Diseño de una Infraestructura TI para un Ambiente de Integración Continua en el Programa de Ingeniería de Sistemas de la Corporación Universitaria del Caribe CECAR

Jorge Enrique Gómez Casseres Barboza

Liseth Candelaria Paternina Pérez

Corporación Universitaria del Caribe – CECAR

Escuela de Posgrado y Educación Continua

Facultad de Ciencias Básicas, Ingenierías y Arquitectura

Especialización en Tecnologías de la Información

Sincelejo

2019

Diseño de una Infraestructura TI para un Ambiente de Integración Continua en el Programa de Ingeniería de Sistemas de la Corporación Universitaria del Caribe CECAR

Jorge Enrique Gómez Casseres Barboza
Liseth Candelaria Paternina Pérez

Trabajo de grado presentado como requisito para optar al título de Especialista en Tecnologías de la Información

Asesor: Ing. Jhon Jaime Méndez Alandete
Magister en Software Libre

Corporación Universitaria del Caribe – CECAR
Escuela de Posgrado y Educación Continua
Facultad de Ciencias Básicas, Ingenierías y Arquitectura
Especialización en Tecnologías de la Información
Sincelejo
2019

Nota de Aceptación

4.0

Director

Evaluador 1

Evaluador 2

Sincelejo, Sucre, 23 de octubre de 2019

Dedicatoria

A Dios por ser el patrocinador de mis sueños, a mis padres por ser mi ejemplo y apoyo incondicional, a mi esposo Fernando y mi hermosa hija Isabella, mi mayor fuente de motivación para seguir superándome, a mi mejor amigo y compañero de trabajo Jorge, con ellos en mi vida todo es posible.

Agradecimientos

Agradezco a Dios, mi familia y amigos por confiar en mí, por apoyarme y ser participe en cada uno de mis logros y metas. Gracias al excelente acompañamiento de nuestro asesor de trabajo, el Ingeniero Jhon Méndez, quien con su profesionalismo nos orientó y fue fundamental para la culminación de nuestro trabajo de grado.

Tabla de contenido

Tabla de Figuras

Resumen

En la actualidad, muchas universidades utilizan recursos clásicos como métodos de enseñanza en carreras enfocadas a las tecnologías de la información e Ingeniería del Software logrando una poca adaptación con las metodologías de desarrollo como las metodologías agiles. Por lo anterior se hace necesario realizar un diseño de un sistema de Integración Continua en el programa de Ingeniería de Sistemas de la Corporación Universitaria del Caribe CECAR, con el fin de aumentar la productividad de los equipos de desarrollo realizando pruebas de requerimientos no funcionales en un ambiente cliente servidor.

Palabras clave: Sistema de Integración continua, metodologías agiles

Abstract

At present, many universities use classical resources as teaching methods in careers focused on information technologies and software engineering, achieving little adaptation with development methodologies such as agile methodologies. Therefore, it is necessary to carry out a design of a continuous integration system in the Engineering program of the Caribbean University Corporation CECAR, in order to increase the productivity of the non-functional requirements test development teams in a client environment server.

Keywords: Continuous Integration System, agile methodologies

Introducción

El programa de Ingeniería de Sistemas perteneciente a la facultad de Ciencias Básicas, Ingenierías y Arquitectura de la Corporación Universitaria del Caribe - CECAR tiene como finalidad formar integralmente Ingenieros de Sistemas con sólida fundamentación en ciencias básicas, ciencias de la computación, ingeniería del software, humanidades e investigación, capaces de atender necesidades sociales y del sector productivo en el contexto regional, nacional e internacional, a través del desarrollo, adaptación, uso y gestión de recursos informáticos y tecnologías de la información. Siendo el desarrollo de software su eje temático principal.

En la actualidad, cada docente que apoya las asignaturas de las líneas relacionadas a las Tecnologías de la Información e Ingeniería del Software solicita semestralmente a los encargados de los laboratorios de informática, la instalación de las herramientas y servicios de TI necesarios para el desarrollo de sus planes de aula. Cabe resaltar que estas herramientas y servicios se instalan de manera local en cada una de las máquinas que posee el laboratorio, causando una pérdida de tiempo al auxiliar de laboratorio en estas funciones y no permitiendo a los estudiantes realizar pruebas de requerimientos no funcionales en un ambiente cliente servidor. Así mismo en cada clase los docentes deben suministrar a los estudiantes los avances desarrollados en proyectos de desarrollo de software a través de copias por pendrive, archivos en la nube, correo electrónico, carpeta en red, entre otros. Este aislamiento de las asignaturas que hacen parte de la línea de desarrollo de software que conforman el plan de estudios del programa de ingeniería de sistemas, en actividades de elicitación, planeación, gestión, análisis, diseño, implementación y prueba de productos de software, generan las siguientes consecuencias:

- Estudiantes y docentes sin Acceso y falta de pruebas de servicios de manera local (en cada máquina), sin posibilidad de identificar los comportamientos del aplicativo en máquinas remotas.
- Los estudiantes no pueden detectar y solucionar problemas de integración de forma continua, sino ocasionalmente, generando caos de última hora cuando se acercan las fechas de entrega de proyectos grupales sobre desarrollo de software.
- Los estudiantes y docentes no cuentan con disponibilidad constante de una versión para pruebas.
- Falta de monitorización del docente de forma continua al cumplimiento de métricas de calidad en los proyectos desarrollados por los estudiantes.

El presente trabajo de grado trata del diseño de una infraestructura TI para crear un sistema de Integración Continua a la vanguardia de las tendencias actuales en desarrollo de software, con el objetivo de solucionar los problemas presentados en el programa de ingeniería de sistemas de la Facultad De Ciencias Básicas, Ingenierías Y Arquitectura de la Corporación Universitaria del Caribe CECAR, tales como la falta de acceso, compartición y modificación de proyectos de software por parte de docentes y estudiantes durante el proceso de desarrollo de software.

1. Objetivos

1.1.General

Diseñar una infraestructura TI para un ambiente de Integración Continua en el programa de Ingeniería de Sistemas de la Corporación Universitaria del Caribe CECAR basado en estándares internacionales para el desarrollo de software.

1.2.Específicos

- ✓ Identificar el pensum académico del programa de Ingeniería de Sistemas para establecer las necesidades a nivel de herramientas de TI y Desarrollo de Software basadas en las metodologías de software adoptadas, para determinar la adaptabilidad al sistema de Integración Continua.

- ✓ Construir el diseño de una infraestructura de TI acorde a las necesidades del programa de ingeniería de sistema para la operatividad del currículo a nivel de Tecnologías de la información y desarrollo de software basado en estándares internacionales.

- ✓ Validar el diseño a través de la implementación de un prototipo que permitan realizar la integración continúa en un ambiente de desarrollo de software basado en las herramientas definidas en el pensum académico del programa de ingeniería de sistemas.

2. Marco Tecnológico

2.1. Tecnologías de la Información

Son un conjunto de servicios, redes, software, dispositivos que tienen como fin el mejoramiento de la calidad de vida de las personas dentro de un entorno, y que se integran a un sistema de información interconectado y complementario son una herramienta primordial en cualquier tipo de organización moderna, pública y privada. (Ruedas, 2013).

2.2.Ingeniería del software

La ingeniería del software es la aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, operación y mantenimiento del software, que es la aplicación de la ingeniería del software. (IEEE, 1990).

La ingeniería del software es una disciplina de ingeniería preocupada por todos los aspectos de la producción de software desde las primeras etapas de especificación del sistema hasta el mantenimiento del sistema después de que éste se haya puesto en uso. Se preocupa de las teorías, métodos y herramientas para el desarrollo profesional de software. (Laboratorio Nacional de Calidad del Software de INTECO, 2009).

La ingeniería del software abarca áreas muy diversas de la informática, en la que se destacan la construcción de compiladores o sistemas operativos y, la que más concierne para este trabajo de investigación, la integración continua, abordando todas las fases del ciclo de vida de desarrollo de cualquier tipo de sistema de información.

2.3.Estándares de Calidad de Software

Los estándares de calidad de software consisten en conceptos, estilos de documentos, metodologías para el diseño, procedimiento, prueba y análisis de software desarrollado, y técnicas acordadas por los creadores de software, cuya finalidad es ofrecer una mayor confiabilidad, mantenibilidad en concordancia con los requisitos exigidos, lo que permite elevar la productividad y el control en la calidad de software, parte de la gestión de la calidad se establecen a mejorar su eficacia y eficiencia. Los estándares de calidad de software deben ser avalados por un grupo de desarrolladores que contribuyen a la definición y mantenimiento de la norma o protocolo, como por ejemplo ISO 9000(ISO), HTML (W3C).

2.4.Tipos de Metodologías de Desarrollo

2.4.1. *Metodología en Cascada*

Es un modelo que empezó a diseñarse en el año 1966, el cual consiste en ordenar las etapas del proceso de desarrollo de software, en donde se especifica que el inicio de cada etapa debe realizarse justo después de la finalización de la etapa anterior. Al final de cada etapa se realiza una revisión final, la cual, determina si el proyecto está listo para avanzar a la siguiente fase. (Maida & Pacienzia, 2015).

2.4.2. *Metodología en Espiral*

La metodología espiral refleja la relación de tareas con prototipos rápidos, mayor paralelismo y concurrencia en las actividades de diseño y construcción. El método en espiral debe todavía ser planificado metódicamente, con las tareas y entregables identificados para cada paso en la espiral. (Maida & Pacienzia, 2015).

2.4.3. *Metodología de Prototipo*

Este modelo comienza con la recolección de requisitos, el desarrollador y que el cliente defina los objetivos globales para el software, originando así un diseño rápido que se centre en una representación de estos aspectos del software que sean visibles para el usuario o cliente. De este diseño surge la construcción de un prototipo y este es evaluado por el usuario o cliente. La interacción ocurre cuando el prototipo satisface las necesidades del cliente. (Maida & Pacienzia, 2015).

2.4.4. *Metodología de Desarrollo Rápido de Aplicaciones (RAD)*

La metodología de desarrollo Rápido de Aplicaciones, mejor conocida como RAD ha tomado gran auge debido a la necesidad que tienen las instituciones de crear aplicaciones funcionales en un plazo de tiempo corto. RAD es un ciclo de desarrollo diseñado para crear aplicaciones de computadoras de alta calidad. Este método trata de un enfoque que está destinado a proporcionar un excelente proceso de desarrollo con la ayuda de otros enfoques. Además, está diseñado para aumentar la viabilidad de todo el procedimiento de desarrollo de software para resaltar la participación de un usuario activo. (Maida & Pacienzia, 2015).

2.4.5. *Programación Extrema (XP)*

Es la metodología más destacada de las metodologías mencionadas anteriormente, principalmente por poner más énfasis en la adaptabilidad que en previsibilidad. Algunos defensores de este método afirman que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos. (Maida & Pacienzia, 2015).

2.4.6. *Scrum*

La metodología de trabajo de Scrum tiene sus principios fundamentales en la década de 1980, la cual fue desarrollada por su necesidad en procesos de reingeniería por Goldratt, Takeuchi y Nonaka. El concepto de Scrum tiene su origen sobre los nuevos procesos de desarrollo utilizados en productos exitosos en Japón y los Estados. Los equipos que desarrollaron estos productos partían de requisitos muy generales, así como novedosos, y debían salir al mercado en mucho menos del tiempo del que se tardó en lanzar productos anteriores. Estos equipos seguían patrones de ejecución de proyecto muy similares. En este estudio se comparaba la forma de trabajo de estos equipos altamente productivos y multidisciplinares con la colaboración entre los jugadores de Rugby y su formación de Scrum, de la cual se tomó su nombre. Scrum es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos. Podríamos decir que SCRUM se basa en cierto caos controlado, pero establece ciertos mecanismos para controlar esta indeterminación, manipular lo impredecible y controlar la flexibilidad. En Scrum se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales. (Maida & Pacienzia, 2015).

2.5. Sistema Control de Versiones

Un Sistema de Control de Versiones es un software que controla y organiza las distintas revisiones que se realicen sobre uno o varios documentos gestionando los diferentes estados por los que pasa una aplicación durante todo el periodo de desarrollo (Otero

Gutierrez, 2011), y que se utiliza principalmente para el desarrollo de proyectos de software, en donde se va guardando un historial con todos los cambios realizados entre versiones y que pueden ser recuperados en cualquier momento de la fase de desarrollo. Con esto se logra una ventaja productiva, debido a que pueden trabajar varios desarrolladores en el proyecto de forma paralela, y puedan publicar individualmente cambios para que el resto de los integrantes pueda tener acceso a la última versión disponible.

2.5.1. *Clasificación de los Sistemas de Control de Versiones*

Los sistemas de control de versiones se clasifican en dos tipos:

- Sistemas Centralizados
- Sistemas Distribuidos

2.5.1.1.Sistemas Centralizados

Los sistemas de control de versiones centralizados las diferentes versiones de un proyecto están almacenadas en un único repositorio de un servidor central. (Tello Leal, Sosa R., & Diego A., 2012). En estos tipos de sistemas el desarrollador trabaja con una copia del servidor, normalmente es la más actualizada. Luego el desarrollador puede realizar los cambios correspondientes a dicha copia para posteriormente cargarla de nuevo al servidor, en donde se alojan los cambios en el repositorio central e informar sobre los posibles errores que se hayan podido efectuar durante el proceso de compilación. Lo anterior lo podemos observar en la siguiente figura:

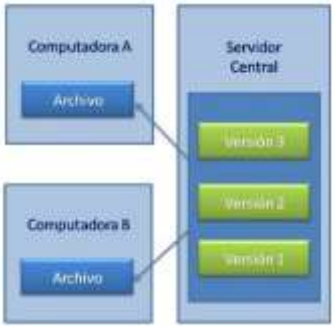

Figura 1 sistema de control centralizado

Las principales aplicaciones de tipo sistema de control centralizado son SVC (Concurrent Version System) y Subversion.

- **CVS (Concurrent Version System):** Es una herramienta que implementa un sistema de control de versiones, la cual mantiene un registro de todo el trabajo realizado y los cambios en ficheros independientes, permitiendo que varios desarrolladores situados en lugares diferentes puedan colaborar en el desarrollo. Fue por mucho tiempo a la principal herramienta utilizada en ambientes Open-Source, su primera versión fue liberada en 1986 hasta el año 2008. (Fundación Wikimedia, Inc, 2019)
- **Subversion:** Es un sistema de control de versiones de código abierto que sirve para guardar versiones de códigos fuentes, usado ampliamente en el desarrollo de software empresarial permitiendo que varios programadores trabajen al mismo tiempo con los mismos archivos permitiendo manejar múltiples versiones y recuperar versiones anteriores. (Tello Leal, Sosa R., & Diego A., 2012, págs. 3-4)

2.5.1.2.Sistemas Distribuidos

En el sistema de control de versiones distribuido cada desarrollador realiza una copia del repositorio de proyectos completo a su computadora, generando un repositorio local del proyecto. (Tello Leal, Sosa R., & Diego A., 2012). Lo anterior permitiendo que cada

desarrollador pueda trabajar de manera independiente y de manera local en cada máquina, en donde se pueden guardar todos los cambios realizados como una version que se alojarán en un repositorio local. Para que cuando el desarrollador lo considere necesario pueda sincronizar el repositorio local con el repositorio remoto, y finalmente el sistema de control de versiones unifica todos los repositorios cargados, de tal manera que quede una única copia que incluya todos los cambios realizados. Lo anterior como se puede apreciar en la figura 2.

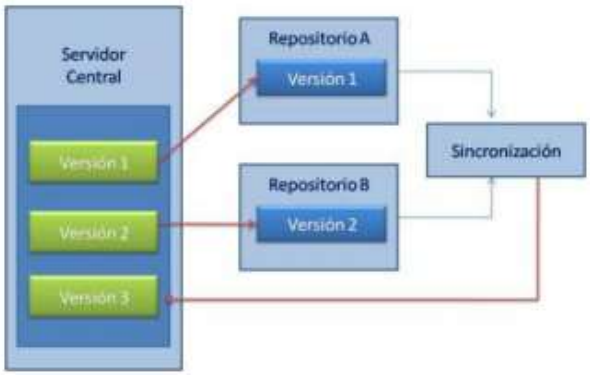

Figura 2 sistema de control de versiones distribuido.

Algunos de las principales herramientas de Sistema de Control de Versiones son GIT, Mercurial, BZR y BestKeeper.

- **GIT:** Es uno de sistemas de control de versiones más populares, de código abierto y gratuito creado en el año 2005 y es el que nos vamos a enfocar para el desarrollo del proyecto de investigación. Con esta herramienta podemos mantener un historial completo de versiones y podemos movernos entre cada uno de los cambios realizados en el código, tiene un sistema de trabajos con ramas que lo hace especialmente potente y rápida. Una de las características de GIT que lo distingue de casi todos los demás es su modelo de bifurcación. En el caso de esta funcionalidad están destinadas a provocar proyectos divergentes de un proyecto principal para hacer experimentos o probar

funcionalidades y poder tener un progreso diferente a nuestro proyecto principal y en algún momento se puede realizar una fusión. (git, 2019).

- **Mercurial:** El sistema de control de versiones Mercurial permite crear una copia de un repositorio, incluyendo todos los archivos del proyecto y su historial de versiones, esta copia de repositorio funciona de forma independiente y autónoma sin necesidad de acceso a la red o a un servidor. Una de las características más relevantes de la herramienta Mercurial es su funcionamiento en línea de comandos. (Tello Leal, Sosa R., & Diego A., 2012, pág. 6).

2.6.DevOps

El termino DevOps fue utilizado por primera vez en el año 2009 por Patrick Debois que es un consultor TI independiente, que asegura que utilizando las prácticas de DevOps las organizaciones pueden llegar a ser más competitivas y eficientes en sus negocios.

Len Bass, Ingo Weber y Liming Zhu definieron el término como:

"un conjunto de prácticas de desarrollo de software destinadas a reducir el tiempo entre los cambios en un sistema y el cambio se coloca en la producción normal, al tiempo que garantiza una alta calidad". (Bass, Weber, & Zhu, 2015).

2.7.Integración Continua (CI)

La integración continua es un modelo informático que fue propuesto inicialmente por Martin Fowler, el cual define el término de la siguiente manera:

Integración continua es una práctica de desarrollo de software en la que los miembros de un equipo integran su trabajo frecuentemente. Generalmente cada persona íntegra material al menos diariamente, dando lugar a múltiples integraciones por día. Cada integración es

verificada por un build automatizado (incluyendo pruebas) para detectar errores de integración lo más rápido posible. (Fowler, 2006).

Otra definición de este concepto la podemos encontrar en el autor Jhon Ferguson Smart que define a la integración continua de la siguiente manera:

La integración continua consiste en reducir el riesgo al proporcionar una retroalimentación más rápida. En primer lugar, está diseñado para ayudar a identificar y solucionar problemas de integración y regresión más rápido, lo que resulta en una entrega más suave y rápida, y menos errores. (Ferguson Smart, 2011).

Los conceptos de estos autores se encuentran estrechamente ligados, los dos tratan de describir el término como la principal base del desarrollo del software en la actualidad, debido a que antes de introducirse las prácticas de integración continua, los equipos de desarrollo tenían serios problemas con el exceso de tiempo y energía dedicados en la fase de integración, que es la fase anterior a la publicación del software, en el cual los desarrolladores individuales o pequeños grupos reunían los cambios realizados, para así, formar un producto funcional. Esta era una tarea algo complicada, por lo que estaba lleno de muchos riesgos en donde surgían errores que no podían ser anticipados y conllevaba a tener problemas tales como retrasos en la entrega del producto, costos no planificados y, por consiguiente insatisfacción de los clientes. Por los problemas anteriormente descritos es que nace la integración continua.

2.7.1. *Herramientas de Integración Continua.*

Hoy día se cuentan con varios frameworks que sirven para generar un flujo de integración continua.

2.7.1.1.Jenkins.

Es un servidor de integración continua de código abierto que se encuentra escrito en Java, utilizada por desarrolladores para automatizar las tareas que estén relacionadas con la

creación, prueba y entrega o implementación de software. (Creative Commons Attribution-ShareAlike 4.0, 2019).

2.7.1.2.Buildbot.

Es un sistema de planificación de trabajos cuya función es poner en cola las tareas y ejecutarlos cuando los recursos necesarios estén disponibles y dar un informe acerca de los resultados. El Buildbot consta de un único buildmaster y uno o más trabajadores, conectados en una topología en estrella. El Buildmaster es configurado y mantenido generalmente es el miembro del equipo del proyecto responsable de los problemas del proceso de construcción. (GNU Public License, 2019).

2.7.1.3.Travis CI.

Es un servicio de integración continúa utilizado para construir y probar proyectos de software alojados en GitHub. Travis CI respalda el proceso de desarrollo mediante la creación y prueba automáticas de los cambios de código, proporcionando información inmediata sobre el éxito del cambio. Travis CI también puede automatizar otras partes del proceso de desarrollo mediante la administración de implementaciones y notificaciones. (©TRAVIS CI, GMBH, 2019).

2.8. Entorno de Desarrollo Integrado (IDE)

Un entorno de desarrollo integrado (IDE) es una aplicación que se utiliza para crear software. Un IDE a menudo puede soportar diferentes lenguajes de programación. Los IDE tienen una serie de herramientas y funciones diferentes que ayudan al desarrollador a crear software. Incluyen también un ambiente en tiempo de ejecución que permite ejecutar el programa paso a paso que sirve para probar el código líneo por línea antes de crear el programa final. (BBC Bitesize, s.f.).

2.9. Sistema de Control de Versiones (SCV)

Un sistema de control de versiones es un software encargado de gestionar las diferentes modificaciones realizadas en un proyecto de desarrollo, llevando un registro sistemático de todos los avances realizados en el proyecto, conservando una copia de respaldo y con esto garantizar una integración del trabajo realizado por todos los miembros del equipo de desarrollo. (Trujillo Silva & Chavez Baquero, 2016).

2.10. Revisión automática de código

El software de revisión de código automatizado verifica el código fuente para el cumplimiento de un conjunto predefinido de reglas o mejores prácticas. El uso de métodos analíticos para inspeccionar y revisar el código fuente para detectar errores ha sido una práctica de desarrollo estándar. Este proceso puede llevarse a cabo tanto de forma manual como automatizada. Con la automatización, las herramientas de software brindan asistencia con la revisión del código y el proceso de inspección. El programa o herramienta de revisión generalmente muestra una lista de advertencias (violaciones de los estándares de programación). Un programa de revisión también puede proporcionar una forma automatizada o asistida por un programador para corregir los problemas encontrados. Este es un componente para dominar fácilmente el software. Esto está contribuyendo a la práctica de Software Intelligence.

2.11. Herramientas de revisión automática de código

2.11.1. SonarQube.

SonarQube proporciona una descripción general del estado general del código fuente y, lo que es más importante, resalta los problemas encontrados en el nuevo código. Con una puerta de calidad configurada del proyecto, simplemente repara la fuga y comenzará a

mejorar mecánicamente. Las integraciones nativas de SonarQube permiten programar fácilmente la ejecución de un análisis desde todos los motores de CI.

2.11.2. Codacy.

Codacy es una herramienta de revisión de código automatizada para Scala, Java, Ruby, JavaScript, PHP, Python, CoffeeScript y CSS. Es un análisis estático continuo sin la molestia. Ahorre tiempo en las revisiones de código. Afronte su deuda técnica.

2.11.3. Code Climate.

Code Climate analiza su código en busca de complejidad, duplicación y errores comunes para determinar los cambios en la calidad y los puntos críticos técnicos de la deuda.

3. Diseño técnico y metodológico

El desarrollo del proyecto será dividido en tres fases:

1. ***Análisis de las herramientas de las tecnologías de la información utilizadas en el pensum del programa de Ingeniería de Sistemas:*** El objetivo es identificar las herramientas tecnologías utilizadas, su propósito dentro del pensum y adaptabilidad al sistema de Integración Continua.

2. ***Diseñar una infraestructura de TI acorde a las necesidades del programa de ingeniería de sistema***: En esta fase se realiza un diseño arquitectónico con base en las necesidades del programa a nivel de herramientas tecnológicas.

3. ***Implementar un prototipo de la infraestructura diseñada:*** En esta etapa se validará el diseño a través de la implementación de un prototipo que permita simular el ambiente de trabajo bajo una integración continua.

4. Desarrollo del prototipo de CI

Siguiendo la metodología planteada se desarrolló el prototipo del Sistema de Integración Continua.

4.1. Análisis de las herramientas de las tecnologías de la información utilizadas en el pensum del programa de ingeniería de sistemas

El propósito de formación de competencias establecidas en el Proyecto Educativo del Programa (PEP) de Ingeniería de Sistemas de CECAR, propone el desarrollo de aplicativos de software que permitan satisfacer las necesidades específicas de organizaciones del entorno, para el adecuado tratamiento de la información, teniendo en cuenta la utilización de metodologías, herramientas y estándares de calidad de desarrollo de software. El plan de estudios del programa de ingeniería de sistema tiene un total de diez (10) semestres académicos, conformado por sesenta y cuatro (64) asignaturas que suman un total de ciento cincuenta y siete (157) créditos académicos, los cuales están organizados por semestre de la siguiente manera:

1. PRIMER SEMESTRE	
ASIGNATURAS	**CRÉDITOS**
Introducción a la Ingeniería de Sistemas	5
Taller de Lengua I	3
Técnica de Aprendizaje	2
Inglés I	3
Matemáticas	2
Vida Universitaria	1
Programación de Computadores	5
Total créditos	21

2. SEGUNDO SEMESTRE	
ASIGNATURAS	**CRÉDITOS**
Algebra Lineal	4
Taller de Lengua II	3
Optativa I	2
Inglés II	2
Lógica Teórica	2
Cálculo Diferencial	4
Programación de Computadores Avanzada	5
Total créditos	22

3. TERCER SEMESTRE	
ASIGNATURAS	**CRÉDITOS**
Constitución Política y Democracia	2
Optativa II	2
Inglés III	3
Cálculo Integral	4
Mecánica	5
Programación y Tecnologías Web	4
Arquitectura y Modelamiento de Datos	3
Total créditos	23

04. CUARTO SEMESTRE	
ASIGNATURAS	**CRÉDITOS**
Bases de Datos	5
Ecuaciones Diferenciales	4
Ética Profesional	0
Estadística Descriptiva	4
Electromagnetismo	5
Programación y Tecnologías Web II	4
Total créditos	22

05. QUINTO SEMESTRE	
ASIGNATURAS	CRÉDITOS
Arquitectura de los Computadores	3
Estadística Inferencial	4
Ingeniería Económica	3
Programación y Tecnología Móvil	5
Estructura de Datos y Análisis de Algoritm	4
Análisis de Técnicas Numéricas	3
Total créditos	23

06. SEXTO SEMESTRE	
ASIGNATURAS	CRÉDITOS
Electrónica Básica	4
Sistemas Operativos	4
Proyectos de Ingeniería	2
Redes de Computadores I	4
Modelamiento de Sistemas Dinámicos	4
Ingeniería del Software y Requerimientos	3
Total créditos	21

07. SÉPTIMO SEMESTRE	
ASIGNATURAS	**CRÉDITOS**
Sistemas Digitales	7
Espíritu Emprendedor	2
Investigación de Operaciones	4
Gestión de Proyectos de Ingeniería de So	4
Infraestructura Tecnológica de TI	3
Redes De Computadores II	4
Arquitectura Empresarial De TI	3
Total créditos	27

08. OCTAVO SEMESTRE	
ASIGNATURAS	**CRÉDITOS**
Innovación Creatividad	2
Simulación de Sistemas	4
Sistemas Distribuidos	4
Fundamentos y Contexto Para La Investigac	3
E-Business y Contexto	4
Electrónica Programable	4
Seguridad Informática	3
Total créditos	24

09. NOVENO SEMESTRE	
ASIGNATURAS	**CRÉDITOS**
Optativa III	2
Enfoques y Herramientas de la Investigac	0
Electiva Profesional en Ingeniería del Sof	4
Tecnologías E-Business	4
Tendencias Tecnológicas en TI / SI	4
Electiva Profesional en Ciencias de la Cor	4
Total créditos	18

10. DÉCIMO SEMESTRE	
ASIGNATURAS	**CRÉDITOS**
Práctica Profesional	5
Investigación y Emprendimiento	3
Tendencias Tecnológicas en Ing. del Softwar	4
Electiva Profesional en Ti/Si	4
Integración de Soluciones Tecnológicas	4
Total créditos	20

Figura 3 plan de estudios ingeniería de sistemas.

Fuente: Elaboración propia

Para el análisis de las herramientas de las Tecnologías de la Información utilizadas en el pensum académico se tuvieron en cuenta:

- ✓ Sistemas Operativos
- ✓ Framework
- ✓ Lenguajes de programación
- ✓ IDE (Integrated Development Enviroment)
- ✓ Motores de bases de datos.
- ✓ Test automáticos
- ✓ Sistema de control de versiones

Seguidamente, se analizaron los planes de aula de las diferentes asignaturas para determinar las tecnologías de la información usadas en cada una de ellas y se obtuvo el siguiente cuadro:

Tabla 1

Herramientas de TI utilizadas en el programa de ingeniería de sistemas.

Tipo de Herramienta TI	**Producto Tecnológico**	**Propósito**
IDE (Integrated Development Enviroment)	✓ Netbeans ✓ Eclipse	Facilita el desarrollo de aplicaciones. El estudiante tiene la libertad de usar el que desee.
Test Automáticos	✓ Junit	Se utiliza de JUnit para realizar las pruebas con

		Java. No se evidencia para otros lenguajes.
Sistema de control de versiones	✓ Git	Para el sistema de control de versiones se utiliza Git, utilizando servicio como el GitHub y GitLab.

Fuente: Elaboración propia

4.2.Diseño de la solución de un ambiente de integración continúa

Para el diseño de la solución se tuvieron en cuenta dos ambientes, el primero Fig 3, los servicios se mantendrán en la nube, para lo cual el estudiante accede con unas credenciales y el sistema devuelve el resultado del análisis, para ser evaluada o y refactorizado el código.

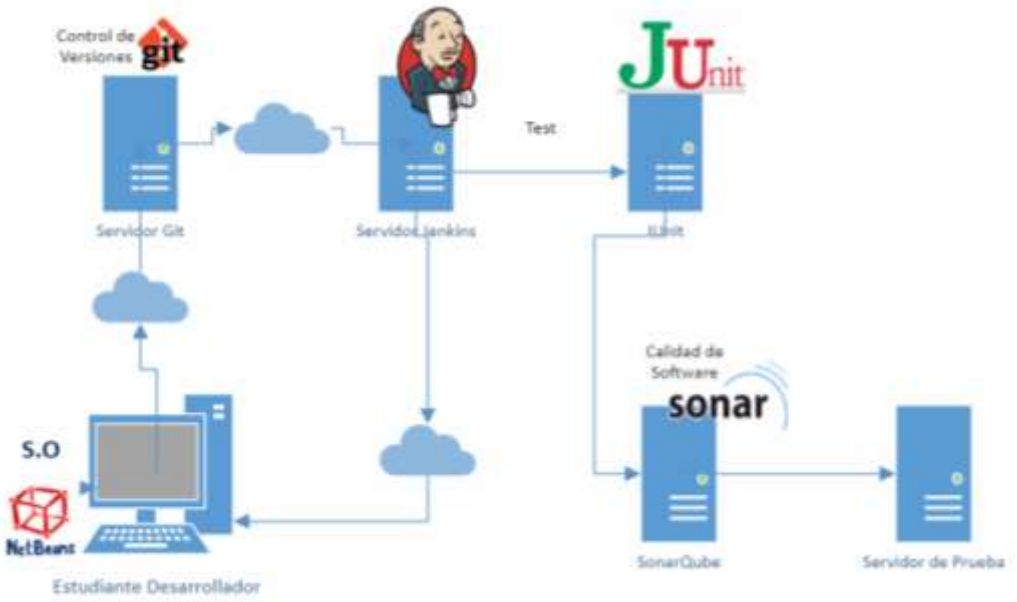

Figura 4 Diseño de integración continúa en la nube.

Fuente: Elaboración propia

Para el segundo ambiente, se desplegará toda la infraestructura de forma local, es decir en cada máquina asignada al estudiante para sus prácticas de desarrollo de software. Lo anterior se realizará a través de una máquina virtual donde se tienen los servicios de integración continua (Ver Fig 4).

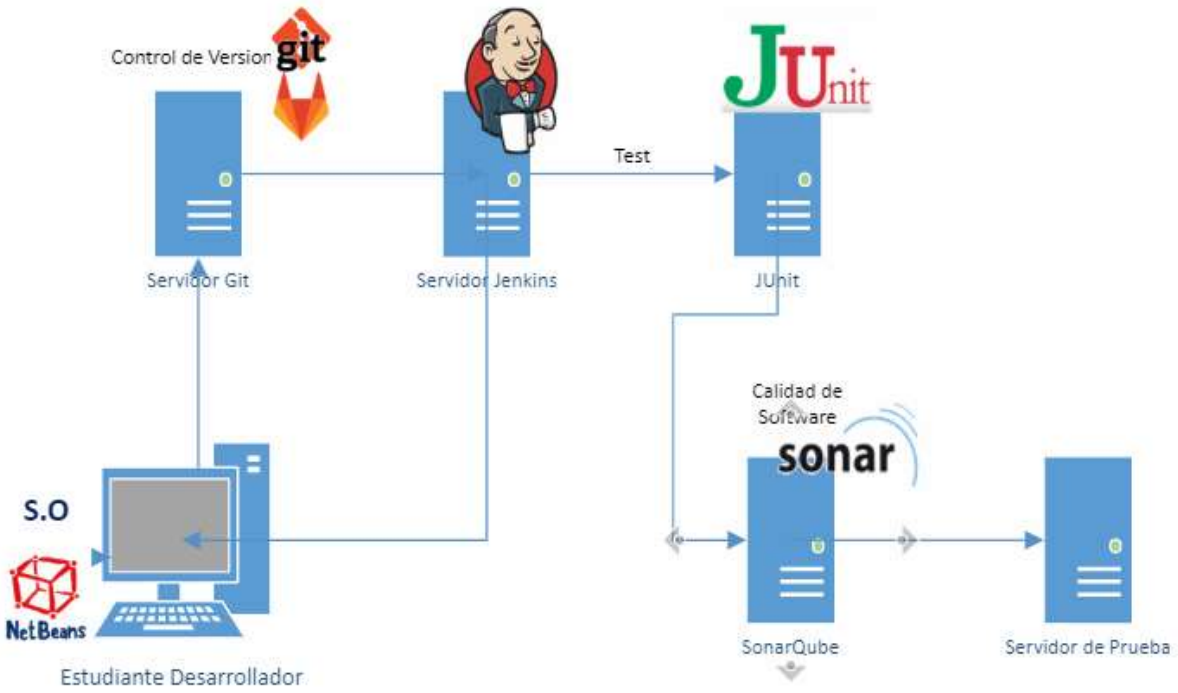

Figura 5 Diseño de integración continúa local.

Fuente: Elaboración propia

Los dos diseños contemplan las mismas tecnologías la diferencia fundamental es donde se desplegarán los servicios, el primero en la nube y la segunda de forma local. Por ende, las funciones de los servicios serán los siguientes:

- **Maven:** Se utilizará Maven como gestor de construcción integrado al IDE
- **IDE:** Con relación al IDE los estudiantes pueden utilizar cualquiera de su preferencia (NetBeans, Eclipse, IntellIdea).

- **Servidor de control versiones:** Se propone la utilización de Git por su alta aceptación en el mercado.
- **JUnit:** Como API para las pruebas, para este caso Java, proponemos JUnit
- **SonarQube:** Servidor para procesos de calidad de software
- **Jacoco:** Servidor de cobertura para las pruebas automatizadas

4.3. Implementación y validación del prototipo diseñado de Integración Continua

Para la implementación del prototipo se tomó como referencia el ambiente 2 diseñado, es decir desplegaremos los servició a nivel local. Es importante anotar que para el despliegue en la nube solo se debe cambiar la forma de acceder.

4.3.1. Creación de un repositorio de código

Se procede con la creación de una cuenta en GitHub (repositorio GIT).

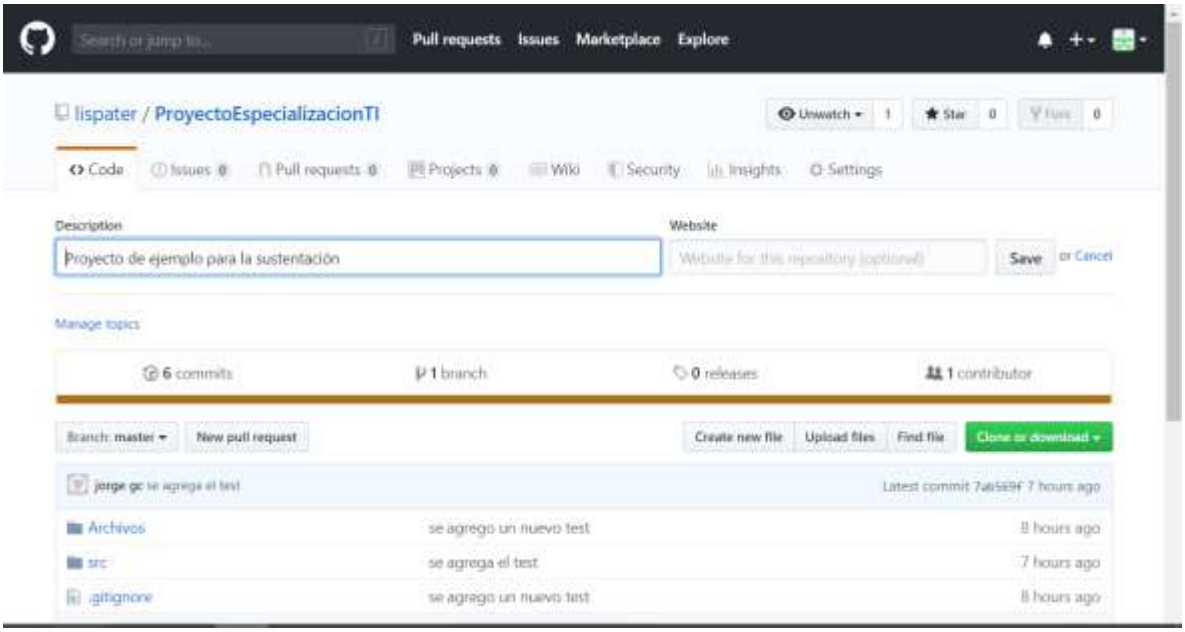

Figura 6 página principal de github.

Fuente: Elaboración propia

4.3.2. Servidor de Integración Continua

Se selecciono Jenkins como servidor de Integración Continua debido principalmente a su naturaleza de código abierto escrito en java, su versatilidad y por ser la herramienta más utilizada para gestionar construcciones de integración continua y canalizaciones de entrega.

Inicialmente se realiza el montaje y configuración del servidor Jenkins de Integración continua (CI) en la máquina virtual. Posteriormente a través de un complemento especial se establece la conexión de Jenkins con el repositorio remoto de GitHub, se suministran los datos de la cuenta usuario y contraseña.

Figura 7 Página principal de Jenkins.

Fuente: Elaboración propia

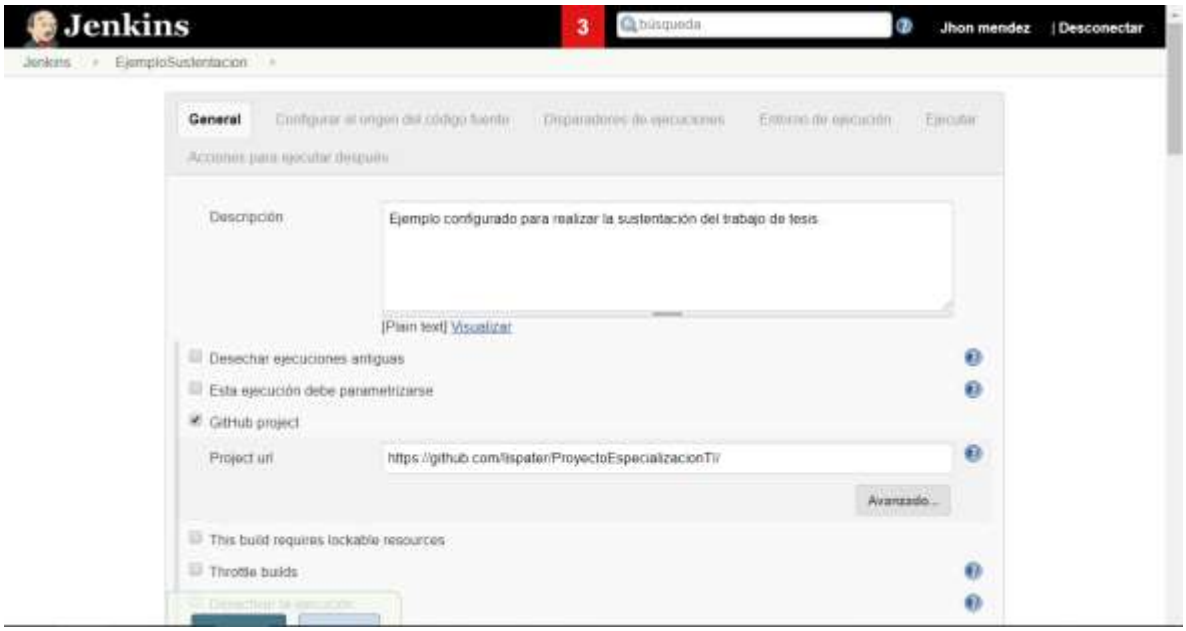

Figura 8 Pestaña general de la configuración de Jenkins.

Fuente: Elaboración propia

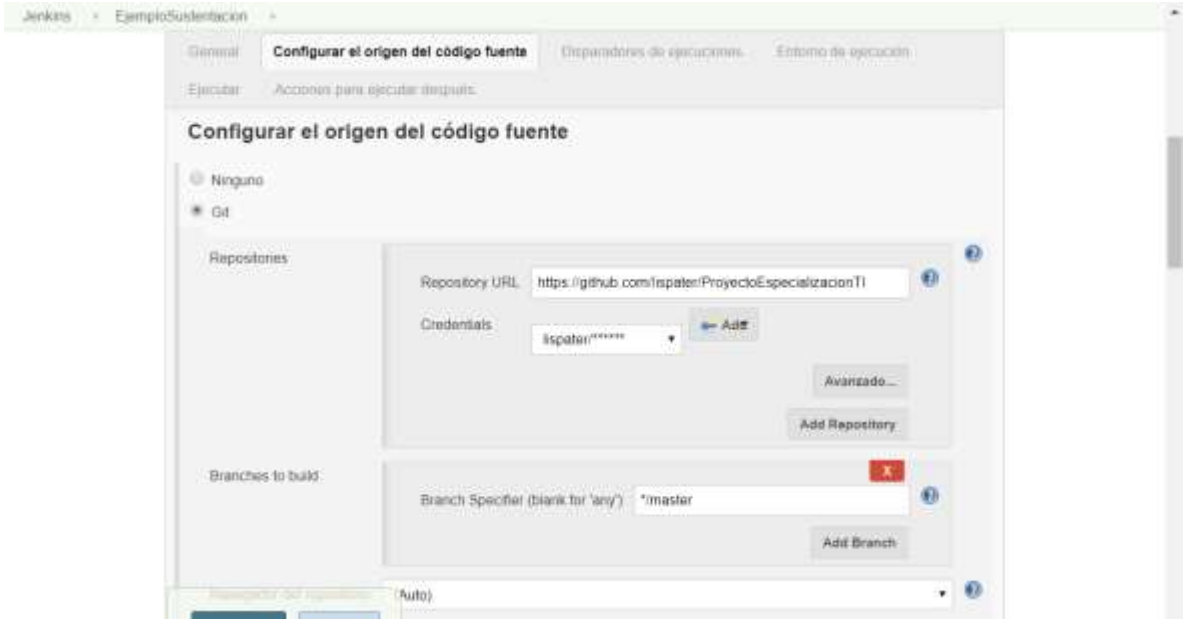

Figura 9 Configurar el origen del código fuente.

Fuente: Elaboración propia

A través de la API de GitHub, Jenkins se encarga de analizar el repositorio:

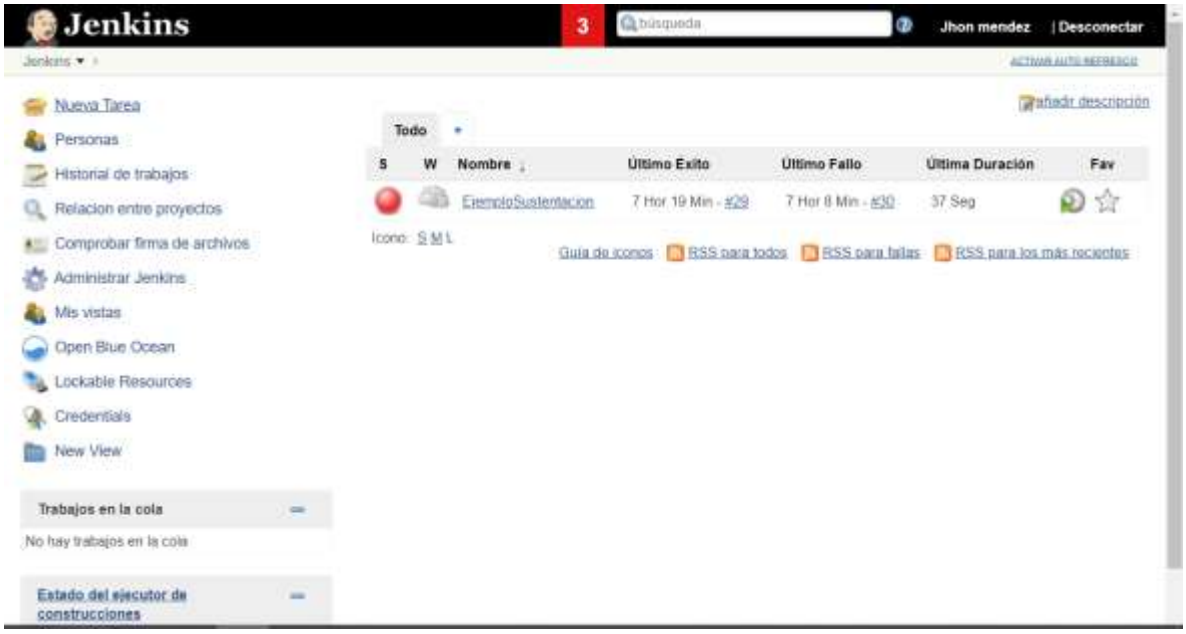

Figura 10 Ventana de inicio de Jenkins.

Fuente: Elaboración propia

4.3.3. Creación de un sistema de construcción automatizado

Se encontraron varios sistemas de gestión de proyectos para Java, pero basándose en la antigüedad de la herramienta en el mercado y una comunidad activa se seleccionó Maven y se realizó la configuración en Jenkins.

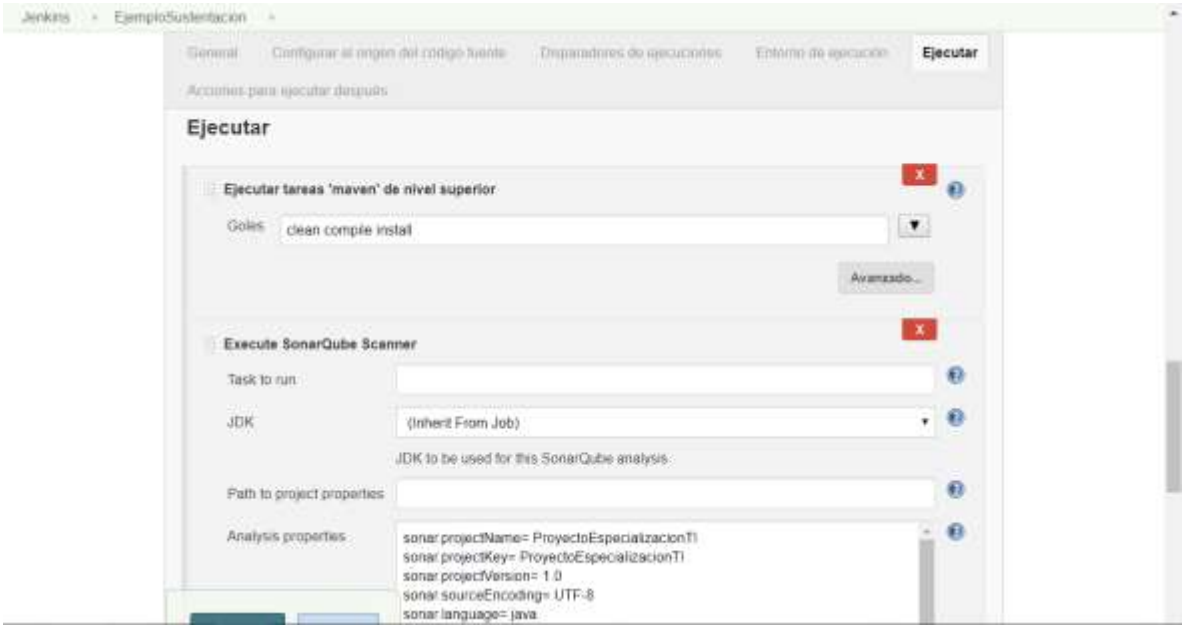

Figura 11 Pestaña Ejecutar de Jenkins.

Fuente: Elaboración propia

4.3.4. Creación de Commits

El objetivo principal de la Integración Continua es la incorporación de cambios lo más frecuente posible, para identificar y resolver los problemas apresuradamente. Por esto la importancia de la actualización de cambios del código en el repositorio local diariamente.

En el IDE de NetBeans se procedió a configurar la conexión con el repositorio remoto GitHub y se realizó el ejercicio con un proyecto desarrollado en lenguaje Java.

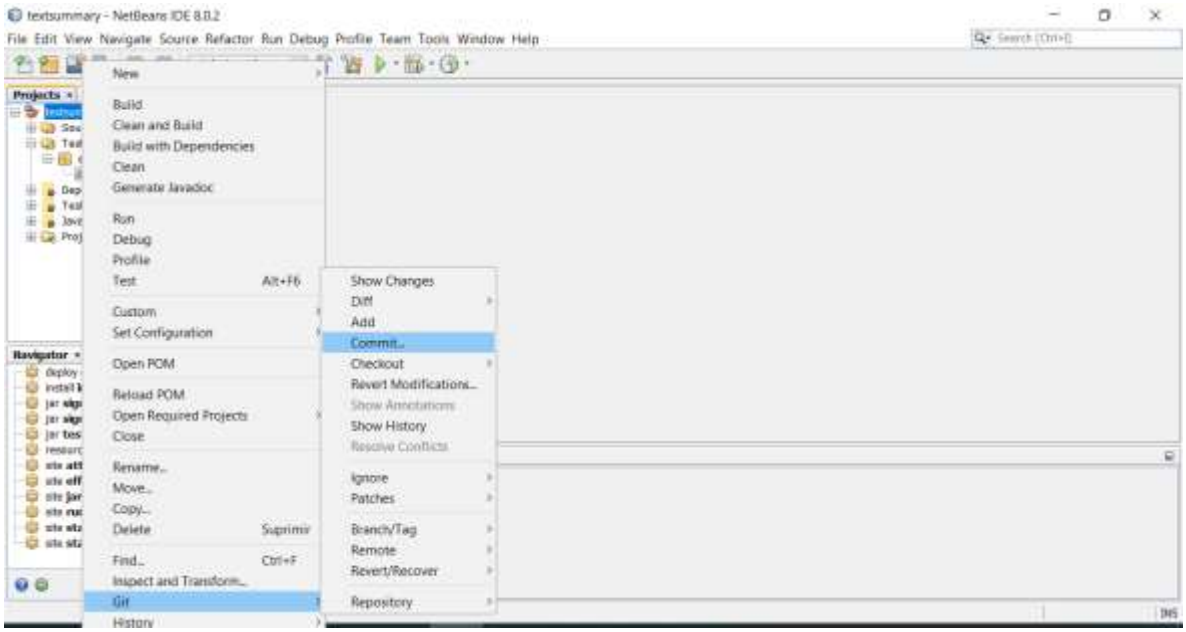

Figura 12 Inserción de un Cambio en el Repositorio.

Fuente: Elaboración propia

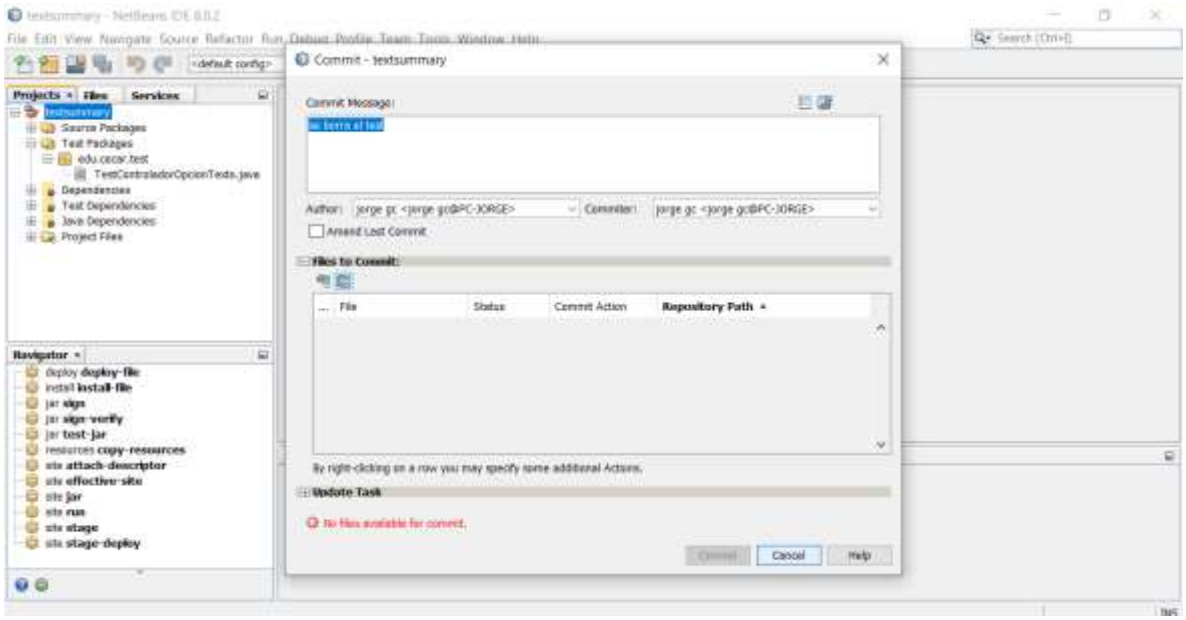

Figura 13 Ventana de Commit.

Fuente: Elaboración propia

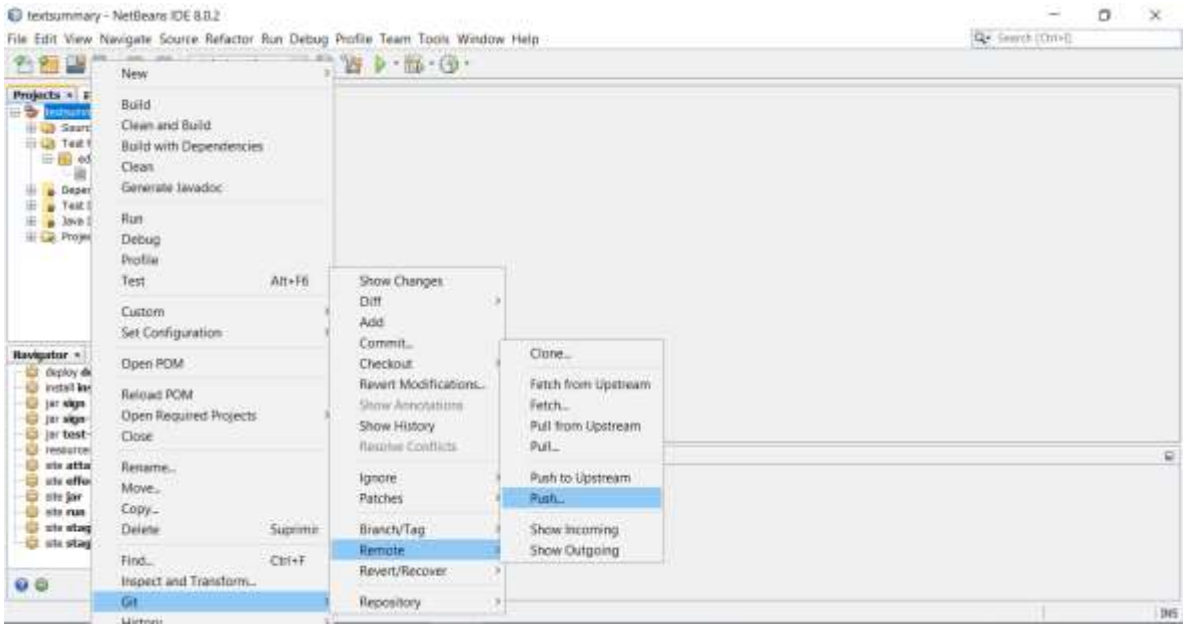

Figura 14 Sincronizar un cambio.

Fuente: Elaboración propia

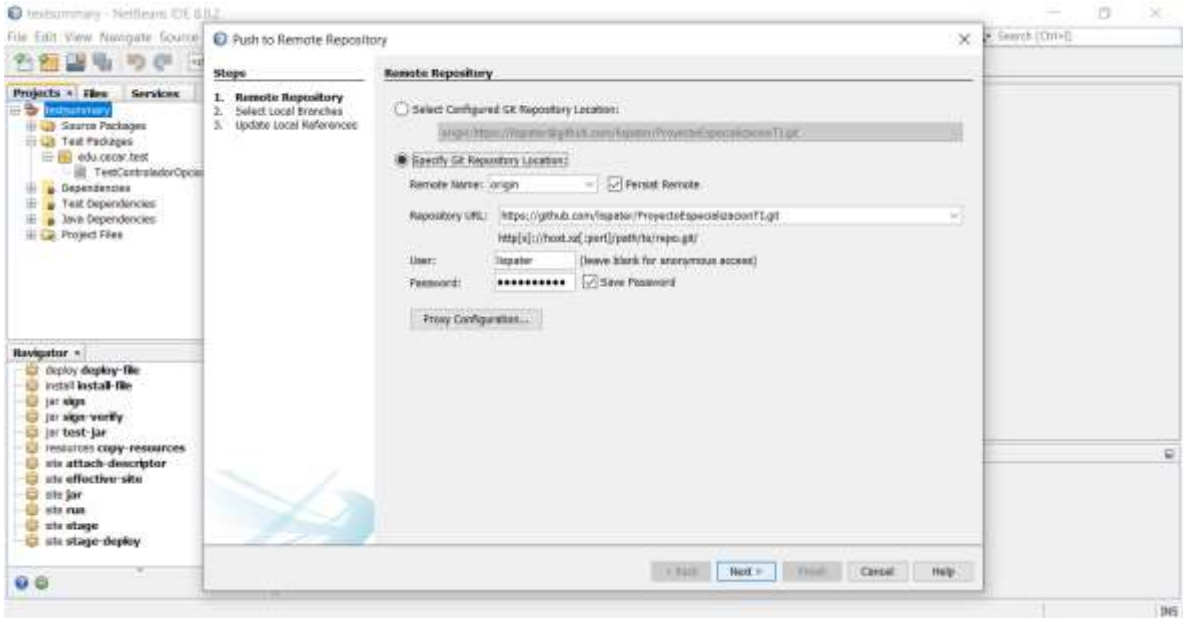

Figura 15 Ventana de Repositorio Remoto de NetBeans.

Fuente: Elaboración propia

Cada *commit* realizado se actualiza en el repositorio remoto de GitHub:

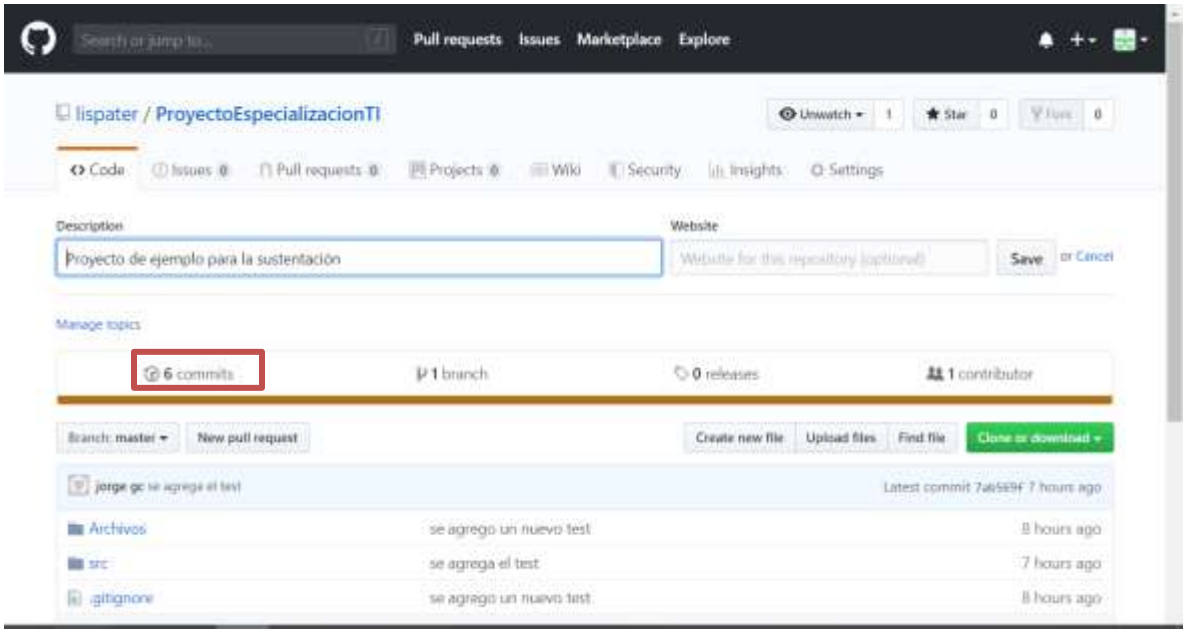

Figura 16 Cambio reflejado en plataforma Github.

Fuente: Elaboración propia

4.3.5. Configuración de Pruebas unitarias

Se configura el framework JUnit para el desarrollo de las pruebas unitarias en Jenkins.

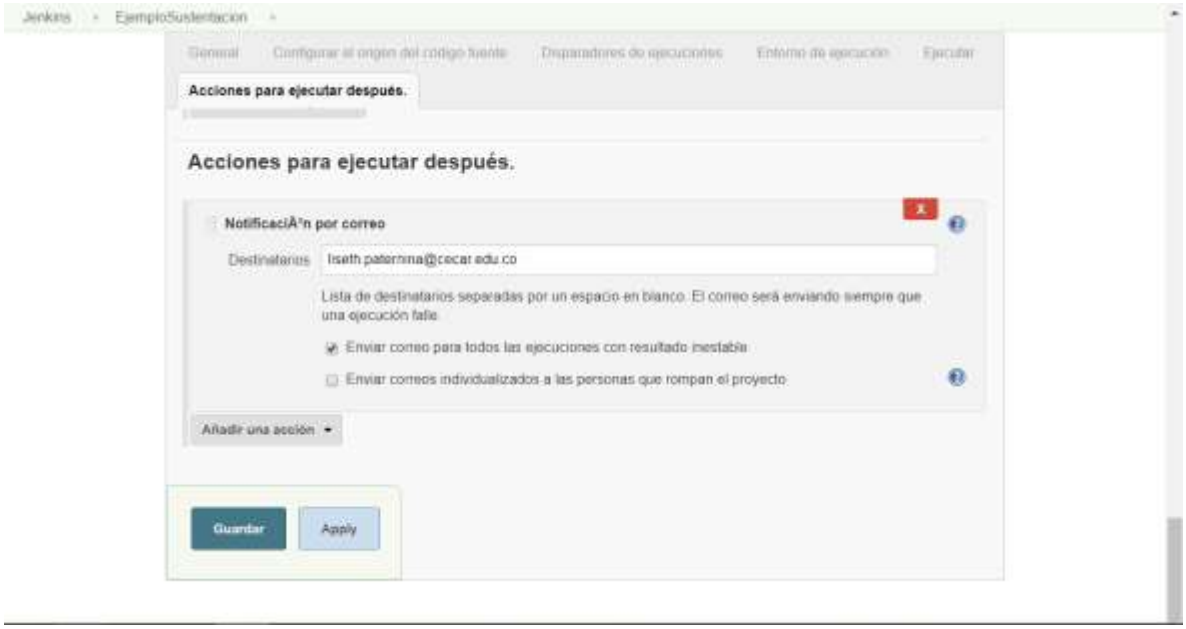

Figura 17 Configuración de Correo Electrónico.

Fuente: Elaboración propia

4.3.6. Generación de pipeline de trabajo

La importancia de mantener un código confiable implica incluir diferentes tipos de pruebas en los diferentes entornos del proyecto, a parte de las pruebas unitarias, con el fin de minimizar los problemas y facilitar la corrección de errores presentados, siendo el objetivo de esta etapa analizar y definir el cumplimiento mínimo de criterios de calidad y buenas prácticas en el desarrollo de software. Por lo anterior se utilizó SonarQube que es un servidor que se dedica al análisis del código alojado en el repositorio remoto, que vela por el cumplimiento de buenas normas de programación. A este conjunto de pruebas que define la estructura del flujo de ejecución de trabajos, se le conoce como el pipeline en Jenkins.

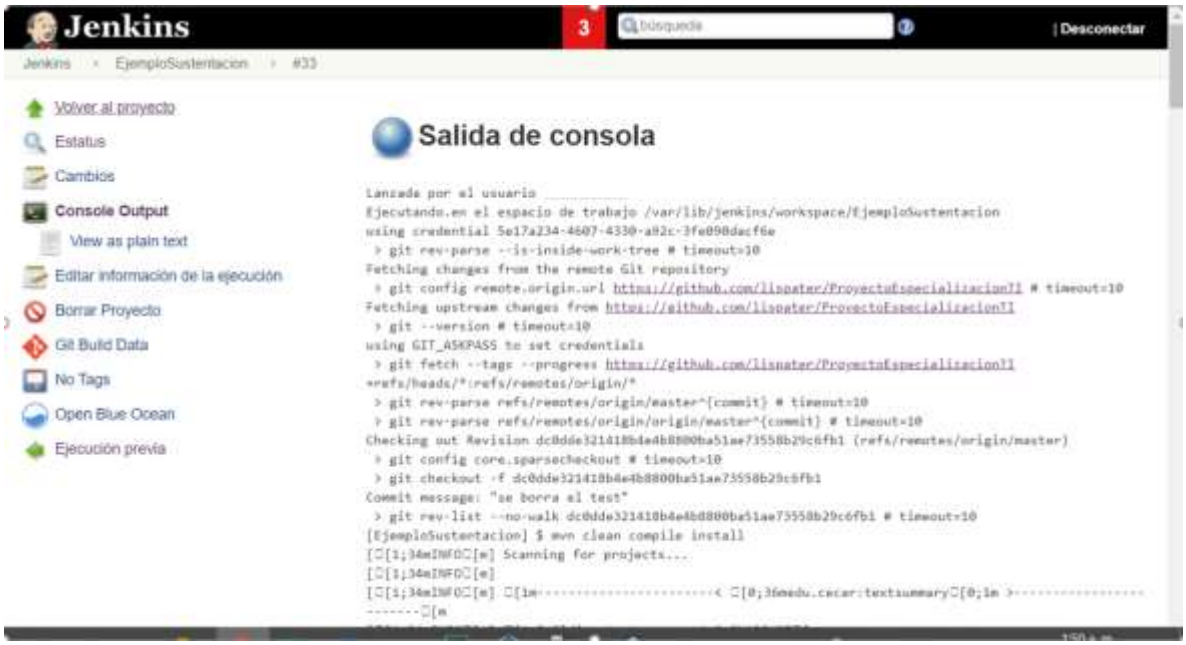

Figura 18 Salida de Consola 1.

Figura 19 Salida de Consola 2.

Fuente: Elaboración propia

Cuando la prueba es exitosa, podremos consultar en SonarQube los detalles del análisis realizado y las respectivas recomendaciones generadas.

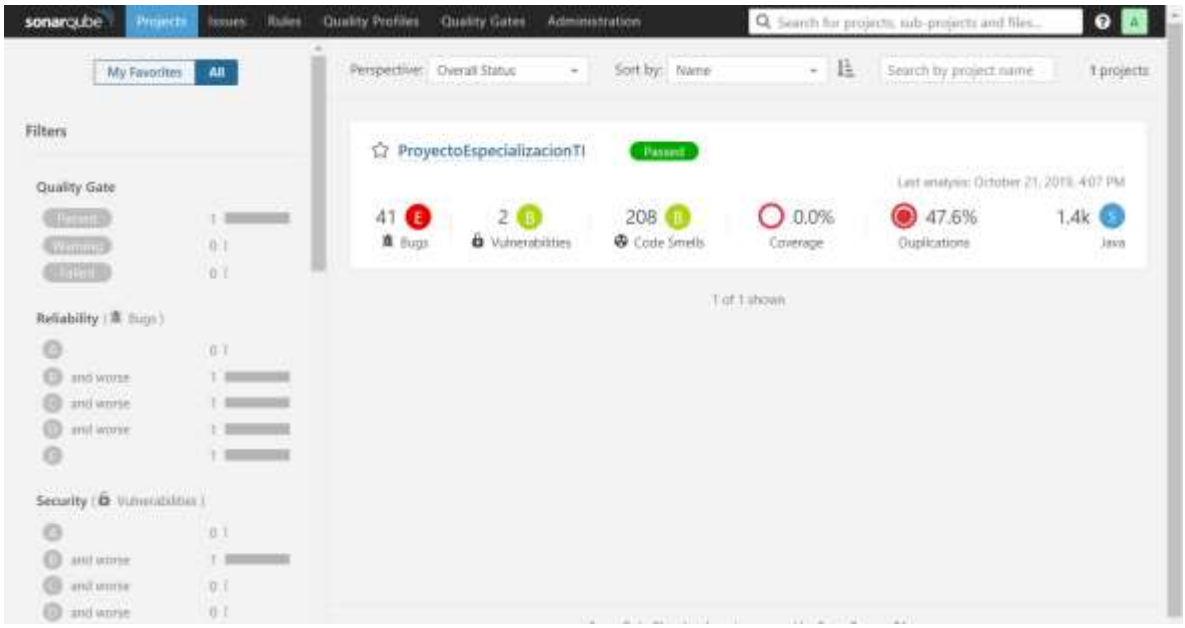

Figura 20 Plataforma SonarQube.

Fuente: Elaboración propia

Si la prueba no pasa los criterios mínimos asignados por el desarrollador, el sistema marcará en rojo el estado de la prueba en el panel general y Jenkins envía una notificación al correo electrónico previamente configurado con los detalles de las fallas encontradas, para su respectiva verificación y corrección.

La ejecuciÃ³n en Jenkins ha fallado: EjemploSustentacion #27 Recibidos ×

Dirección no configurada todavía lun., 21 oct. 15:24 (hace 22 horas)
para mí

inglés › español Traducir mensaje Desactivar para: inglés ×

Echa un vistazo a <http://127.0.0.1:8080/job/EjemploSustentacion/27/display/redirect?page=changes{1}>

Cambios:

[jorge gc] se agrego test

Lanzada por el usuario
Ejecutando.en el espacio de trabajo <http://127.0.0.1:8080/job/EjemploSustentacion/ws/>
using credential 5e17a234-4607-4330-a92c-3fe090dacf6e
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/lispater/ProyectoEspecializacionTI # timeout=10
Fetching upstream changes from https://github.com/lispater/ProyectoEspecializacionTI
> git --version # timeout=10
using GIT_ASKPASS to set credentials
> git fetch --tags --progress https://github.com/lispater/ProyectoEspecializacionTI +refs/heads/*:refs/remotes/origin/*
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 3b720089599c3a8af94d67ed616b1b2d97de91c6 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 3b720089599c3a8af94d67ed616b1b2d97de91c6

Figura 21 Prueba de fallo enviada desde Jenkins.

Fuente: Elaboración propia

Este proyecto tiene gran importancia, debido a que se enfoca en el diseño de una infraestructura que ayudará a que el programa de ingeniería de sistemas de la corporación universitaria del caribe CECAR adopte tecnologías de la información que han sido reconocidas mundialmente como tecnologías eficientes para el desarrollo de software.

A sus docentes y estudiantes a resolver diversos problemas que se vienen presentando en las asignaturas relacionadas con el desarrollo de software, tales como pérdida de tiempo al momento de realizar la integración de sus proyectos de desarrollo, presentación de errores a última hora, omisión de requerimientos no funcionales.

Conclusiones

A continuación, se describen las conclusiones producto del desarrollo del proyecto:

- Las utilizaciones de sistemas de integración continua a nivel de pregrado permitirán a los estudiantes mejorar los tiempos de desarrollo y calidad de software debido a que les automatizara algunas tareas que las realizaban de manera manual para el control de versiones, testing y cumplimiento de métricas.

- El egresado de ingeniería de sistemas de CECAR será más competitivo al comprender y usar herramientas de integración continua para fábricas de software que tengan implantado estos sistemas.

- El sistema de integración continua Jenkins automatiza de manera sencilla las tareas revisión de repositorios, ejecución de pruebas, envió de mensajes de error o mejoras en la calidad de software a cada grupo de trabajo. Asi mismo Jenkins puede automatizar cualquier tipo de tareas no solo de desarrollo de software sino también de administración de sistemas.

Recomendaciones

- Implementar el Sistema de Integración continua desde sexto semestre de programa de Ingeniería de Sistemas en donde el estudiante cuenta con los conocimientos fundamentales para utilizar estas herramientas.

- Migrar el sistema Ubuntu Desktop a Sistemas Operativos Servidores tales como: Ubuntu server, Centos o Windows Server, entre otros.

- Adicionar un servidor de pruebas que esté conectado a internet para que los estudiantes visualicen sus productos de software.

Referencias bibliográficas

Creative Commons Attribution-ShareAlike 4.0. (2019). *Jenkins*. Recuperado de https://jenkins.io/

©TRAVIS CI, GMBH. (2019). *Travis CI*. Recuperado de https://travis-ci.com/

Amazon Web Services. (2018). *AWS*. Recuperado de https://aws.amazon.com/es/devops/continuous-integration/

Bass, L., Weber, I., & Zhu, L. (2015). *DevOps: La perspectiva de un arquitecto de software.*

BBC Bitesize. (s.f.). *BBC*. Recuperado el abril de 2019, de https://www.bbc.com/bitesize/guides/zgmpr82/revision/1

Corporacion Universitaria del Caribe CECAR. (2018). Recuperado de https://cecar.edu.co/

Ferguson Smart, J. (2011). *Jenkins: The Definitive Guide: Continuos Integration for the Masses.* California: O'Reilly Media.

Fowler, M. (1 de Mayo de 2006). *MARTINFOWLER.COM.* Recuperado de https://martinfowler.com/articles/continuousIntegration.html

Fundación Wikimedia, Inc. (2019). *Wikipedia, La enciclopedia libre*. Recuperado de https://es.wikipedia.org/wiki/CVS

García Oterino, A. M. (08 de Agosto de 2014). *javiergarzas.com.* Recuperado de http://www.javiergarzas.com/2014/08/implantar-integracion-continua.html

git . (2019). *git--distributed-is-the-new-centralized*.

GNU Public License. (2019). *Buildbot the Continuos Integration Framework*. Recuperado de https://buildbot.net/

IEEE. (1990).

Laboratorio Nacional de Calidad del Software de INTECO. (2009). *Curso de Introducción a la Ingeniería del Software.* España: Creative Commons.

Maida, E., & Pacienzia, J. (2015). *Metodologías de Desarrollo de Software.* pontificia universidad católica argentina santa maría de los buenos aires (Tesis de Pregrado), Buenos Aires.

Otero Gutierrez, D. (2011). *Desarrollo de una aplicación web para control de versiones de software (Tesis de Pregrado).* Universidad Carlos III de Madrid, Madrid.

Ruedas, J. G. (21 de Agosto de 2013). *Instituto Español de Estudios Estratégicos.* Obtenido de http://www.ieee.es/Galerias/fichero/docs_marco/2013/DIEEEM16-2013_TIC_JGomezRueda.pdf

Tello Leal, E., Sosa R., C., & Diego A., T.-L. (2012). REVISIÓN DE LOS SISTEMAS DE CONTROL DE VERSIONES UTILIZADOS. *Revista Ingenierías UsbMed*, 74-81.

Trujillo Silva, D. M., & Chavez Baquero, A. (2016). *Sistema de Control de Versiones para el Desarrollo de Software Seguro (Pasantia Investigativa).* Bogotá, Colombia.

WIKIPEDIA, La enciclopedia libre. (2018). Recuperado de https://es.wikipedia.org/wiki/Integraci%C3%B3n_continua.

Anexos

1.1.Instalación GIT

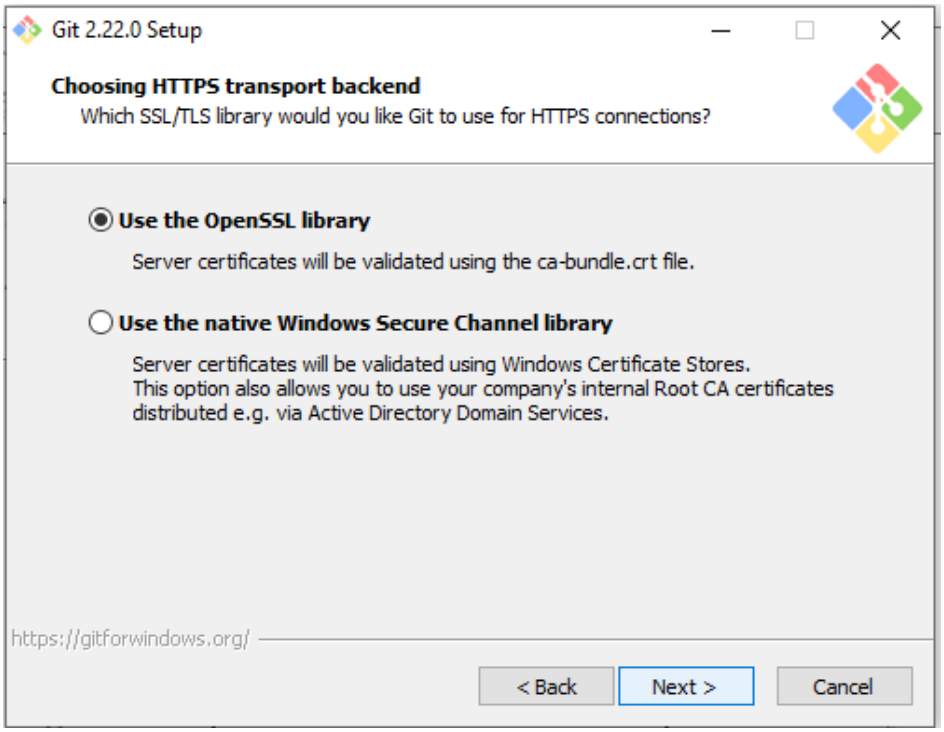

Figura 22 Inicio de instalación Software Git 2.22.0.

Fuente: Elaboración propia

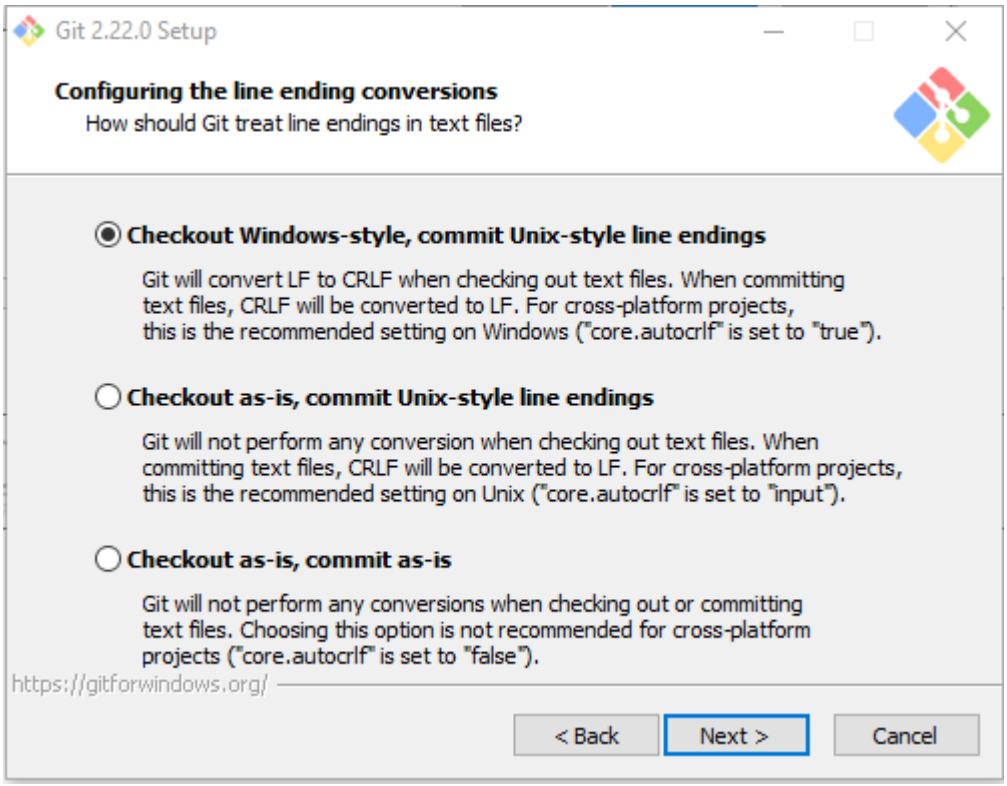

Figura 23 Configuración de GIT.

Fuente: Elaboración propia

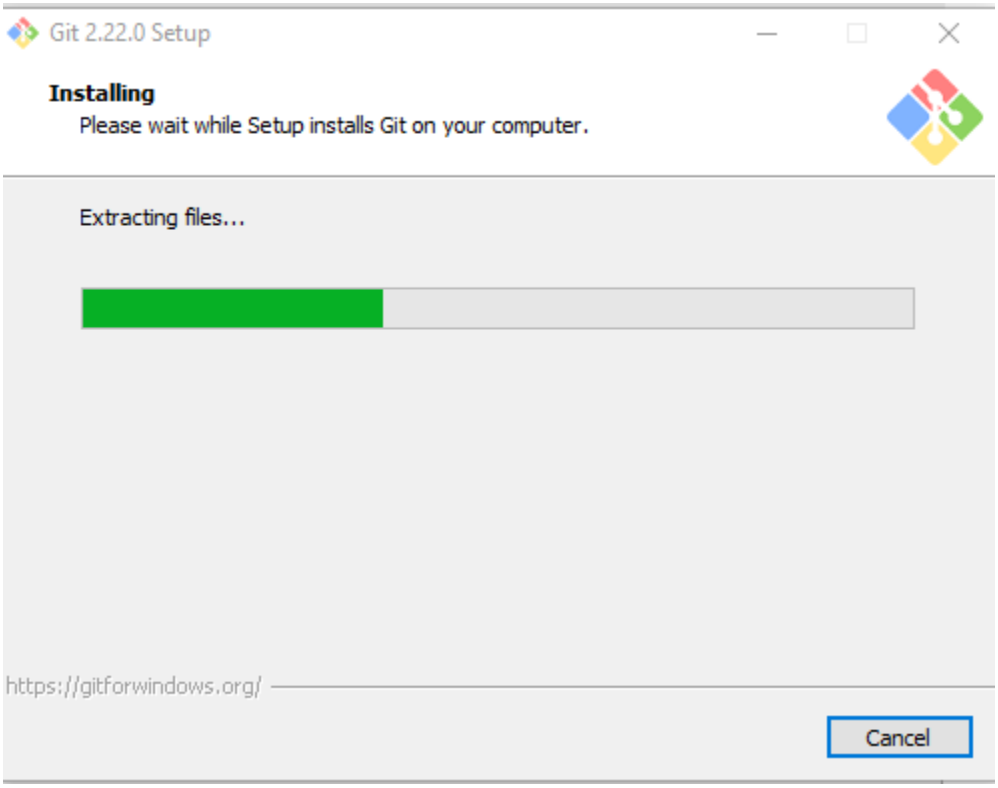

Figura 24 Instalación de Git.

Fuente: Elaboración propia

4.4. Instalación SonarQube

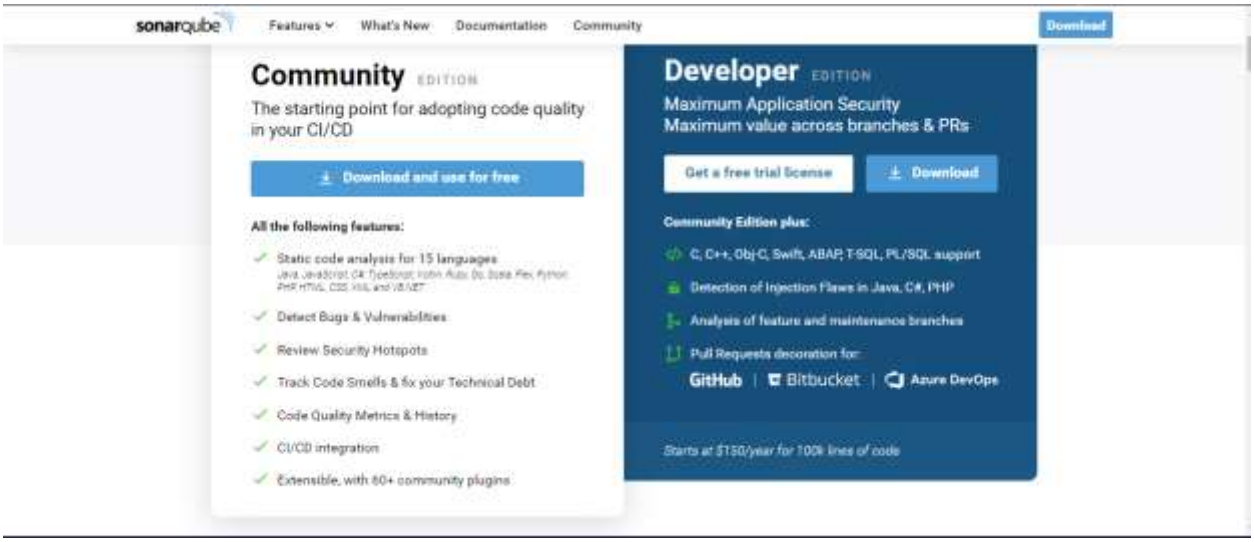

Figura 25 Página web de descarga del software SonarQube

Fuente: Elaboración propia

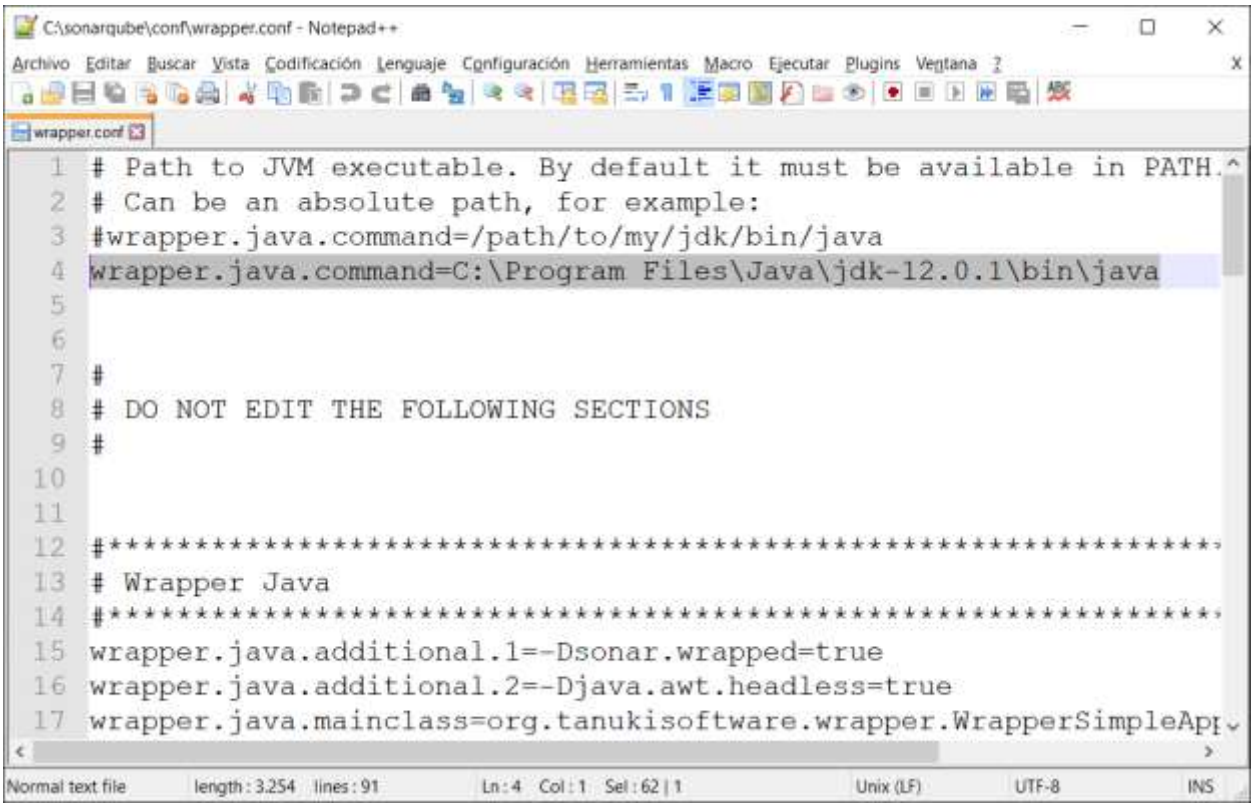

Figura 26 Configuración de SonarQube.

Fuente: Elaboración propia

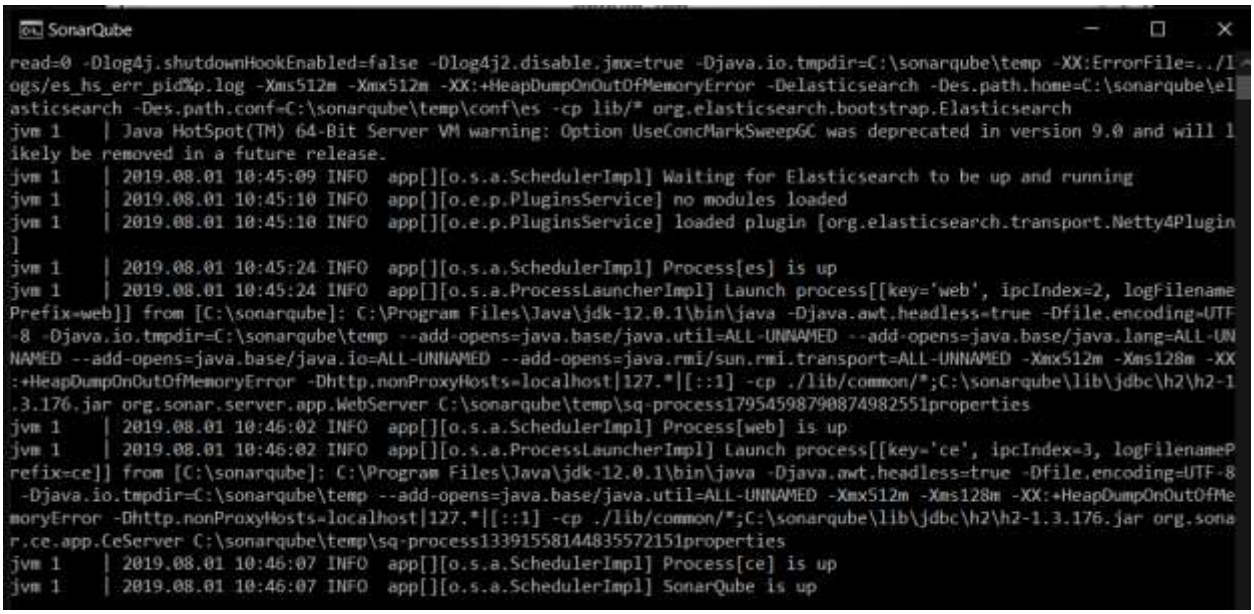

Figura 27 Ejecución SonarQube.

Fuente: Elaboración propia

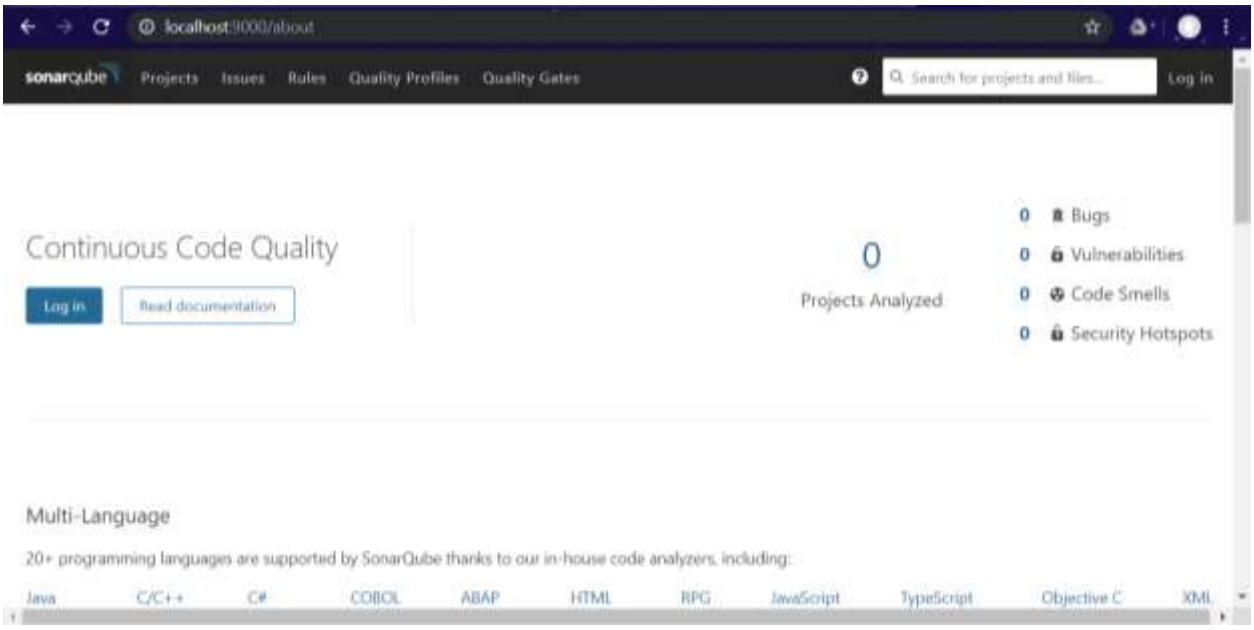

Figura 28 Ventana Principal SonarQube.

Fuente: Elaboración propia

4.5. Instalación de Jenkins

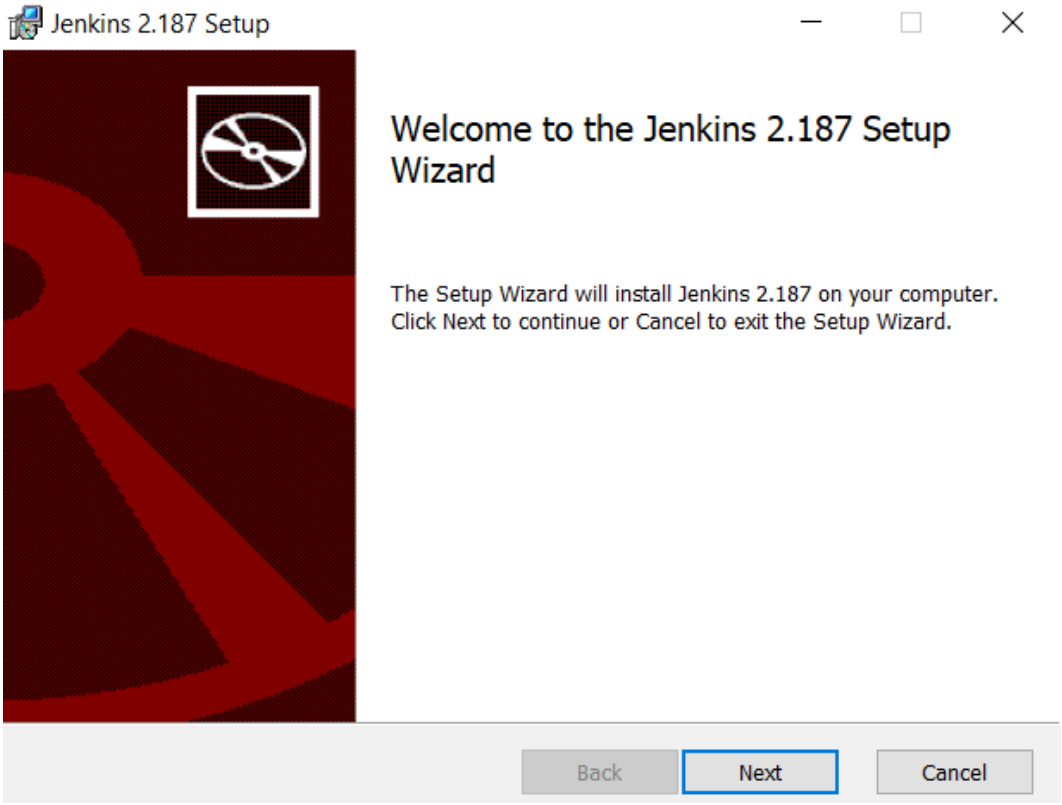

Figura 29 Asistente de instalación del Software Jenkins.

Fuente: Elaboración propia

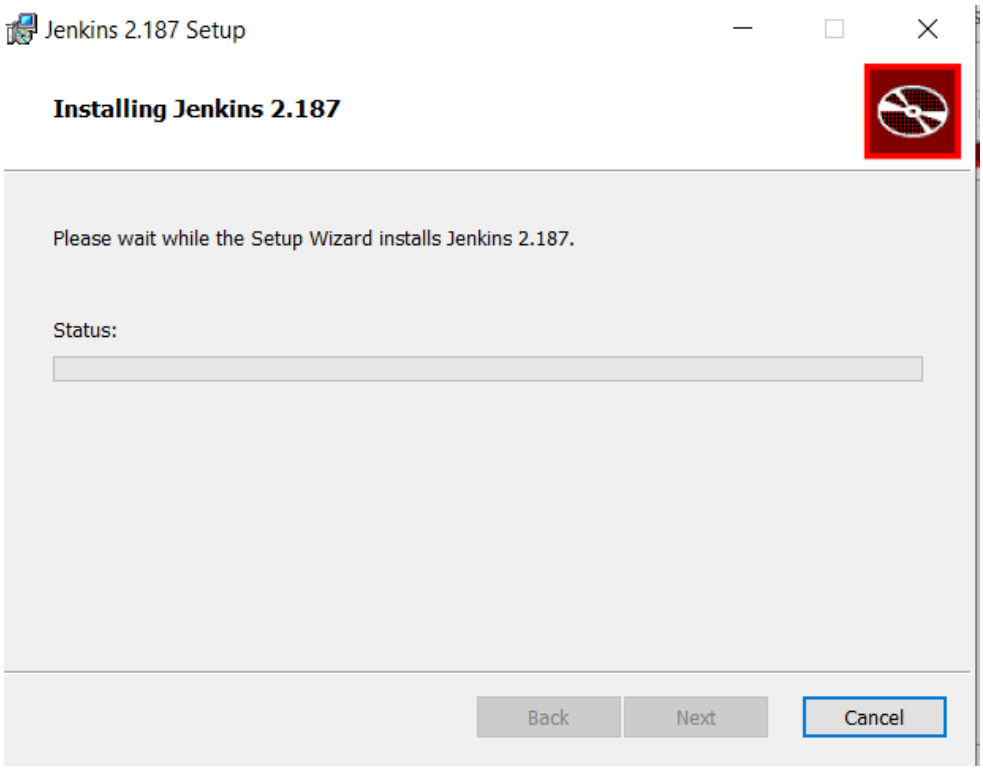

Figura 30 Instalación Jenkins.

Fuente: Elaboración propia

Figura 31 Desbloqueo de Jenkins.

Fuente: Elaboración propia

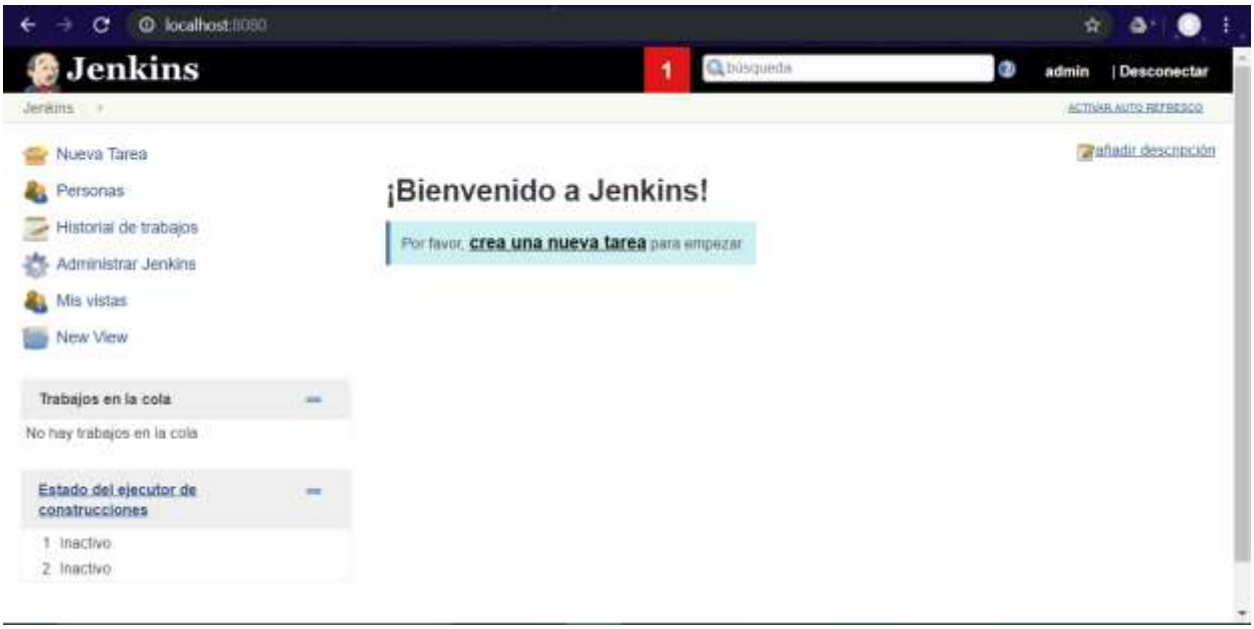

Figura 32 Ventana principal de Jenkins.

Fuente: Elaboración propia

Para el ambiente de pruebas del prototipo se instalaron y configuraron las herramientas sobre una máquina virtual utilizando el virtualizador VirtualBox (Debido a que es utilizado en los laboratorios del programa para realizar sus prácticas) y sistema operativo Ubuntu Server 18.04 (Ver figura 16).

Figura 33 Máquina Virtual con los servicios configurado.

Fuente: Elaboración propia

Se creó un repositorio en el portal de GitHub, seguidamente se adiciono a este un proyecto de software realizado sobre la plataforma de desarrollo Java (Ver figura 17)

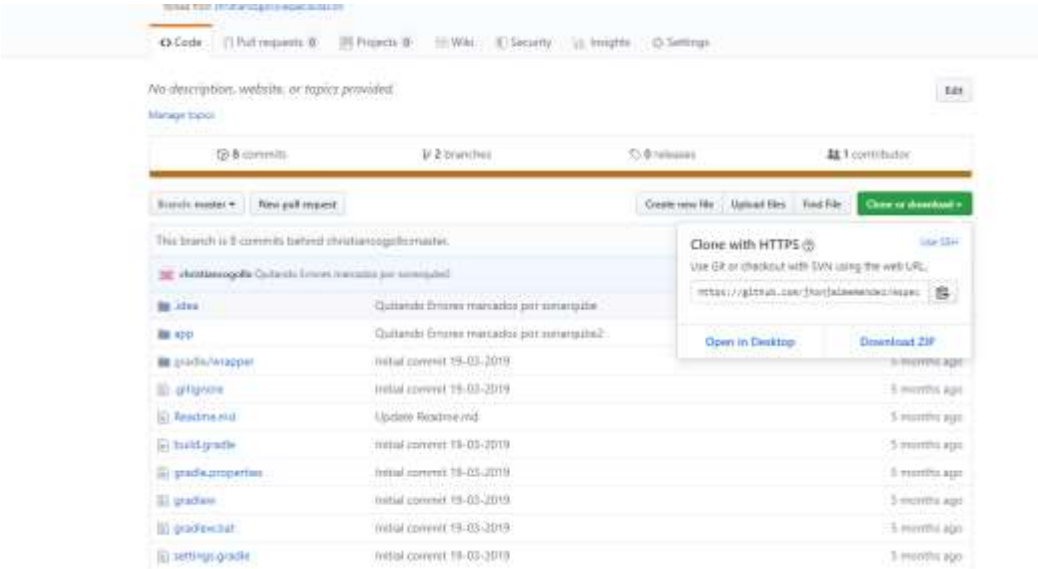

Figura 34 Repositorio creado en GitHub.
Fuente: Elaboración propia

Después de la creación del repositorio se procedió a la configuración de la herramienta de integración de Jenkins especificando el repositorio (ver figura 18) y la configuración de la herramienta de calidad de software SonarQube.

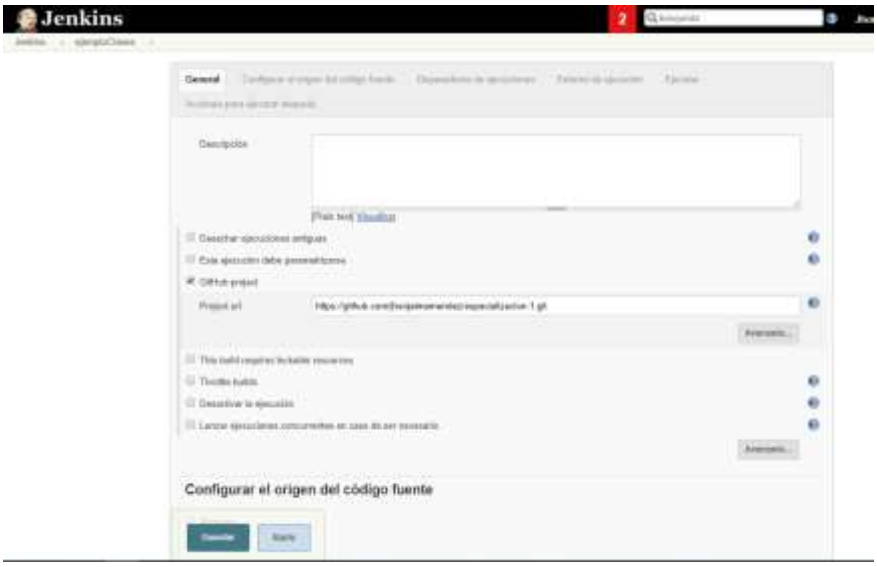

Figura 35 Configuración del repositorio en Jenkins.

Fuente: Elaboración propia

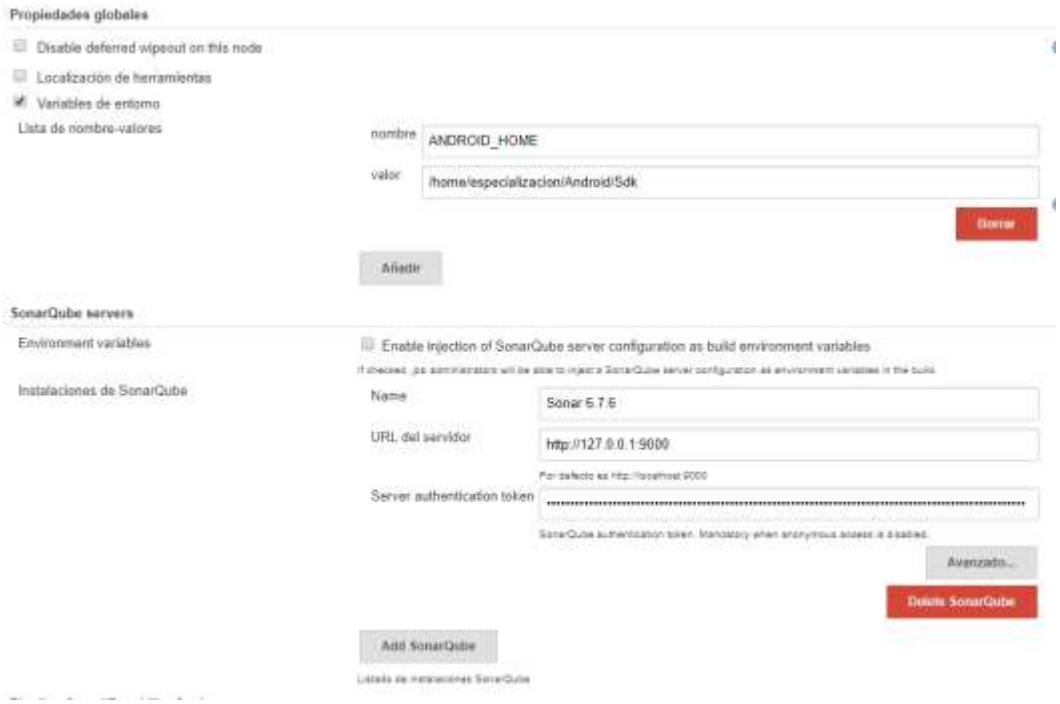

Figura 36 Configuración del SonarQube.

Fuente: Elaboración propia

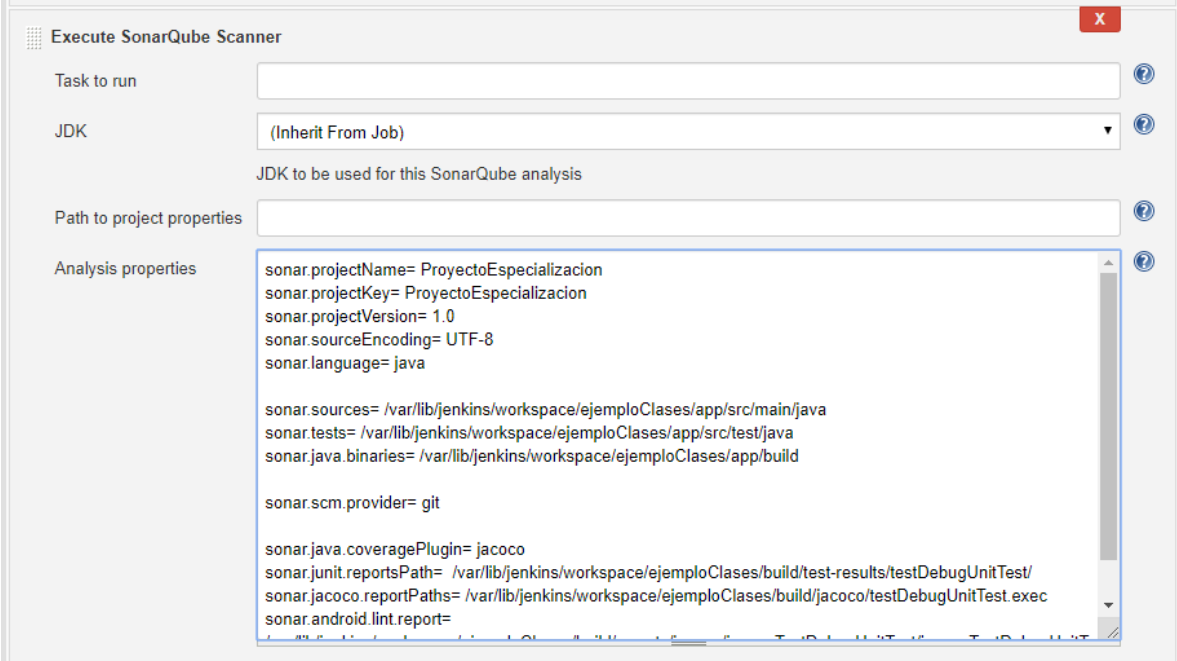

Figura 37 Pestaña de Configuración de SonarQube Scanner.

Fuente: Elaboración propia

Después de iniciar el sistema de integración continua se muestran los resultados obtenidos:

Figura 38 Resultados obtenidos por las herramientas de integración continua.

Fuente: Elaboración propia