listo para su lanzamiento a los usuarios finales. Estos *commit* se dan debido al compromiso del equipo de desarrollo y de pruebas en entregar continuamente los proyectos para

que estén disponibles para ser lanzados y para los usuarios. Dependerá del administrador de proyectos, con apoyo del coordinador de desarrollo, realizar un plan estratégico para liberar estos *release* y que estén disponibles para los usuarios finales.

## 4.8. Retroalimentación

El desarrollo del sistema operativo de EdulibreOS y del escritorio Innova deberá incluir el proceso de retroalimentación en la implementación de integración continua y administración al cambio, proceso que proporcionará información sobre la salud del proceso de desarrollo en el sistema Operativo EdulibreOS y del escritorio Innova, proporcionando información de los *build* de cada proyecto en desarrollo, así como el estado de las pruebas a todas las partes interesadas: administrador de proyectos, coordinador de desarrollo, desarrolladores y ejecutores de pruebas, para que estén informados del estado de los proyectos y sus avances de forma continua.

### 4.8.1. Puntos de control

Es el inicio del desarrollo del proyecto. En este punto de control, como parte del proceso de retroalimentación, deberá notificarse a los involucrados del proyecto la asignación e iniciación del desarrollo. Esta configuración se podrá realizar en la herramienta de integración continua.

*Commit* en los repositorios. Independientemente del *branch* al que se le realice el *commit* se deberá configurar en la herramienta de integración continua los *commit* que se le realicen a los *branch* de los proyectos. Si es un *branch* de desarrollo, se deberá notificar al encargado del desarrollo y al coordinador de desarrollo. Si es un *branch* de pruebas, se deberá notificar al coordinador de

desarrollo y al coordinador de pruebas, para que coordinen la ejecución de pruebas. Para el *branch* de *release*, se deberá notificar al administrador de proyectos y al coordinador de desarrollo, con el fin de que los involucrados estén enterados de una forma constante sobre el estado de los cambios de los desarrollos y nuevas funcionalidades.

En la ejecución de pruebas es importante que los involucrados e interesados estén informados del estado de ejecución de pruebas de una forma continua, y en un breve tiempo, para que se ejecuten satisfactoriamente y que los proyectos puedan ser lanzados constantemente. Es muy importante la comunicación entre el equipo de trabajo y los interesados, proporcionando información del estado y avances de los proyectos en desarrollo del sistema operativo EdulibreOS y el escritorio Innova, que como resultado busca poner a disposición de los usuarios nuevas funcionalidades de una forma continua.

### 4.8.2. Documentación

La documentación requerida para el desarrollo del sistema operativo EdulibreOS y del escritorio Innova deberá ser simple, clara, concisa y breve, para que no entorpezca el proceso de integración continua y administración al cambio. Debido a que los desarrollos no deberán exceder las dos semanas, la documentación no puede ser tan extensa, y se deberán crear los siguientes documentos:

- Documento de especificación de requerimiento: en este documento se deberán especificar las funcionalidades requeridas en un documento breve y claro que proporcione información al miembro del equipo de trabajo de lo que debe desarrollar.

- *Script* de pruebas: por cada proyecto se deberá crear un *script* de pruebas que contendrá los valores y escenarios que se evaluarán en la ejecución de pruebas.

- Cartas de finalización de desarrollo y cartas de finalización de pruebas satisfactorias: para llevar un control de los desarrollos finalizados y de la ejecución de pruebas satisfactorias de los distintos proyectos y los responsables.

# CONCLUSIONES

1. A partir de la ayuda proporcionada por la asociación Edulibre y del proceso de investigación, se logró realizar el presente trabajo, en el cual se describe el proceso de desarrollo que los miembros del equipo de Edulibre llevan a cabo para realizar sus proyectos, identificando cada fase y a las personas que forman parte del mismo, los entregables que se obtienen y la forma en que se realizan las pruebas dentro del equipo y con los usuarios finales.

2. Al realizar la investigación con los miembros del equipo de trabajo de Edulibre se pudo identificar la necesidad de un sistema de control de versiones que permita tener en un repositorio los proyectos que manejan, un sistema de integración continua que permita realizar las compilaciones de los proyectos cada vez que los miembros del equipo realicen cambios para verificar que no hayan problemas en la construcción, ya que la falta de un plan de pruebas a seguir en cada implementación y por ende la automatización de pruebas ayudaría a reducir tiempos de búsqueda de errores en el código.

3. Los puntos de control permiten establecer indicadores para la toma de acciones a ejecutar al momento de implementar la integración continua y administración del cambio en un proceso de desarrollo, por lo que, para el desarrollo del sistema operativo EdulibreOS y del escritorio Innova, se definieron 8 puntos de control a tomar en cuenta, en donde se establecieron las acciones a tomar considerando cada fase del proceso.

4. Uno de los mayores problemas que existe en el desarrollo de un proyecto es la falta de definición de roles, lo que puede generar confusión, caos e incluso conflictos en el clima laboral. Es por ello que en todo proyecto es de gran importancia definir claramente los roles de cada miembro del equipo, sus funciones y responsabilidades. Para el caso de Edulibre se identificaron 5 roles al momento de realizar la investigación, para los cuales en este documento se han definido las funciones y responsabilidades dentro del proceso de desarrollo. Además, se han agregado 3 roles más para la implementación del proceso de pruebas.

5. Un punto identificado como parte de las oportunidades de mejora fue la falta de un plan de pruebas dentro de la asociación Edulibre, por lo que es un proceso nuevo del cual no se tienen antecedentes, lo que no permite estimar la cantidad de tiempo que le toma a los miembros del equipo detectar errores en el proceso de desarrollo actual, ya que las pruebas son realizadas en tiempo de implementación por el desarrollador, sin un plan u orden específico, o hasta que el usuario final utiliza el producto.

6. El proceso para implementar la automatización de pruebas puede ser lento al inicio, ya que existe una brecha de aprendizaje que puede crecer conforme a la complejidad de cada herramienta y disposición de cada miembro del equipo de aprender a utilizarla, pero luego podrán observarse los resultados y la optimización de tiempo y esfuerzo requerido para probar los proyectos y detectar errores. Debido a la actual falta de un plan de pruebas, y a que no se pudo estimar el tiempo promedio para encontrar errores, no fue posible estimar directamente la cantidad de tiempo que puede reducirse al implementar la automatización de pruebas.

7. El proceso para implementar el control de configuración, por ser un proceso nuevo, puede tornarse complicado y lento al inicio debido al tiempo de aprendizaje y adaptación que requiere. Al ser un proceso nuevo no es posible estimar la reducción de problemas de configuraciones de servicios, puertos, accesos y versiones de las herramientas utilizadas.

8. Se logró identificar tres herramientas de integración continua, las cuales tenían como característica principal correr en sistemas Linux, permitir realizar *builds* a través de línea de comandos y ser gratuitas. De las tres herramientas se seleccionó Jenkins, por ser una herramienta madura que ofrece toda la funcionalidad sin ningún costo, contraria a las otras dos, que ofrecen funcionalidad limitada en sus versiones gratuitas o les falta madurez.

9. Se logró identificar tres herramientas para realizar pruebas, las cuales tenían como característica principal correr en sistemas Linux, ser independientes del lenguaje de programación o manejar Gambas y permitir la automatización de pruebas. De las tres herramientas se seleccionó el componente para Gambas, el cual sirve para realizar pruebas unitarias en este lenguaje, que es el que utilizan los miembros de Edulibre. Sin embargo, también se puede considerar, en caso de ser necesario, el uso de *Linux Desktop Testing Project*, para realizar pruebas funcionales, ya que es una herramienta creada para Linux y que permite probar entornos gráficos.

# RECOMENDACIONES

1. La definición de roles es una parte importante y un punto de partida para evitar conflictos en el ambiente laboral y así realizar los proyectos de forma ordenada. Es por ello que se han propuesto nuevos roles, adicionales a los ya identificados, que ayudarán a la implementación de integración continua y pruebas, y además se han definido una serie de funciones y responsabilidades para todos los roles. Por ello se recomienda cumplir con esta definición, para así facilitar la implementación de nuevos procesos, evitar confusión entre los miembros del equipo y facilitar el cumplimiento de cada fase del proceso de desarrollo.

2. Muchas veces en el desarrollo de un proyecto no se realizan pruebas, o no más que aquellas que el desarrollador realiza en algún momento para probar ciertas validaciones realizadas. Sin embargo, es importante probar todo el código de manera continua y ver como esto puede afectar no solo al desarrollo realizado sino a otros módulos. Todo esto ayudará a identificar de forma temprana errores o problemas y encontrar de manera más rápida la causa. Por ello se recomienda la implementación de pruebas unitarias, funcionales y del sistema, definidas en este documento a través de un plan de pruebas realizado por los roles correspondientes y ejecutado por los mismos.

3. La implementación del proceso de integración continua y pruebas es solo la parte inicial, ya que debe haber una retroalimentación continua a lo largo del ciclo de desarrollo. Esta retroalimentación permitirá que el

equipo siga creciendo, se definan detalles inconclusos e incluso se encuentren puntos de oportunidad y mejora. Además, ayudará a que el proceso no se quede estancado sino que continúe creciendo.

4. Como con toda nueva implementación, la resistencia al cambio puede ser un inconveniente debido a la complejidad y tiempo de aprendizaje que requiere, lo que se deberá superar en equipo. Implementar el proceso completo de integración continua y administración al cambio puede ser muy complejo, incluso más complicado en equipos de trabajo sin una metodología de desarrollo bien definida, por lo que se recomienda implementar el proceso de integración continua y administración al cambio por partes, para que no sea muy complejo y se cuente con el compromiso de los miembros del equipo para completarlo. Con esto se logrará una correcta implementación de integración continua y administración al cambio en el proceso de desarrollo actual del sistema operativo EdulibreOS y del escritorio Innova.

# BIBLIOGRAFÍA


1. Accurev. *Software configuration management best practices.* [en línea]. <http://resources.idgenterprise.com/original/AST-0062469_AccuRev-SCM-Best-Practices.pdf>. [Consulta: 20 de febrero de 2016].

2. Accurev. *Software configuration management best practices for continuous integration.* [en línea]. < http://www.elegosoft.com/files/Downloads/Paper/best-practices_continuous-integration-in-scm_en.pdf >. [Consulta: 20 de febrero de 2016].

3. BORRELL, Guillem. *El control de versiones.* [en línea]. <http://torroja.dmt.upm.es/media/files/cversiones>. [Consulta: 20 de febrero de 2016].

4. CERVANTES, Damián. *Propuesta de entorno de integración continua en el centro de informatización universitaria*. Ciudad de la Habana: 2010. 25 p.

5. CHACON, Scott; STRAUB, Ben. *Pro Git.* 2da ed. Estados Unidos: Apress, 2014. 584 p.

6. COLLINS-SUSSMAN, Ben; W. FITZPATRICK, Brian; PILATO, C. Michael. *Version control with subversion.* [en línea]. <http://svnbook.red-bean.com/en/1.2/svn-

book.html#svn.webdav.clients.fs-impl.linux>. [Consulta: 20 de febrero de 2016].

7. Edulibre. *Página oficial*. [en línea]. http://edulibre.net/. [Consulta: 20 de febrero de 2016].

8. Edulibre. *Página oficial*. [en línea]. http://edulibre.net/index.php/about-us/. [Consulta: 20 de febrero de 2016].

9. Edulibre. *Página oficial*. [en línea]. http://edulibre.net/index.php/objetivos/. [Consulta: 20 de febrero de 2016].

10. FOGEL, Karl. *Cómo llevar a buen puerto un proyecto de código libre.* [en línea]. <http://producingoss.com/es/vc.html>. [Consulta: 20 de febrero de 2016].

11. FOWLER, Martin. *Continuous integration*. [en línea]. http://www.martinfowler.com/articles/continuousIntegration.html [Consulta: 20 de febrero de 2016].

12. FRANCOIS, Fabien. *Symfony 1.4, la guía definitiva.* [en línea]. < http://librosweb.es/libro/symfony_1_4/capitulo_15/automatizacion_de_pruebas.html >. [Consulta: 20 de febrero de 2016].

13. GARCÍA, Ana. *Aprende a implantar integración continua desde cero (I): ¿Por qué integración continua?* [en línea]. http://www.javiergarzas.com/2014/08/implantar-integracion-continua.html. [Consulta: 20 de febrero de 2016].

14. GARCÍA OTERINO, Ana M. del Carmen. *¿Cómo enfoco las pruebas de forma ágil?* [en línea]. http://www.javiergarzas.com/2015/01/testing-agil.html. [Consulta: 20 de febrero de 2016].

15. GARZÁS, Javier. *¿Tardaríais mucho en pasar a producción un cambio en sólo una línea de código? Aprende entrega continua.* [en línea]. http://www.javiergarzas.com/2012/11/entrega-continua-continuous-delivery.html. [Consulta: 20 de febrero de 2016].

16. HAYES, Linda. *The automated testing handbook.* [en línea]. < http://www.softwaretestpro.com/itemassets/4772/automatedtesting handbook.pdf >. [Consulta: 20 de febrero de 2016].

17. HUMBLE, Jez; FARLEY, David. *Continuous delivery reliable software releases through build, test, and deployment automation.* Estados Unidos: Pearson Education Inc., 2011. 463 p.

18. IT-MENTOR. *Integración Continua* [en línea]. http://materias.fi.uba.ar/7548/IntegracionContinua.pdf. [Consulta: 20 de febrero de 2016].

19. JONASSEN, Anne. *Configuration management principles and practice.* Boston: Pearson Education Inc., 2003. 432 p.

20. La Hora. *Tecnología en escuelas e institutos*. [en línea]. http://lahora.gt/mineduc-solo-el-5-de-escuelas-e-institutos-cuenta-con-tecnologia/. [Consulta: 20 de febrero de 2016].

21. M. DUVALL, Paul; MATYAS, Steve; GLOVER, Andrew. *Continuous integration improving software quality and reducing risk.* Estados Unidos: Pearson Education Inc., 2007. 283 p.

22. O'SULLIVAN, Bryan. *Mercurial: the definitive guide* [en línea]. < http://hgbook.red-bean.com/read/ >. [Consulta: 20 de febrero de 2016].

23. SINK, Eric. *Version control by example.* 1ra ed. Estados Unidos: Pyrenean Gold Press, 2011. 210 p.

24. Software Program Manager Network. *Little book of configuration management.* Computers & Concepts Associates, 1998. 49 p.

25. WINGERD, Laura; SEIWALD, Christopher. *High-level best practices in software configuration management.* [en línea]. < https://www.perforce.com/sites/default/files/pdf/perforce-best-practices.pdf >. [Consulta: 20 de febrero de 2016].

26. WILSENACH, Rouan. *DevOpsCulture.* [en línea]. http://martinfowler.com/bliki/DevOpsCulture.html. [Consulta: 20 de febrero de 2016].

27. YEPES, Yeimy. *Aplicación de integración continua para el mejoramiento de la calidad de software en Sisfo Ltda.* Universidad Católica Popular del Risaralda: 2010. 35 p.

# APÉNDICES

Apéndice 1. **Propuesta de plan operativo para implementación de la administración del cambio e integración continua**

## 1.1. Organización

### 1.1.1. Visión

"Brindamos la oportunidad a las niñas y niños de Latino América de tener acceso a una educación de calidad a través de tecnologías de información, guiados por los principios de código abierto."[12]

### 1.1.2. Misión

"Desarrollamos soluciones de tecnología de información de código abierto y asesoramos en su uso en las escuelas de nivel primario, integrándolas a su práctica pedagógica."[13]

### 1.1.3. Objetivos

La asociación Edulibre define los siguientes objetivos:

- General

[12] EDULIBRE. http://edulibre.net/index.php/about-us/. Consulta 20 de febrero de 2016.
[13] Ibid.

Continuación del apéndice 1.

- “Promover el uso de un Sistema Operativo con programas y herramientas educativas totalmente libres por medio de EdulibreOS a los niños y estudiantes guatemaltecos (as) o de cualquier parte de Latinoamérica, mediante el aprendizaje tecnológico”[14].

- Específicos
  - “Crear una plataforma totalmente educativa.
  - Que los usuarios sean capaces de utilizar las herramientas educativas que se les ofrece.
  - Promover el aprendizaje de los niños o de los estudiantes.
  - Facilitar herramientas tecnológicas libres”[15].

### 1.1.4. Roles

Los roles definen la función de cada miembro del equipo de trabajo de Edulibre. Un miembro puede tener más de un rol de acuerdo a los miembros disponibles o experiencia que posean. A continuación se presentan los roles que poseen actualmente con una breve descripción de su función:

| ROL | Responsabilidades |
|---|---|
| Administrador de proyectos | ○ Administrar los requerimientos de los nuevos proyectos o mantenimientos del sistema operativo y escritorio, priorizando los requerimientos en conjunto con el coordinador de desarrollo.<br>○ Verificar el cumplimiento del desarrollo del proyecto de acuerdo a su planificación en conjunto con el coordinador |

[14] EDULIBRE. http://edulibre.net/index.php/objetivos/. Consulta 20 de febrero de 2016.
[15] Ibid.

Continuación del apéndice 1.

| | |
|---|---|
| | ○ del proyecto.<br>○ Coordinar el desarrollo y pruebas de los proyectos de acuerdo a la prioridad establecida, en conjunto con los coordinadores de desarrollo y pruebas.<br>○ Coordinar instalaciones y actualizaciones del sistema operativo en las escuelas. |
| Coordinador de desarrollo | ○ Administrar los proyectos de desarrollo y mantenimiento del sistema operativo y del escritorio, validando los tiempos de desarrollo y el cumplimiento de los requerimientos.<br>○ Asignar proyectos a los desarrolladores y darles seguimiento a los avances.<br>○ Priorizar los proyectos de desarrollo en conjunto con el administrador de proyectos.<br>○ Presentar al administrador de proyectos los avances y estados de los proyectos. |
| Desarrollador | ○ Realizar una breve planificación de tareas sobre el desarrollo a realizar, y validarlas con el coordinador de desarrollo.<br>○ Realizar el desarrollo del proyecto de acuerdo a la planificación aprobada.<br>○ Notificar sobre los avances al coordinador de desarrollo.<br>○ Realizar pruebas unitarias y funcionales. |
| Coordinador de pruebas | ○ Administrar la ejecución de pruebas a proyectos finalizados.<br>○ Asignar responsables de ejecución de pruebas.<br>○ Revisar el plan de pruebas en donde se especifican los escenarios a evaluar y pruebas a realizar. |
| Ejecutor de pruebas | ○ Crear el plan de pruebas para cada proyecto.<br>○ Validar el plan de pruebas con el coordinador de pruebas.<br>○ Ejecutar las pruebas e informar sobre los errores encontrados, especificando de forma clara el error y los valores con los que este se dio. |
| Usuario | ○ Utilizar el sistema e informar sobre cualquier falla/error. |

Fuente: elaboración propia.

Continuación del apéndice 1.

## 1.2. Objetivos operativos

1. Implementar control de versiones en el desarrollo del sistema operativo EdulibreOS y su escritorio Innova.

2. Implementar integración continua en el desarrollo del sistema operativo EdulibreOS y su escritorio Innova.

3. Implementar un plan de pruebas automatizadas en el desarrollo del sistema operativo EdulibreOS y su escritorio Innova.

## 1.3. Plan operativo

| Objetivos | Actividades | Responsable | Cuándo | Cómo |
|---|---|---|---|---|
| Implementar control de versiones | Registrarse en Bitbucket. | Coordinador de desarrollo. Desarrollador. | Cada vez que se incorpore una persona nueva al proyecto, por ser la primera vez, se registra a todos los involucrados. | En la página oficial de Bitbucket. |
| | Crear repositorios de los desarrollos en Bitbucket. | Coordinador de desarrollo. | Cada vez que se inicie un proyecto nuevo. (Actualmente solo se tiene el proyecto de Innova y EdulibreOS.) | En la página oficial de Bitbucket, utilizando su cuenta. |

Continuación del apéndice 1.

| | | | | |
|---|---|---|---|---|
| | Instalar y configurar Git. | Coordinador de desarrollo. Desarrollador. | Al realizar cambio de máquinas o al incorporarse alguien nuevo debe realizar este proceso. Por ser la primera vez todos deben realizar este paso. | Utilizando la línea de comandos. |
| | Agregar los proyectos de desarrollo a Git y a Bitbucket. | Coordinador de desarrollo. Desarrollador. | Cada vez que se inicie un proyecto nuevo. (Actualmente sólo se tiene el proyecto de Innova y EdulibreOS.) | Utilizando la línea de comandos y desde la página de Bitbucket. |
| | Crear los *branchs* para los proyectos. | Coordinador de desarrollo. | Cada vez que se inicie un proyecto nuevo. (Actualmente solo se tiene el proyecto de Innova y EdulibreOS.) | Utilizando la línea de comandos o el cliente de control de versiones. |
| | Agregar y confirmar los cambios al repositorio local. | Desarrollador. | Todos los días, cada vez que se agregue una nueva funcionalidad. | Utilizando la línea de comandos o el cliente de control de versiones. |
| | Actualizar los cambios al repositorio origen. | Desarrollador. | Todos los días, al finalizar el día. | Utilizando la línea de comandos o el cliente de control de versiones. |
| | Actualizar el repositorio local. | Desarrollador. | Todos los días, al iniciar el día. O al menos una vez por semana para evitar la mayor cantidad de conflictos. | Utilizando la línea de comandos o el cliente de control de versiones. |

Continuación del apéndice 1.

| | | | | |
|---|---|---|---|---|
| Implementar integración continua | Instalar y configurar Jenkins. | Coordinador de desarrollo. | Al realizar cambio de máquinas o al incorporarse alguien nuevo debe realizar este proceso. Por ser la primera vez todos deben realizar este paso. | Utilizando la línea de comandos. |
| | Configurar el *plugin* de Git en Jenkins. | Coordinador de desarrollo. | La primera vez que se accede a Jenkins luego de su instalación. | Accediendo a Jenkins a través de un navegador. |
| | Configurar el *plugin* de Bitbucket en Jenkins. | Coordinador de desarrollo. | La primera vez que se accede a Jenkins luego de su instalación. | Accediendo a Jenkins a través de un navegador. |
| | Configurar los *commit* de los proyectos para crear los *releases*. | Coordinador de desarrollo. | La primera vez que se accede a Jenkins luego de su instalación. | Accediendo a Jenkins a través de un navegador. |
| Implementar plan de pruebas automatizadas | Agregar componentes requeridos para las pruebas unitarias | Desarrollador. | La primera vez que se accede a Gambas luego de su instalación. Si ya está instalado, solo una vez. | Utilizando la herramienta Gambas. |
| | Crear un plan de pruebas con los distintos escenarios y valores. | Ejecutor de pruebas. Coordinador de pruebas. Desarrollador. | Cada vez que se finalice la creación de una nueva funcionalidad. | Realizando reuniones con el desarrollador y encargado de pruebas, en donde se cree un documento para establecer los lineamientos de la prueba. |

Fuente: elaboración propia.

Continuación del apéndice 1.

En el capítulo 2 puede encontrarse información más detallada de cómo implementar cada sistema, en las secciones guía.

Continuación del apéndice 1.

### 1.4. Cronograma

| Objetivos | Metas | Cronograma | | | |
|---|---|---|---|---|---|
| | | SEP | OCT | NOV | DIC |
| Implementar control de versiones. | Disciplinar a los miembros del equipo de desarrollo con las buenas prácticas sobre el control de versiones. | X | | | |
| Implementar integración continua | Automatizar los *releases* de nuevas funcionalidades en los distintos ambientes. | | X | X | |
| Implementar plan de pruebas automatizadas | Definir un procedimiento de pruebas por cada proyecto.<br>Automatizar el proceso de pruebas. | | | X | X |

Apéndice 2. **Manual para implementacion de administración continua e integración al cambio**

### 2.1. PASO 1: Implementación de control de versiones

Como primer paso en la implementación de administración del cambio e integración continua se deberá implementar un control de versiones para llevar el control de cambios en las versiones de los desarrollos.

Continuación del apéndice 2.

### 2.1.1. Configuración inicial

La configuración inicial del repositorio y control de versiones estará a cargo del coordinador de desarrollo. Esta configuración prepara el ambiente de control de versiones, por lo que deberá realizarse solamente una vez. A continuación se describen los pasos a seguir para aplicar la configuración inicial.

#### 2.1.1.1. Registrarse en Bitbucket

El coordinador de desarrollo deberá registrar su cuenta de correo electrónico en el sitio de Bitbucket. (Guía – Registrarse en Bitbucket).

#### 2.1.1.2. Crear repositorio

El coordinador de desarrollo deberá crear los repositorios EdulibreOS, Innova y *Release* (Guía – Crear repositorio), cada uno para un propósito específico:

- EdulibreOS: para almacenar las versiones de los cambios en el código fuente del sistema operativo EdulibreOS.

- Innova: para almacenar las versiones de los cambios en el código fuente del escritorio Innova.

Continuación del apéndice 2.

- *Release*: para almacenar las fuentes utilizadas de los distintos lanzamientos.

#### 2.1.1.3. Instalar Git

El coordinador de desarrollo deberá instalar Git en su computadora para inicializar los repositorios y crear los *branch* respectivos con el fin de crear el ambiente adecuado para el manejo de control de versiones. (Guía – Instalar Git) Continuación del apéndice 2.

#### 2.1.1.4. Crear repositorio local basado en repositorio origen

El coordinador de desarrollo deberá crear los repositorios localmente en su computadora clonando el repositorio origen (repositorio en Bitbucket). (Guía – Crear repositorio local clonando el repositorio origen).

#### 2.1.1.5. Agregar los archivos fuentes iniciales al repositorio local

El coordinador de desarrollo estará a cargo de agregar los directorios y archivos fuentes base en los repositorios locales EdulibreOS e Innova. Deberá utilizar la sintaxis adecuada de Git para agregar los directorios y archivos incluidos en cada repositorio y deberá confirmar los cambios. (Guía – Realizar cambios al repositorio local y confirmarlos).

Continuación del apéndice 2.

#### 2.1.1.6. Crear los *branch*

En los repositorios EdulibreOS e Innova el coordinador de desarrollo estará a cargo de crear los siguientes *branch*:

- Desarrollo: *branch* donde se aplicarán los cambios de los nuevos desarrollos.

- Correcciones: *branch* donde se aplicarán correcciones.

Estos *branch* se deberán crear por cada uno de los dos repositorios mencionados. (Guía – Crear *branch*).

#### 2.1.1.7. Actualizar repositorio origen

El coordinador de desarrollo deberá actualizar los repositorios origen (repositorios en Bitbucket) con los cambios realizados en los repositorios locales, para que el repositorio origen esté actualizado con las fuentes base y sus respectivos *branch*. (Guía – Actualizar repositorio origen).

Aplicando los pasos indicados se tendrá listo el ambiente de control de versiones inicial con las fuentes base de cada repositorio, así como de su trabajo.

### 2.1.2. Iniciación de desarrollo o corrección

El coordinador de desarrollo deberá crear un *branch* por cada desarrollo nuevo identificado adecuadamente. Las correcciones irán directamente sobre el *branch* de correcciones. A continuación se describen los pasos a aplicar.

#### 2.1.2.1. Crear *branch*

El coordinador de desarrollo deberá crear un *branch* por cada proyecto que inicie su desarrollo, basándose en el *branch* de desarrollo. El *branch* deberá estar identificado con el identificador del proyecto. (Guía – Crear *branch*).

#### 2.1.2.2. Actualizar repositorio origen

En el paso anterior la creación del *branch* quedó en el repositorio local, por lo que se deberá actualizar en el repositorio origen para que esté disponible para los miembros del equipo. (Guía – Actualizar repositorio origen).

### 2.1.3. Desarrollar o realizar correcciones

El miembro del equipo de trabajo deberá disponer de un ambiente adecuado para realizar el desarrollo de proyectos o correcciones. Para esto se deberán aplicar los siguientes pasos.

Continuación del apéndice 2.

#### 2.1.3.1. Registrarse en Bitbucket

El miembro del equipo de trabajo que interactuará con un repositorio origen deberá registrarse en la plataforma de Bitbucket. Este registro se realiza solamente una vez. (Guía – Registrarse en Bitbucket).

#### 2.1.3.2. Instalar Git

El miembro del equipo de trabajo que interactuará con un repositorio deberá instalar y configurar la herramienta de control de versiones Git en su computadora. (Guía – Instalar Git).

#### 2.1.3.3. Obtener repositorio de trabajo

El miembro del desarrollo deberá obtener el repositorio de trabajo, el cual dependerá del desarrollo o corrección sobre el cual se vaya a aplicar.

- Si el miembro de desarrollo no tiene localmente el repositorio sobre el cual realizará el desarrollo o corrección, deberá obtenerlo localmente, primero obteniendo el repositorio origen. (Guía – Crear repositorio local clonando repositorio origen).

- Si el miembro de desarrollo ya tiene localmente el desarrollo sobre el cual realizará el desarrollo o corrección, deberá actualizar el repositorio local por si al repositorio origen se le han aplicado cambios. (Guía – Actualizar repositorio local).

#### 2.1.3.4. Seleccionar *branch* asignado para el desarrollo o corrección

El miembro del equipo de trabajo que tenga asignado un desarrollo o corrección lo deberá hacer sobre el *branch* asignado, para lo cual tendrá que seleccionar el *branch* indicado. (Guía – Cambiar de *branch*).

#### 2.1.3.5. Realizar cambios al repositorio local y confirmarlos

El miembro del equipo de trabajo que esté realizando un desarrollo o una corrección deberá agregar sus cambios al repositorio local y confirmarlos para que los cambios queden registrados. Los *commit* de cambios se deberán realizar como mínimo una vez al día por cada miembro del equipo de trabajo. (Guía – Realizar cambios en el repositorio local y confirmarlos).

#### 2.1.3.6. Actualizar repositorio origen

El miembro del equipo de trabajo deberá actualizar los cambios realizados en su repositorio local al repositorio origen sobre el *branch* de trabajo como mínimo una vez por día. (Guía – Actualizar repositorio local).

#### 2.1.3.7. Integrar cambios al repositorio origen del *branch* principal

Continuación del apéndice 2.

El miembro del equipo de trabajo, en conjunto con el coordinador de desarrollo, deberá aplicar los cambios realizados de un desarrollo o corrección al *branch* principal de desarrollo o correcciones realizando los siguientes pasos:

- **Actualizar repositorio local**
  Se deberá actualizar el repositorio local para validar que no existan cambios en el repositorio origen. De existir cambios será necesario incluirlos. (Guía – Actualizar repositorio local).

- **Cambiar al *branch* principal**
  Se deberá cambiar al *branch* principal de desarrollo o correcciones, al que se le incluirán los cambios realizados según corresponda. (Guía – Cambiar *branch*).

- **Fusionar *branch***
  Se deberán fusionar los cambios del *branch* de desarrollo o correcciones, según sea el caso, al *branch* principal de desarrollo o corrección. (Guía – Fusionar *branch*).

- **Actualizar repositorio origen**
  Cuando la fusión de los *branch* se haya realizado correctamente, se deberán actualizar los cambios al repositorio origen. (Guía – Actualizar repositorio origen).

Con los pasos indicados se logrará tener un control de versiones adecuado para el desarrollo o corrección de errores sobre el sistema operativo EdulibreOS y el escritorio Innova.

Continuación del apéndice 2.

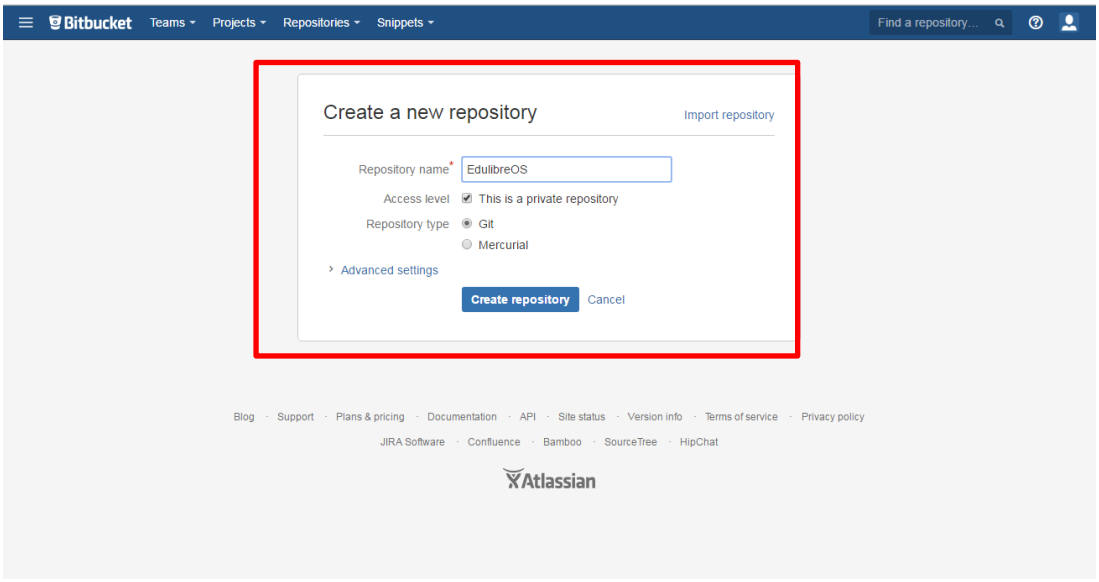
### 2.1.4. Guía

**Registrarse en Bitbucket**

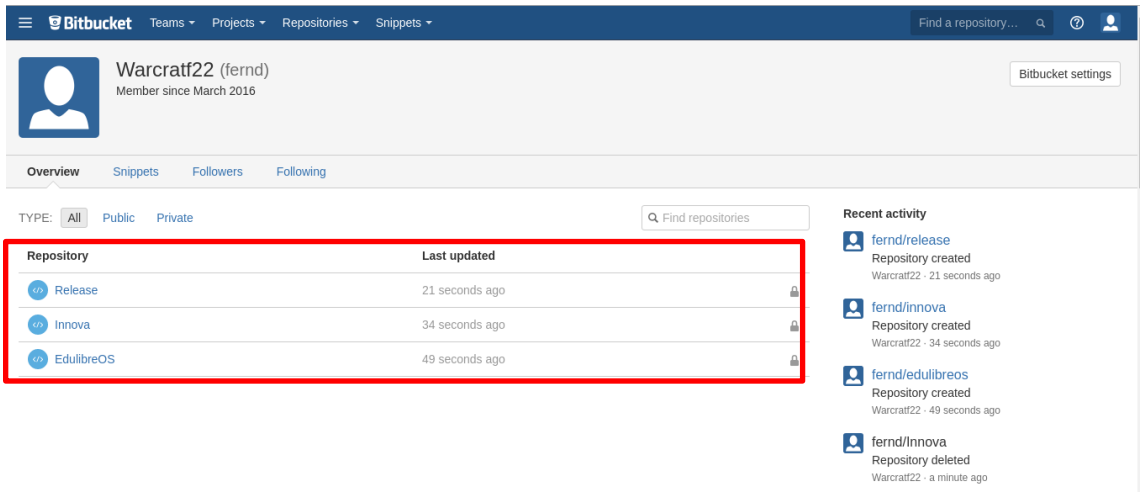
El miembro del equipo de trabajo que tenga interacción con algún repositorio deberá registrarse en https://bitbucket.org/.

**Crear repositorio**

Únicamente el coordinador de desarrollo podrá crear los repositorios utilizados para los desarrollos realizados para Edulibre, mostrando el siguiente ejemplo:



Continuación del apéndice 2.




**Instalar y configurar Git**

Se deberá instalar y configurar la herramienta de control de versiones Git en las computadoras de los miembros del equipo de trabajo que interactúen con los repositorios.

Continuación del apéndice 2.

Instalar Git ejecutando el comando "*sudo apt-get install git*":

```
juliofernando@Theroths:~$ sudo apt-get install git
```

Verificar correcta instalación de Git ejecutando el comando "*which git*", si es correcta indica la ruta de instalación de Git.

```
juliofernando@Theroths:~$ which git
/usr/bin/git
juliofernando@Theroths:~$
```

Configurar nombre del usuario ejecutando el comando "*git config --global user.name 'nombre_usuario'*".

```
juliofernando@Theroths:~$ git config --global user.manme "Julio Fernando"
juliofernando@Theroths:~$
```

Continuación del apéndice 2.

Configurar correo electrónico del usuario ejecutando "*git config --global user.email 'correo_usuario*'".

```
juliofernando@Theroths:~$ git config --global user.email "juliofernando.oe@gmail.com"
juliofernando@Theroths:~$
```

**Crear repositorio local clonando el repositorio origen**

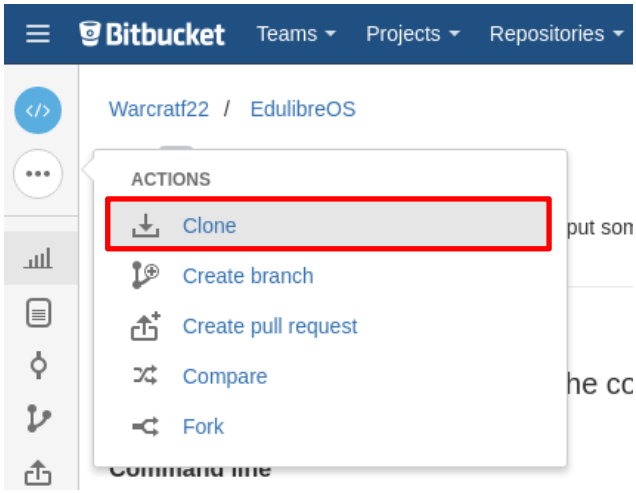
Para tener una copia del repositorio origen (repositorio Bitbucket), se deberá crear un directorio local para guardar el repositorio localmente, clonando el repositorio origen, que creará una copia exacta del repositorio origen al repositorio local.

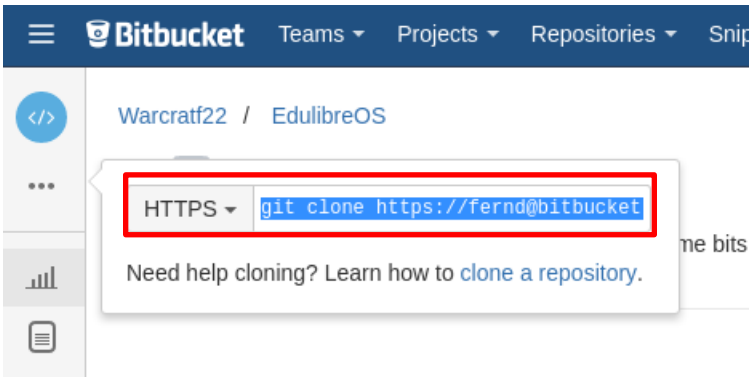
Crear el directorio "Repositorios" que contendrá los repositorios de todos los proyectos y acceder al nuevo directorio "Repositorios".

```
juliofernando@Theroths:~/Repositorios
```

Obtener *url* para clonar el repositorio origen (repositorio Bitbucket):

Continuación del apéndice 2.





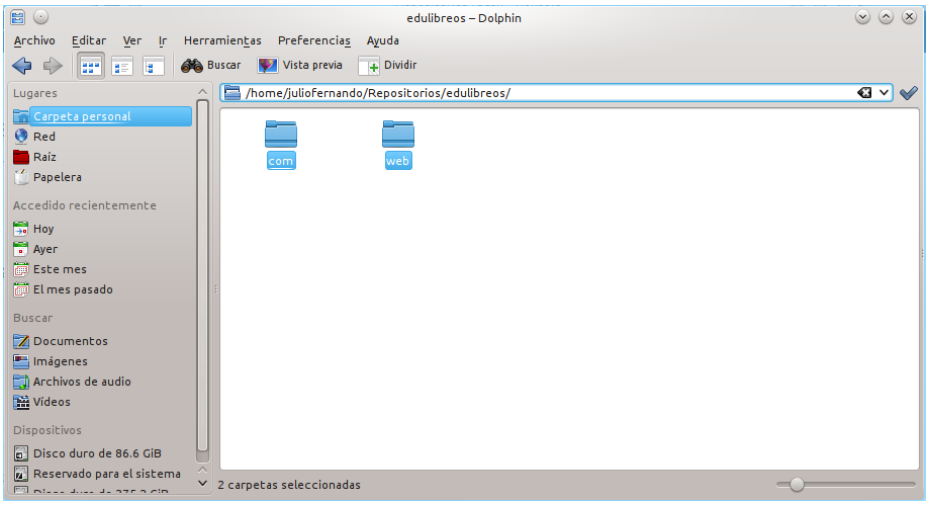

Clonar repositorio origen (repositorio Bitbucket) al repositorio local, ejecutando el comando "*git clone link_repositorio*":

```
juliofernando@Theroths:~/Repositorios$ git clone https://fernd@bitbucket.org/fernd/edulibreos.git
Clonar en «edulibreos»...
Password for 'https://fernd@bitbucket.org':
warning: Parece que ha clonado un repositorio vacío.
Checking connectivity... hecho.
juliofernando@Theroths:~/Repositorios$
```

**Realizar cambios al repositorio local y confirmarlos**

Los miembros del equipo de trabajo que desean agregar cambios al proyecto podrán realizarlo agregando directorios, archivos o realizando modificaciones sobre los directorios y archivos existentes. Se deberán agregar los cambios en el directorio del repositorio sobre el cual se está trabajando de forma gráfica o a consola.

Continuación del apéndice 2.




En consola acceder al repositorio sobre el que está trabajando.

```
juliofernando@Theroths:~/Repositorios$ cd edulibreos/
juliofernando@Theroths:~/Repositorios/edulibreos$
```

Agregar los directorios u archivos nuevos utilizando el comando "*git add directorio/archivo*", para que sean añadidos al repositorio local y con el comando "*git status*" poder ver el estado del repositorio. Se muestran los cambios realizados pero que deben ser confirmados:

Continuación del apéndice 2.

```
juliofernando@Theroths:~/Repositorios/edulibreos$ git add web/
juliofernando@Theroths:~/Repositorios/edulibreos$ git add com/
juliofernando@Theroths:~/Repositorios/edulibreos$ git status
En la rama master

Commit inicial

Cambios para hacer commit:
  (use «git rm --cached <archivo>...« para eliminar stage)

        new file:   com/ac.js
        new file:   com/mov.js
        new file:   web/about
        new file:   web/index.html
        new file:   web/menu.html
        new file:   web/nuevo.html

juliofernando@Theroths:~/Repositorios/edulibreos$
```

Confirmar los cambios realizados con el comando "*git commit*" en el repositorio local:

```
juliofernando@Theroths:~/Repositorios/edulibreos$ git commit -m "commit inicial"
[master (root-commit) 2a36a09] commit inicial
 6 files changed, 6 insertions(+)
 create mode 100644 com/ac.js
 create mode 100644 com/mov.js
 create mode 100644 web/about
 create mode 100644 web/index.html
 create mode 100644 web/menu.html
 create mode 100644 web/nuevo.html
juliofernando@Theroths:~/Repositorios/edulibreos$
```

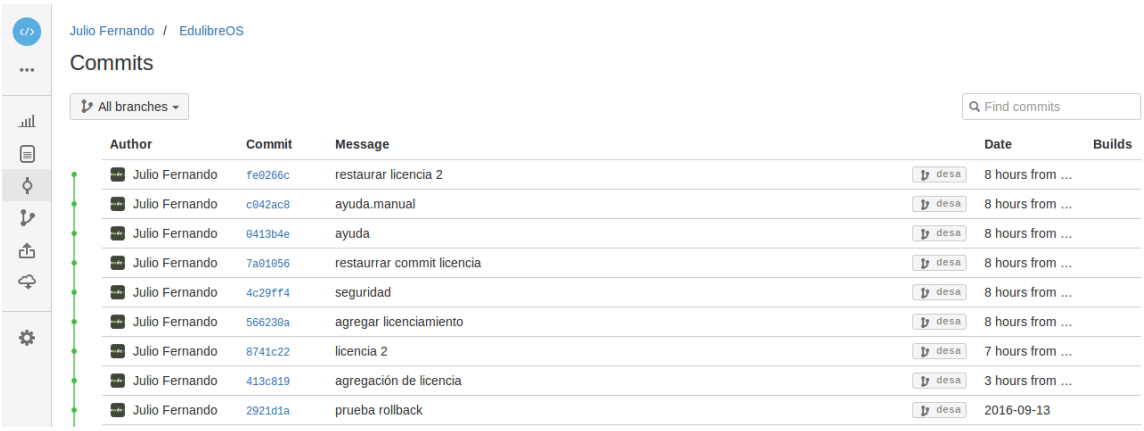
Revisar el estado del proyecto con el comando "*git status*", donde se indica que se está trabajando sobre el *branch* master (*branch* principal) y que no hay cambios pendientes ni confirmaciones sobre cambios realizados.

Continuación del apéndice 2.

```
juliofernando@Theroths:~/Repositorios/edulibreos$ git status
En la rama master
Your branch is based on 'origin/master', but the upstream is gone.
  (use "git branch --unset-upstream" to fixup)

nothing to commit, working directory clean
juliofernando@Theroths:~/Repositorios/edulibreos$
```

**Revertir y eliminar cambios confirmados**

El desarrollador, además de confirmar los cambios realizados, podrá revertirlos o eliminarlos de forma local, es decir que la eliminación de confirmaciones o reversión únicamente se reflejará en el repositorio local, donde posteriormente los podrá actualizar al repositorio origen. Eliminar los cambios, lo cual no se recomienda, y si se realiza se deberá hacer con mucho cuidado, debido que no se puede recuperar ninguna modificación de las confirmaciones que se eliminaron. Se muestran las confirmaciones en el repositorio origen.




Para mostrar la bitácora de confirmaciones en el repositorio local se utilizará el comando:

Continuación del apéndice 2.

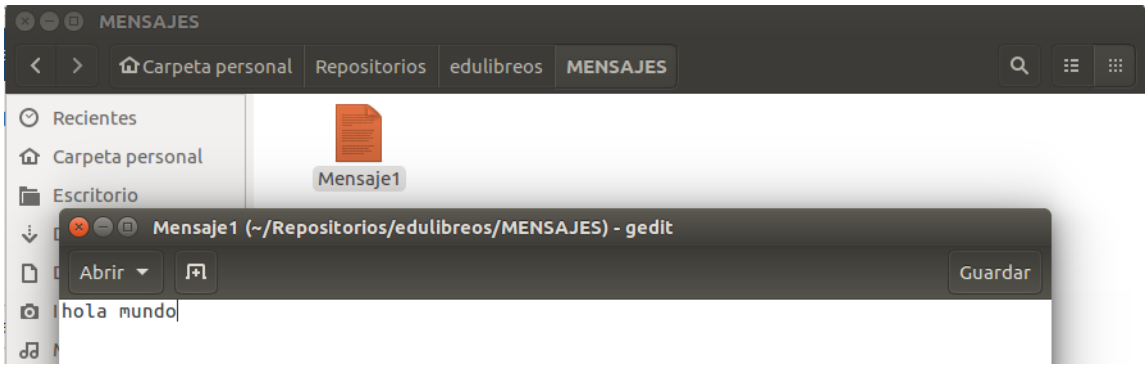
```
fernd@Fernd:~/Repositorios/edulibreos$ git log
```

Bitácora de confirmaciones del log local, donde la última confirmación tiene la descripción “restaurar licencia 2”.

```
commit fe0266c48f371c3d525a61bb25fb5b8818c731bd
Author: Julio Fernando <juliofernando.oe@gmail.com>
Date:   Mon Sep 19 16:23:22 2016 -0600

    restaurar licencia 2

commit c042ac87f3673bda0f379e85fc9f1547ea0c8aa1
Author: Julio Fernando <juliofernando.oe@gmail.com>
Date:   Mon Sep 19 16:09:27 2016 -0600

    ayuda.manual

commit 0413b4ea9c9df285e38662918afe1be26b88f895
Author: Julio Fernando <juliofernando.oe@gmail.com>
Date:   Mon Sep 19 16:06:28 2016 -0600

    ayuda

commit 7a01056784f10a277e1155c67bf4eba936ab296c
Author: Julio Fernando <juliofernando.oe@gmail.com>
Date:   Mon Sep 19 16:04:10 2016 -0600

    restaurrar commit licencia
:
```

Se agrega una modificación y se confirma:

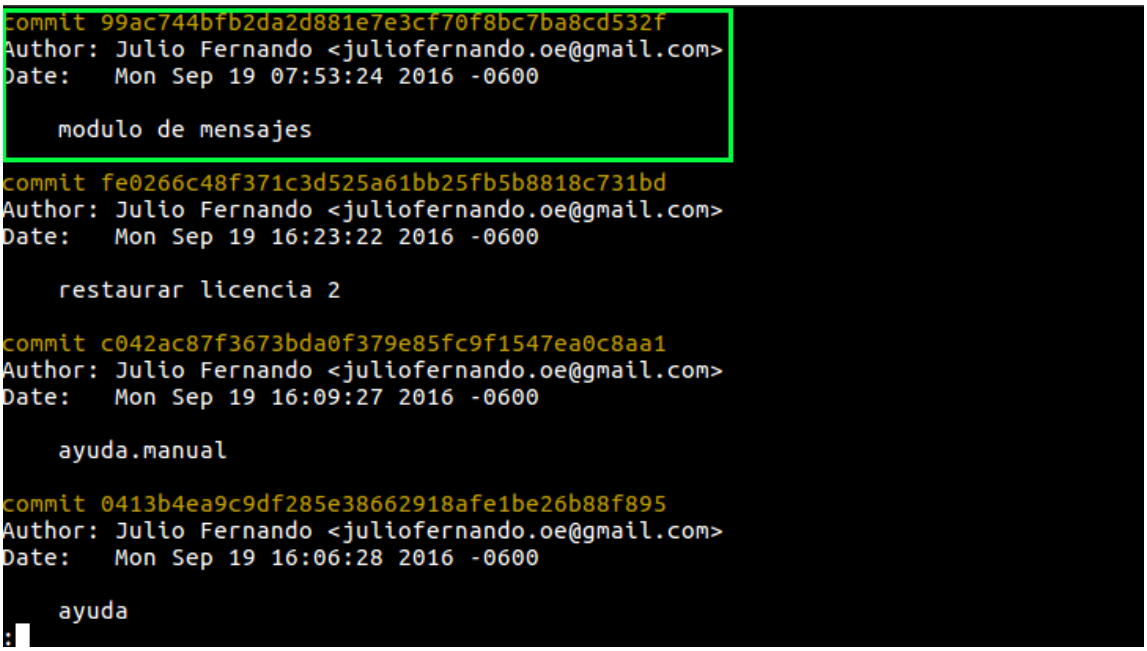
Continuación del apéndice 2.



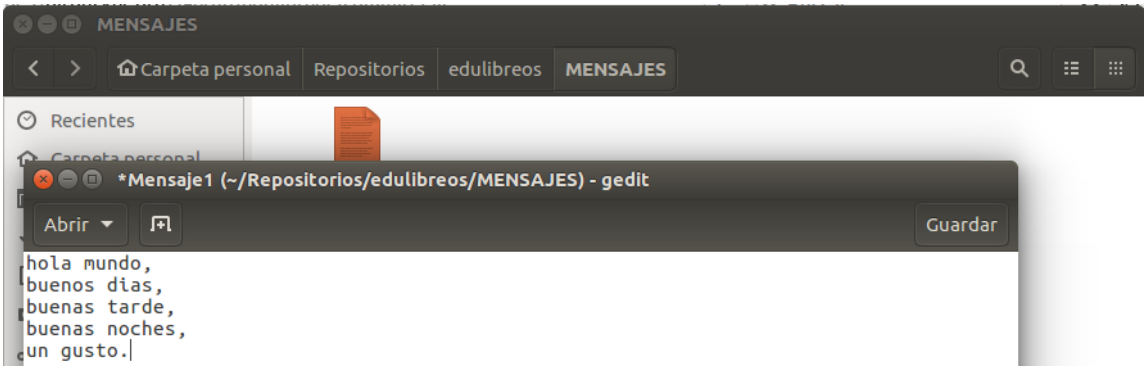

```
fernd@Fernd:~/Repositorios/edulibreos$ git status
En la rama desa
Su rama está actualizada con «origin/desa».
Archivos sin seguimiento:
  (use «git add <archivo>...» para incluir en lo que se ha de confirmar)

        MENSAJES/

no se ha agregado nada al commit pero existen archivos sin seguimiento (use «git
 add» para darle seguimiento)
fernd@Fernd:~/Repositorios/edulibreos$ git add MENSAJES
fernd@Fernd:~/Repositorios/edulibreos$ git status
En la rama desa
Su rama está actualizada con «origin/desa».
Cambios para hacer commit:
  (use «git reset HEAD <archivo>...» para sacar del stage)

        nuevo archivo: MENSAJES/Mensaje1

fernd@Fernd:~/Repositorios/edulibreos$ git commit -m "modulo de mensajes"
[desa 99ac744] modulo de mensajes
 1 file changed, 1 insertion(+)
 create mode 100644 MENSAJES/Mensaje1
fernd@Fernd:~/Repositorios/edulibreos$
```

Se muestra nuevamente la bitácora de las confirmaciones del repositorio local, con la nueva confirmación “módulo de mensajes”:

Continuación del apéndice 2.

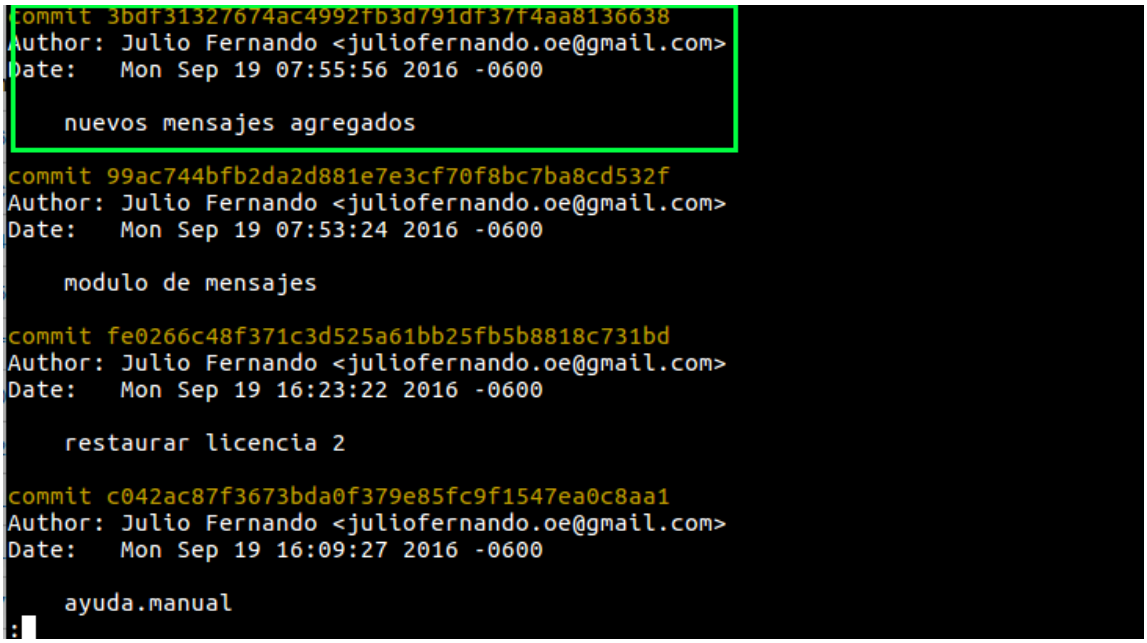



Se agrega de nuevo una nueva modificación y la se confirma:




Se muestra nuevamente la bitácora de confirmaciones del repositorio local, con la nueva confirmación “nuevos mensajes agregados”:

Continuación del apéndice 2.




Ahora se procede a retornar a una confirmación anterior, eliminando todas las confirmaciones posteriores a la del punto de retorno que se aplica en el repositorio local, utilizando el siguiente comando:

```
fernd@Fernd:~/Repositorios/edulibreos$ git reset --hard
```

Se retornará una confirmación, eliminando la última confirmación del repositorio local:

Continuación del apéndice 2.

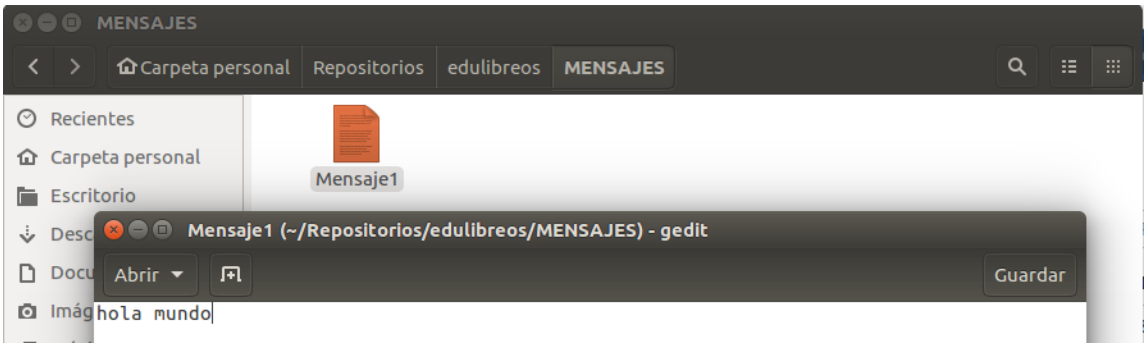
Utilizando el siguiente comando e indicando el identificador de la confirmación a la que se desea retornar, que se obtiene de "*commit*", seleccionando los primeros 7 caracteres de izquierda a derecha:

```
fernd@Fernd:~/Repositorios/edulibreos$ git reset --hard 99ac744
```

Se muestra nuevamente la bitácora de las confirmaciones del repositorio local para confirmar que la última confirmación fue eliminada exitosamente:

```
commit 99ac744bfb2da2d881e7e3cf70f8bc7ba8cd532f
Author: Julio Fernando <juliofernando.oe@gmail.com>
Date:   Mon Sep 19 07:53:24 2016 -0600

    modulo de mensajes

commit fe0266c48f371c3d525a61bb25fb5b8818c731bd
Author: Julio Fernando <juliofernando.oe@gmail.com>
Date:   Mon Sep 19 16:23:22 2016 -0600

    restaurar licencia 2

commit c042ac87f3673bda0f379e85fc9f1547ea0c8aa1
Author: Julio Fernando <juliofernando.oe@gmail.com>
Date:   Mon Sep 19 16:09:27 2016 -0600

    ayuda.manual

commit 0413b4ea9c9df285e38662918afe1be26b88f895
Author: Julio Fernando <juliofernando.oe@gmail.com>
Date:   Mon Sep 19 16:06:28 2016 -0600

    ayuda
:
```

Se muestra que los últimos cambios fueron eliminados y restaurados a la confirmación a la que se retorna:



Continuación del apéndice 2.

Si se desea revertir los cambios confirmados, podrán revertir a una confirmación anterior, revirtiendo los anteriores cambios confirmados y aplicando una confirmación después de realizar la reversión, para crear una confirmación con los mismos cambios a la confirmación que se revirtió. Utilizando el siguiente comando e indicando el identificador de la confirmación, se revertirán los cambios a la confirmación indicada:

```
fernd@Fernd:~/Repositorios/edulibreos$ git revert fe0266c
```

La diferencia entre *reset* y *revert* es que *revert* no elimina las confirmaciones posteriores a la confirmación de retorno; *revert* crea una nueva confirmación idéntica a la confirmación de retorno para que estén disponibles las confirmaciones que se revirtieron.

**Crear *branch***

El coordinador de desarrollo deberá crear los *branch* master de todos los repositorios y deberá crear los *branch* para desarrollo y para correcciones de cada repositorio. Además, por cada proyecto que inicie su desarrollo, deberá crear un *branch* a partir del *branch* de desarrollo.

Crear el *branch* con el comando "*git branch nombre_branch*":

```
juliofernando@Theroths:~/Repositorios/edulibreos$ git branch desarrollo
juliofernando@Theroths:~/Repositorios/edulibreos$
```

Verificando el estado del repositorio, se puede notar que a pesar de crear el nuevo *branch* desarrollo aún se está sobre el *branch master*, lo cual sucede porque el *branch* solo se creó localmente y aún falta subirlo al origen.

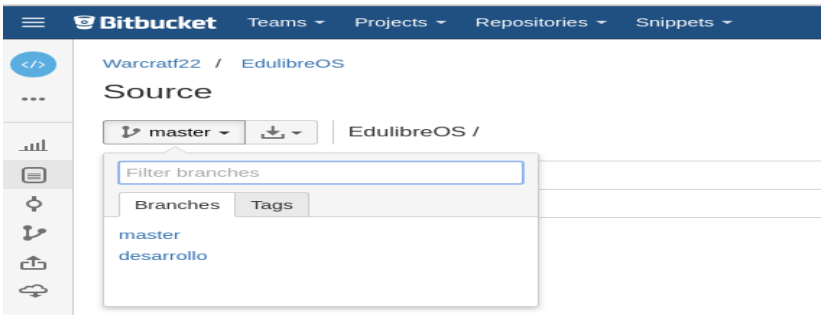
Continuación del apéndice 2.

```
juliofernando@Theroths:~/Repositorios/edulibreos$ git branch desarrollo
juliofernando@Theroths:~/Repositorios/edulibreos$ git status
En la rama master
Your branch is up-to-date with 'origin/master'.

nothing to commit, working directory clean
juliofernando@Theroths:~/Repositorios/edulibreos$
```

Para aplicar los cambios realizados y confirmados en el repositorio local se debe realizar un *push*, esto aplica también para la creación y eliminación de un *branch*, por lo que para que el *branch* se cree en el origen es necesario hacer un *push*.

```
juliofernando@Theroths:~/Repositorios/edulibreos/web$ git push origin desarrollo
Password for 'https://fernd@bitbucket.org':
Total 0 (delta 0), reused 0 (delta 0)
remote:
remote: Create pull request for desarrollo:
remote:   https://bitbucket.org/fernd/edulibreos/pull-requests/new?source=desarrollo&t=1
remote:
To https://fernd@bitbucket.org/fernd/edulibreos.git
 * [new branch]      desarrollo -> desarrollo
juliofernando@Theroths:~/Repositorios/edulibreos/web$
```

Captura de pantalla con el *branch* desarrollo creado y el *branch master*:




**Cambiar de *branch***

Cambiar de *branch* con el comando "*git checkout nombre_branch*", si se ejecuta correctamente indicará el cambio de *branch*:

Continuación del apéndice 2.

```
juliofernando@Theroths:~/Repositorios/edulibreos$ git checkout desarrollo
Switched to branch 'desarrollo'
juliofernando@Theroths:~/Repositorios/edulibreos$
```

**Fusionar *branch***

Consiste en fusionar los cambios de las fuentes de dos repositorios, que incluyen directorios, archivos y cambios en los archivos.

Cambiar al *branch* al que se le fusionaran los cambios del mismo y de otro *branch*:

```
juliofernando@Theroths:~/Repositorios/edulibreos$ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
juliofernando@Theroths:~/Repositorios/edulibreos$
```

Ejecutar el *merge* sobre el *branch* activo para fusionarlo con otro *branch*:

```
juliofernando@Theroths:~/Repositorios/edulibreos$ git merge desarrollo
Updating 2a36a09..c3e0bb1
Fast-forward
 web/prueba.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 web/prueba.txt
juliofernando@Theroths:~/Repositorios/edulibreos$
```

**Actualizar repositorio origen**

Es recomendado actualizar el repositorio local antes de actualizar el repositorio origen, para obtener los cambios que se pudieran encontrar en el repositorio origen y que no se encuentran en el repositorio local. Para actualizar el repositorio origen con los cambios realizados y confirmados en el repositorio local, se deberá ejecutar el comando "*git push origin nombre_branch*" para actualizar los cambios realizados y confirmados de un *branch* del repositorio local hacia el repositorio origen sobre el *branch* especificado:

Continuación del apéndice 2.

```
juliofernando@Theroths:~/Repositorios/edulibreos$ git push origin master
Password for 'https://fernd@bitbucket.org':
To https://fernd@bitbucket.org/fernd/edulibreos.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://fernd@bitbucket.org/fernd/edulibreos.git'
consejo: Updates were rejected because the remote contains work that you do
consejo: not have locally. This is usually caused by another repository pushing
consejo: to the same ref. You may want to first integrate the remote changes
consejo: (e.g., 'git pull ...') before pushing again.
consejo: See the 'Note about fast-forwards' in 'git push --help' for details.
juliofernando@Theroths:~/Repositorios/edulibreos$
```

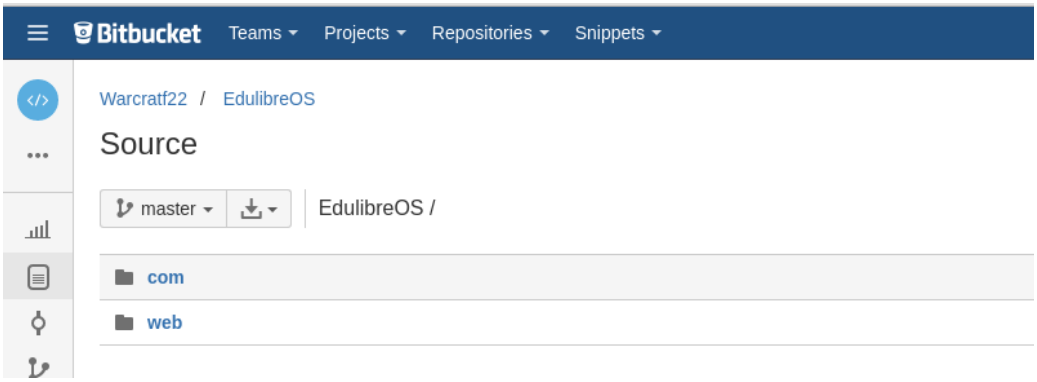
Esto indica que existen conflictos que no tienen los cambios del repositorio origen (repositorio Bitbucket) en el repositorio local, y que debe solventarse el conflicto aplicando los cambios del repositorio origen al repositorio local (si aplica).

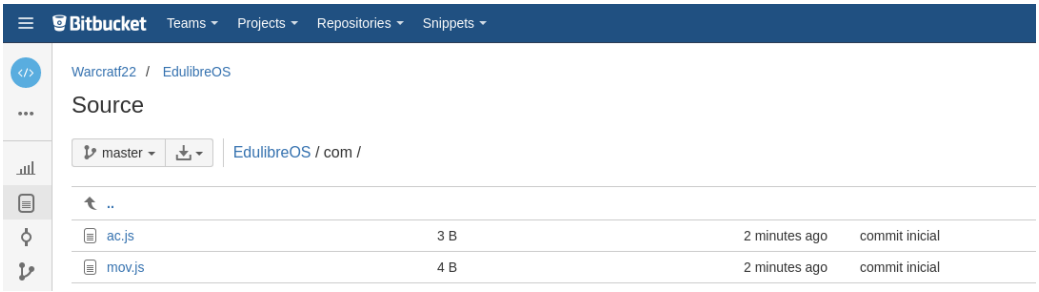
Nuevamente se intenta actualizar el repositorio origen de la misma forma indicada:

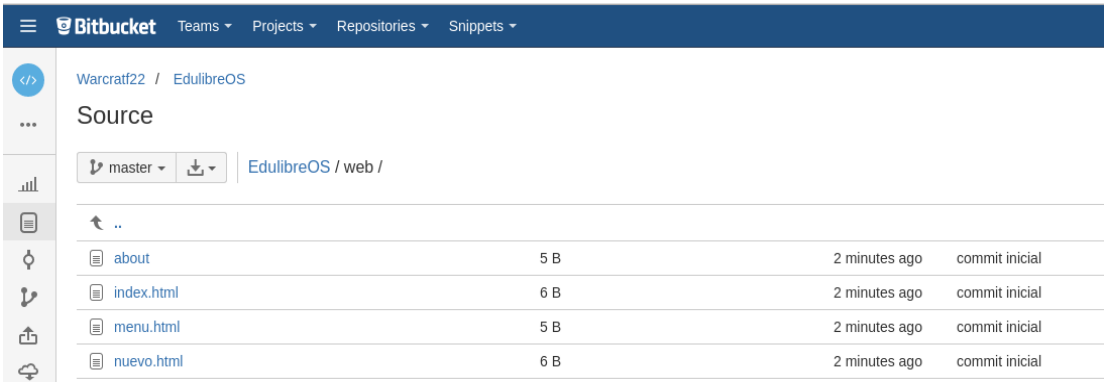
```
juliofernando@Theroths:~/Repositorios/edulibreos$ git push origin master
Password for 'https://fernd@bitbucket.org':
Counting objects: 7, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 390 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To https://fernd@bitbucket.org/fernd/edulibreos.git
   b483bff..0a4b605  master -> master
juliofernando@Theroths:~/Repositorios/edulibreos$
```

En la siguiente página se muestran capturas de pantalla con los cambios de directorios y archivos en el repositorio origen (repositorio Bitbucket):

Continuación del apéndice 2.




**Actualizar repositorio local**

Antes de realizar una actualización al repositorio origen, siempre se deberá actualizar el repositorio local con la información del repositorio origen.

Continuación del apéndice 2.

Accediendo al directorio del repositorio que se desea actualizar (ej. EdulibreOS), se ejecuta el comando "*git pull --all*" que agregará los cambios del repositorio origen al repositorio local:

```
juliofernando@Theroths:~/Repositorios/edulibreos$ git pull --all
Fetching origin
Password for 'https://fernd@bitbucket.org':
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
De https://bitbucket.org/fernd/edulibreos
   2a36a09..b483bff  master     -> origin/master
Merge made by the 'recursive' strategy.
 web/prueba2 | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 web/prueba2
juliofernando@Theroths:~/Repositorios/edulibreos$
```

## 2.2. PASO 2: Implementación de integración continua

Para continuar se deberá implementar el sistema de integración continua, el cual, a partir de los cambios subidos al repositorio origen o a las ramas, realizará un *build* para verificar que no existan errores. Mientras más pronto los desarrolladores suban sus cambios, será más fácil y rápido detectar alguna anomalía en el código.

### 2.2.1. Instalar Jenkins

La instalación de Jenkins estará a cargo del coordinador de desarrollo y deberá realizarse solamente una vez. A continuación se describen los pasos a seguir para instalar Jenkins:

#### 2.2.1.1. Instalar Java

El coordinador de desarrollo deberá verificar si en el sistema ya se encuentra instalado Java. Si no, será necesario que realice la instalación del *jre* y *jdk*. (Guía – Instalar Java).

#### 2.2.1.2. Instalar Jenkins

Una vez instalado Java, el coordinador de desarrollo deberá instalar Jenkins a través de los comandos correspondientes. Al finalizar la instalación y, si esta fue correcta, se podrá acceder a la interfaz *web* a través del navegador y de la dirección del ordenador y el puerto 8080. (Guía – Instalar Jenkins).

### 2.2.2. Configurar Jenkins

La configuración de Jenkins estará a cargo del coordinador de desarrollo y deberá realizarse solamente una vez. A continuación se describen los pasos a seguir para configurar Jenkins.

#### 2.2.2.1. Instalar *plugin* de Git

Para poder integrar Jenkins con Git es necesario descargar e instalar el *plugin* con nombre “Git *plugin*” en Jenkins. Este paso se realiza fácilmente desde la interfaz web, ya que la misma realiza la descarga e instalación, el usuario solo debe buscar el *plugin* a instalar. (Guía – Instalar *plugin* en Jenkins).

#### 2.2.2.2. Instalar *plugin* de Bitbucket

De igual manera que el paso anterior, para poder integrar Jenkins con Bitbucket es necesario descargar e instalar el *plugin* con nombre “Bitbucket *plugin*” en Jenkins. A pesar de que Git y Bitbucket ya se encuentran integrados, es necesario instalar los *plugins* en Jenkins. (Guía – Instalar *plugin* en Jenkins).

#### 2.2.2.3. Crear tarea

Una vez descargados e instalados los dos *plugins*, será necesario crear una tarea en Jenkins por cada *build* que se vaya a realizar, ya que cada tarea tendrá la configuración a ejecutar para cada proyecto al momento de que un desarrollador suba sus cambios al origen. Para este proyecto se crearán dos tareas, una para el escritorio Innova y otra para EdulibreOS, ya que cada uno de estos genera salidas diferentes. (Guía – Crear tarea en Jenkins).

### 2.2.3. Configurar repositorio en Bitbucket

La configuración del repositorio en Bitbucket estará a cargo del coordinador de desarrollo y deberá realizarse solamente una vez. A continuación se definen los pasos para configurar el repositorio en Bitbucket.

#### 2.2.3.1. Crear *webhook*

En Bitbucket será necesario crear un *webhook* que permita la interacción entre Bitbucket y Jenkins, puesto que se requiere que al momento de subir los cambios al repositorio se lance una llamada a Jenkins y que este llame a la

tarea que realiza el *build* correspondiente. (Guía – Crear *webhook* en Bitbucket).

### 2.2.4. Guía

**Instalar Java**

Instalar el jre de java ejecutando el comando "*sudo apt-get install openjdk-7-jre*":

```
juliofernando@Theroths:~$ sudo apt-get install openjdk-7-jre
[sudo] password for juliofernando:
```

Luego instalar el jdk de java ejecutando el comando "*sudo apt-get install openjdk-7-jdk*":

```
juliofernando@Theroths:~$ sudo apt-get install openjdk-7-jdk
[sudo] password for juliofernando:
```

Para comprobar que la instalación fue correcta, ejecutar el comando "*java -version*":

```
juliofernando@Theroths:~$ java -version
java version "1.7.0_101"
OpenJDK Runtime Environment (IcedTea 2.6.6) (7u101-2.6.6-0ubuntu0.14.04.1)
OpenJDK 64-Bit Server VM (build 24.95-b01, mixed mode)
juliofernando@Theroths:~$
```

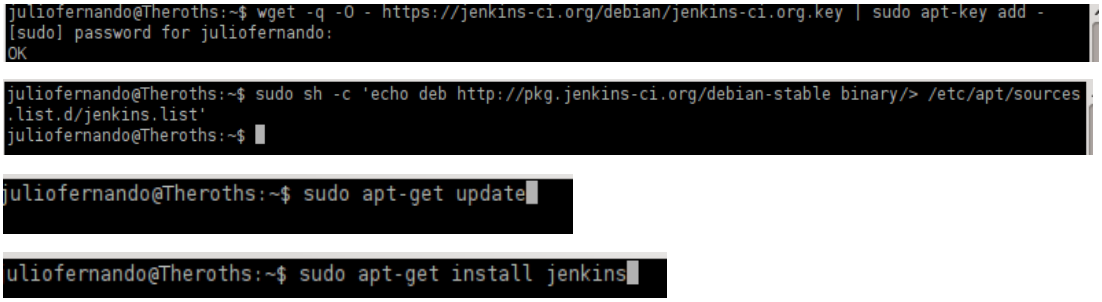
**Instalar Jenkins**

Para ejecutar la instalación de Jenkins es necesario ejecutar los siguientes comandos, uno a uno:

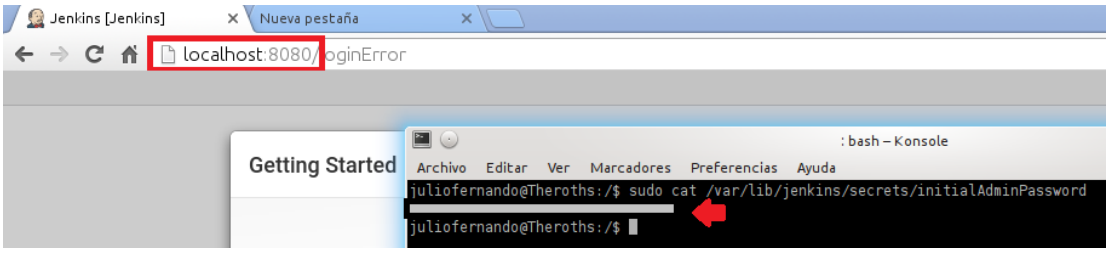
- *wget –q –O – https://jenkins-ci.org/debian/jenkins-ci.org.key | sudo apt-key add –*

Continuación del apéndice 2.

- *sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'*
- *sudo apt-get update*
- *sudo apt-get install Jenkins*

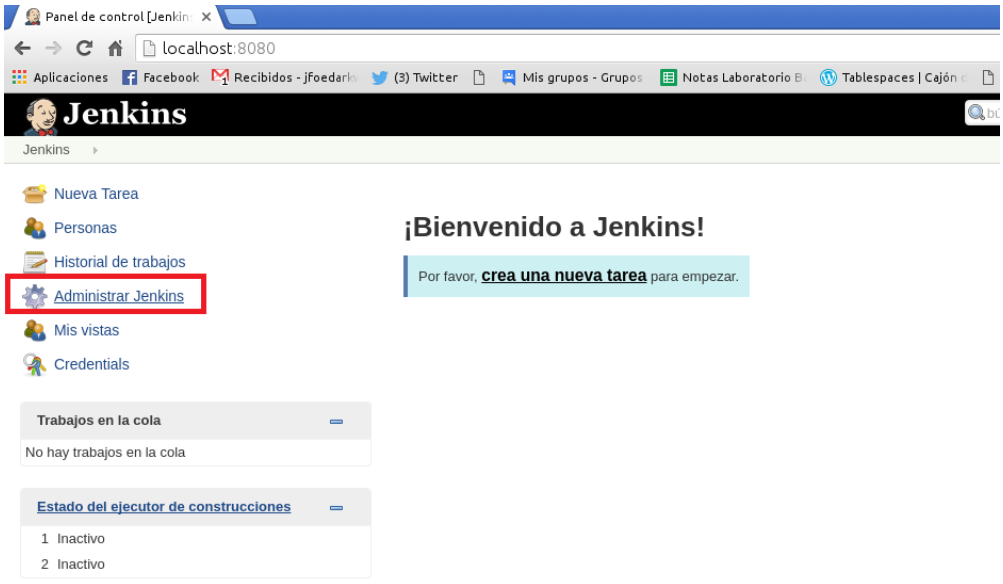



Para comprobar que la instalación fue correcta se debe abrir un navegador e ingresar a la dirección “IP/8080”. Al cargar la página requerirá una contraseña, la cual se encuentra en la dirección “/var/lib/jenkins/secrets/initialAdminPassword”. Obtener la contraseña e ingresarla para así poder empezar a utilizar Jenkins:




**Instalar *plugin* en Jenkins**

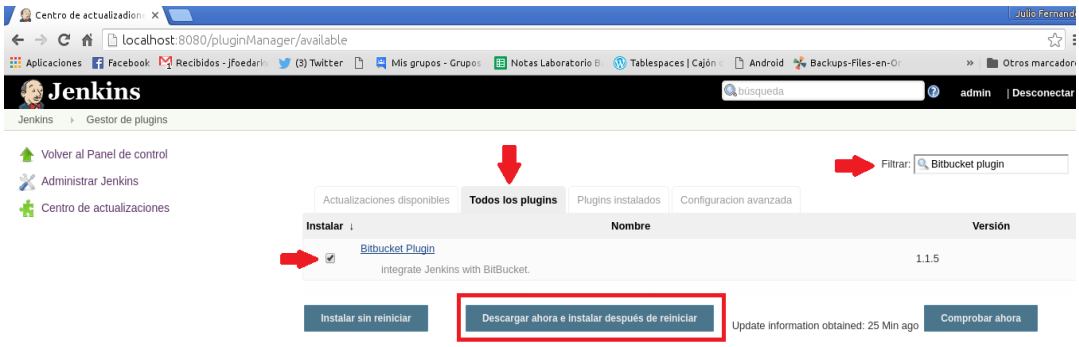
- Seleccionar la opción “Administrar Jenkins”:



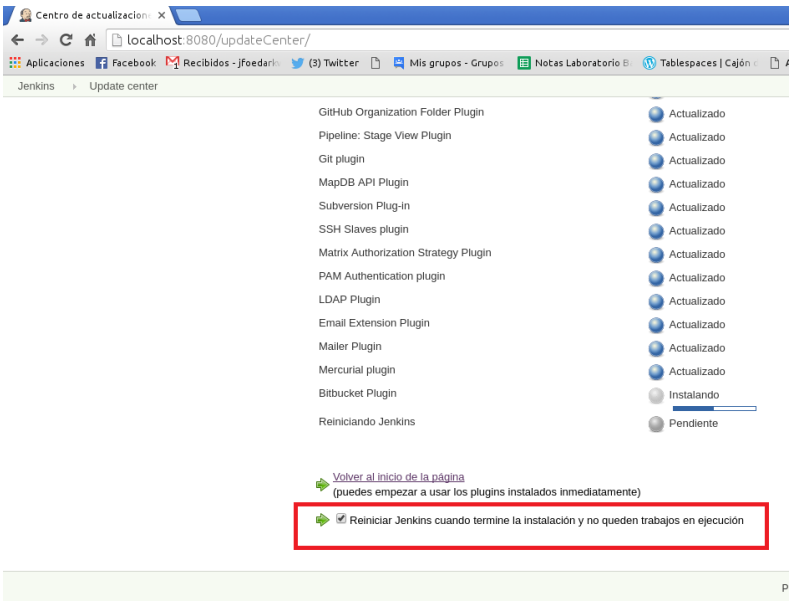

- Buscar y seleccionar la opción “Administrar Plugins”:


- Elegir la pestaña “Todos los *plugins*”. En la opción de búsqueda escribir el nombre del *plugin* a instalar, por ejemplo “Bitbucket *plugin*”. Seleccionar la opción desplegada por la búsqueda y que concuerde con

el nombre, luego dar clic en el botón “Descargar ahora e instalar después de reiniciar”:



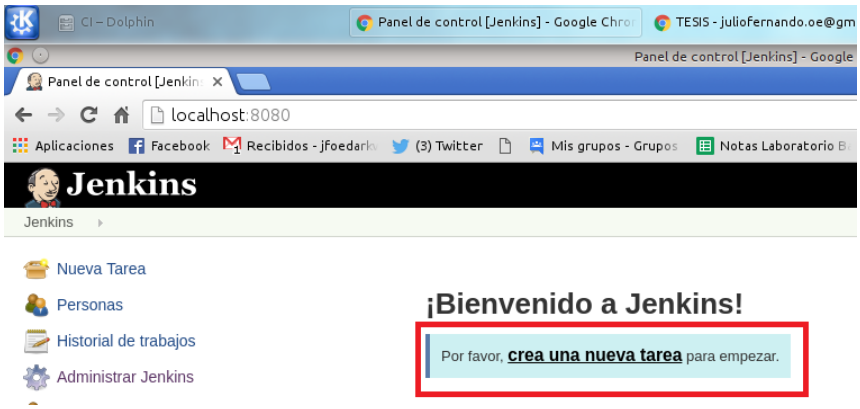

- Chequear la opción “Reiniciar Jenkins cuando termine la instalación y no queden trabajos en ejecución”:




**Crear tarea en Jenkins**

- Seleccionar la opción “Crear una nueva tarea”:

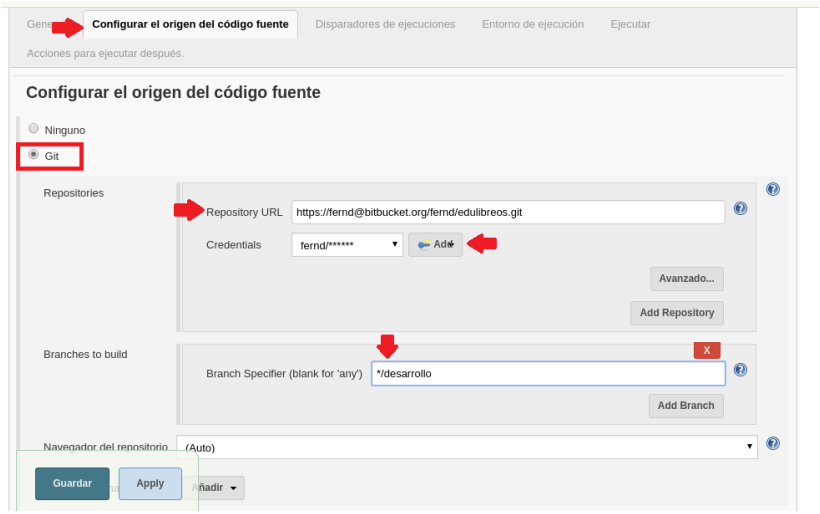
Continuación del apéndice 2.



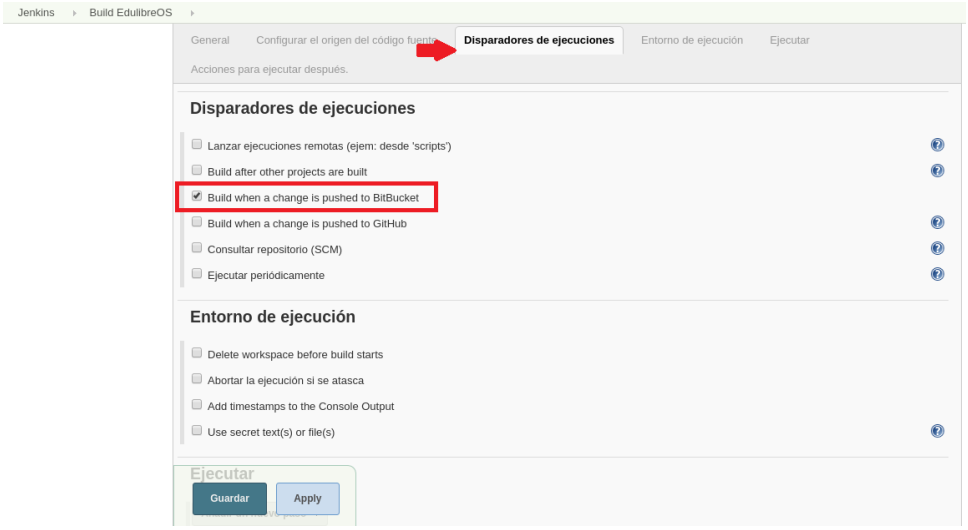

- Ingresar un nombre para la tarea y elegir la opción “Crear un proyecto de estilo libre”:


- Dirigirse a la opción “Configurar el origen del código fuente” y seleccionar la opción Git. Ingresar la URL que brinda el repositorio de Bitbucket. Por ser un repositorio privado, será necesario agregar las credenciales. En la opción “*Branches to build*” se puede especificar el nombre de la rama dentro del repositorio a la que se le va a realizar el *build* con esta tarea:

Continuación del apéndice 2.



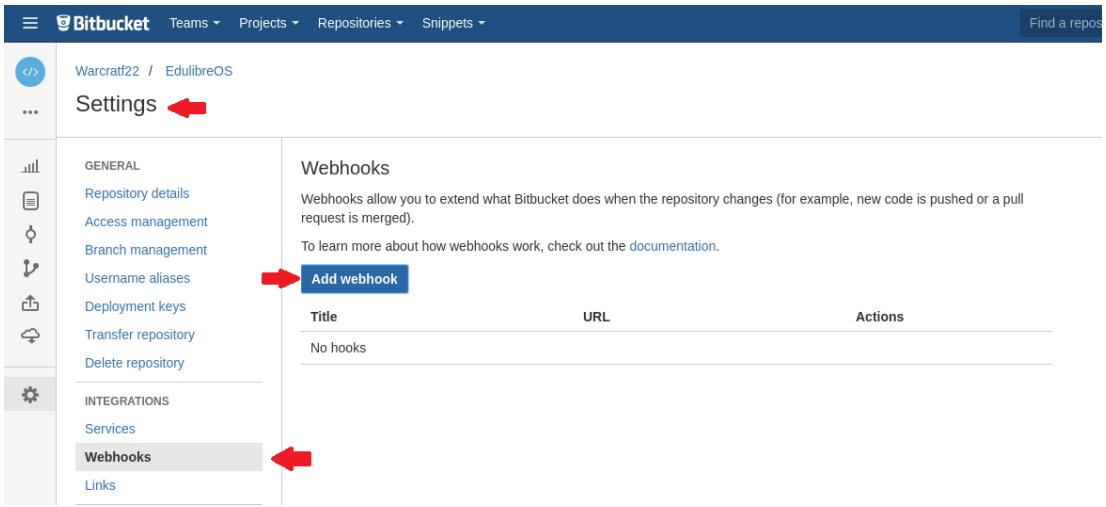

- En la opción “Disparadores de ejecuciones”, seleccionar la opción “*Build when a change is pushed to Bitbucket*”:



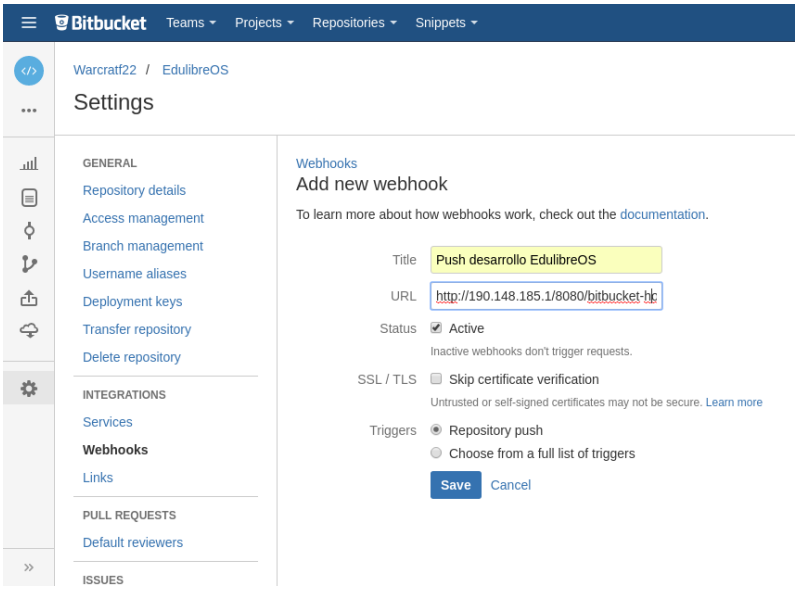

**Crear *webhook* en Bitbucket**

- Ingresar al repositorio en Bitbucket y seleccionar la opción “*Settings*”. Seleccionar la opción “*Webhooks*” y hacer clic en “*Add webhook*”:

Continuación del apéndice 2.




- Agregar un título y la URL a través de la cual se accede a Jenkins agregando al final /bitbucket-hook/. Es decir: localhost/8080/bitbucket-hook/:



Continuación del apéndice 2.

## 2.3. PASO 3: Implementación de pruebas

Como paso final, se debe implementar un sistema de pruebas y automatizarlas a través de las herramientas instaladas y configuradas con anterioridad. Las pruebas se realizan sobre las nuevas implementaciones de código dentro de los proyectos EdulibreOS e Innova. Estas ayudarán a que cada desarrollador pueda encontrar errores o fallas en su código y de igual manera validar que no afecte al código de otras personas.

### 2.3.1. Descargar y agregar el componente *UnitForm*

El proceso de pruebas debe ser implementado por todo desarrollador, por lo que este paso debe ser realizado por cada desarrollador en sus propias máquinas y deberá ser supervisado por el coordinador de desarrollo y de pruebas. El componente *UnitForm* ha sido desarrollado para el entorno Gambas y se encuentra disponible para su descarga. Una vez descargado, solamente hay que agregar el componente al proyecto y activarlo. (Guía – Descargar y agregar componente *UnitForm*).

### 2.3.2. Utilizar el componente *UnifForm*

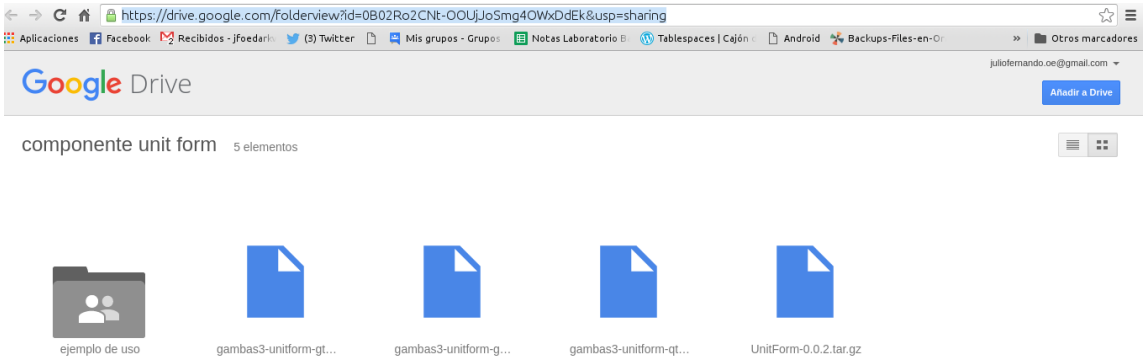
Una vez agregado el componente, será necesario que el coordinador de desarrollo en conjunto con el coordinador de pruebas creen las clases necesarias para implementar las pruebas en cada proyecto y actualicen el repositorio origen. Esto para que cuando los desarrolladores actualicen sus repositorios locales ya tengan a su disposición las clases y puedan utilizarlas. (Guía – Utilizar componente *UnitForm*).

Continuación del apéndice 2.

### 2.3.3. Guía

**Descargar y agregar componente *UnitForm***

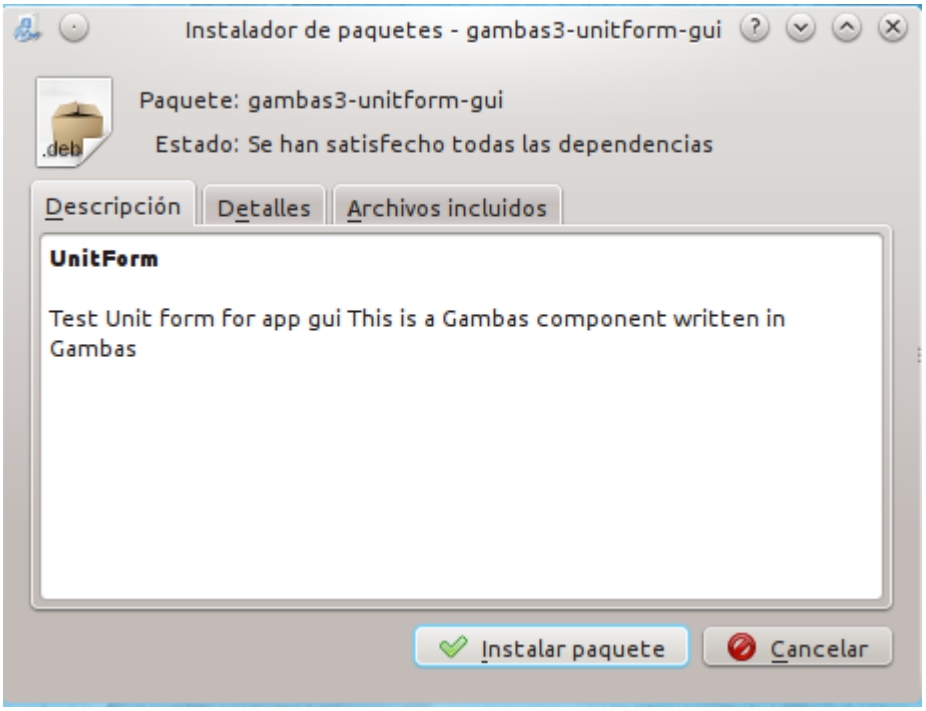
- Descargar los componentes de la página: https://drive.google.com/folderview?id=0B02Ro2CNt-OOUjJoSmg4OWxDdEk&usp=sharing





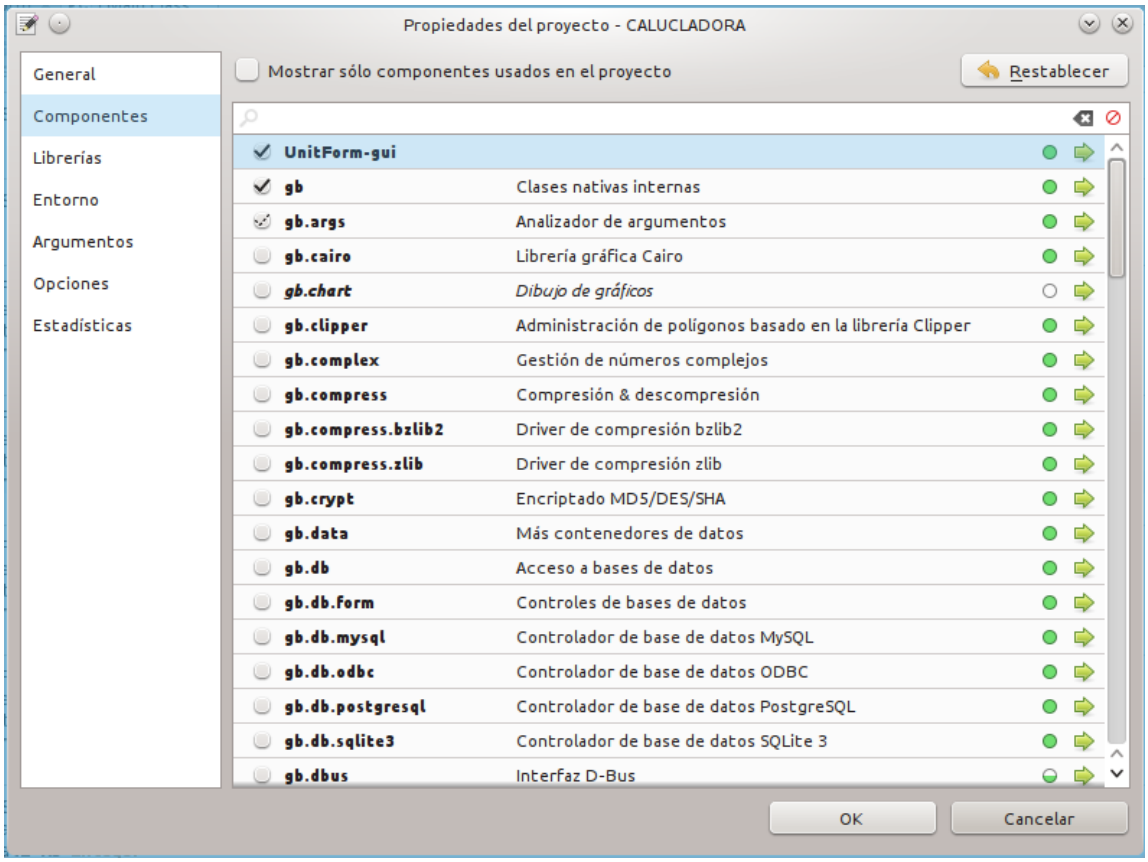
Continuación del apéndice 2.

- Instalar los 3 componentes en el sistema:



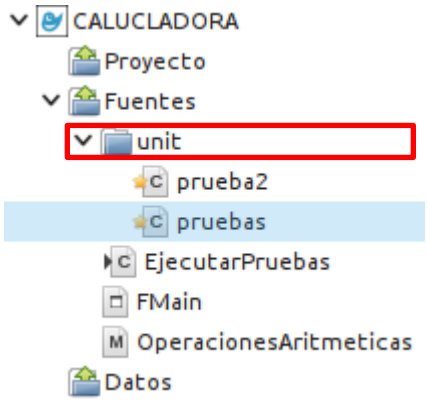

- En el proyecto en Gambas dirigirse a la opción “Proyecto”. Elegir la opción “Propiedades”.

- Dirigirse a la pestaña “Componentes” y buscar los componentes instalados. Estos aparecerán con el nombre de *UnitForm* al inicio. Seleccionar los componentes y dar clic en ok:



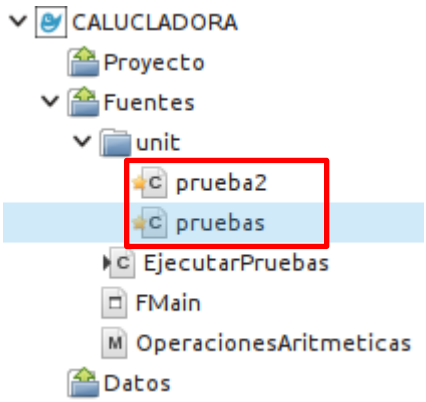

**Utilizar componente *UnitForm***

- En el proyecto crear una carpeta con el nombre "*unit*":



Continuación del apéndice 2.

- Dentro de la carpeta, agregar una clase en donde se implementaran las pruebas. Pueden ser varias clases:



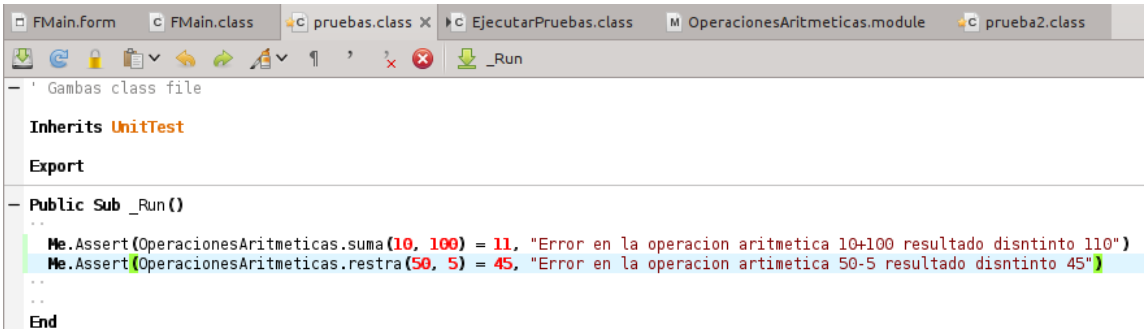

- En cada clase en que se implementen las pruebas se debe colocar al inicio, que hereda de la clase *UnitTest*:

```
' Gambas class file

Inherits UnitTest

Export
```

- En cada clase también se debe agregar el método:
  Public Sub_Run()
  End

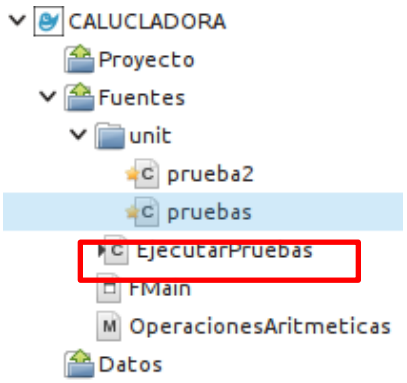
- Dentro del método *_Run*() se deben definir las pruebas que se van a realizar. Estas deben iniciar con la palabra reservada Me. La palabra Me seguida de un punto despliega una lista de posibles métodos a utilizar,

esto dependerá del valor que se quiera obtener como retorno. Y dentro se define la condición a evaluar:



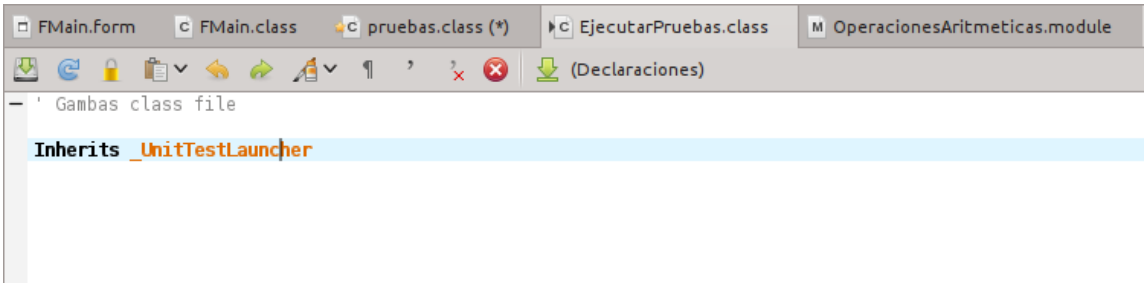

En el ejemplo se define una prueba, en donde si la función a probar devuelve el valor esperado, da como resultado verdadero, pero si no da como resultado falso y muestra el mensaje “Error”. Se puede agregar la cantidad de pruebas que sean necesarias.

- Se debe crear una clase fuera de la carpeta *unit*, la cual mandará a llamar a las pruebas:



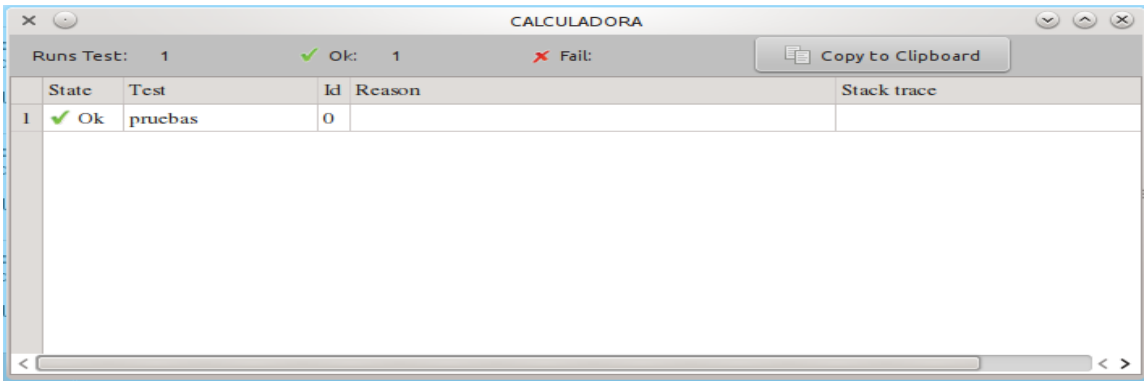
Continuación del apéndice 2.

Agregando el siguiente código, utilizando la herencia de la case *_UnitTestLauncher*:




- La clase creada con anterioridad será la que se mande a ejecutar para realizar las pruebas. Una vez se ejecuta despliega una ventana nueva en donde se muestran las pruebas realizadas y sus resultados.

Resultado de pruebas satisfactorias:



Continuación del apéndice 2.

Resultado de pruebas insatisfactorias: