ESCUELA DE INGENIERÍA EN SISTEMAS

Tema:

VALIDADOR DE REQUERIMIENTOS PARA LA EJECUCIÓN AUTOMÁTICA DE PRUEBAS UNITARIAS MEDIANTE UN PIPELINE

Proyecto de investigación previo a la obtención del título de Ingeniero de Sistemas y Computación

Línea de Investigación:

TECNOLOGÍAS DE LA INFORMACIÓN Y LA COMUNICACIÓN

Autor:

MATEO CASTAÑO VASQUEZ

Director:

MG. JOSÉ MARCELO BALSECA MANZANO.

Ambato – Ecuador

Agosto 2019

PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

SEDE AMBATO

HOJA DE APROBACIÓN

Tema:

"VALIDADOR DE REQUERIMIENTOS PARA LA EJECUCIÓN AUTOMÁTICA DE PRUEBAS UNITARIAS MEDIANTE UN PIPELINE"

Línea de Investigación:

TECNOLOGÍAS DE LA INFORMACIÓN Y LA COMUNICACIÓN

Autor:

MATEO CASTAÑO VASQUEZ

José Marcelo Balseca Manzano, Ing. Mg. f._____
CALIFICADOR

Galo Mauricio López Sevilla, Ing. Mg. f._____
CALIFICADOR

Liliana del Rocío Mena Hernández, Ing. Mg. f._____
CALIFICADORA

Mónica Patricia Mena Moreno, Ing. Mg. f._____
DIRECTORA ESCUELA DE INGENIERÍA EN SISTEMAS

Hugo Rogelio Altamirano Villarroel, Dr. f._____
SECRETARIO GENERAL DE LA PUCESA

AMBATO-ECUADOR

AGOSTO-2019

DECLARACIÓN Y AUTORIZACIÓN

Yo: **MATEO CASTAÑO VASQUEZ,** con **CC. 175976080-2,** autora del trabajo de graduación intitulado: "Validador de requerimientos para la ejecución automática de pruebas unitarias mediante un pipeline", previa a la obtención del título profesional de **Ingeniero de sistemas y computación,** en la escuela de **Ingeniería en sistemas.**

1.- Declaro tener pleno conocimiento de la obligación que tiene la Pontificia Universidad Católica del Ecuador, de conformidad con el artículo 144 de la Ley Orgánica de Educación Superior, de entregar a la SENESCYT en formato digital una copia del referido trabajo de graduación para que sea integrado al Sistema Nacional de Información de la Educación Superior del Ecuador para su difusión pública respetando los derechos de autor.

2.- Autorizo a la Pontificia Universidad Católica del Ecuador a difundir a través de sitio web de la Biblioteca de la PUCE Ambato, el referido trabajo de graduación, respetando las políticas de propiedad intelectual de Universidad

Ambato, agosto 2019

MATEO CASTAÑO VASQUEZ

CC. 175976080-2

DEDICATORIA

.

AGRADECIMIENTO

.

RESUMEN

La automatización de procesos en el desarrollo de *software* es una tendencia en incremento, es una manera de maximizar el provecho de recursos, por lo tanto es factible mencionar que en varias de las múltiples metodologías de desarrollo existentes, una de las etapas clave es la ejecución de pruebas unitarias y está también se puede automatizar, pero en muchas ocasiones esta etapa se realiza de manera manual o no se realiza por diferentes motivos tales como: bajo presupuesto del proyecto, la incorrecta planeación, el olvido del encargado de la ejecución o la falta de un instructivo para la ejecución automática de pruebas unitarias, por lo que el objetivo de este proyecto es desarrollar un instructivo, cuya finalidad es la creación de un sistema de herramientas interconectadas como un *pipeline* y que pueda ejecutar automáticamente las pruebas unitarias de un proyecto Java, para así evitar la ejecución manual y satisfacer la inexistencia de un instructivo de este tipo. El marco de trabajo Scrum se utiliza para dividir y definir tareas a desarrollarse de manera incremental durante periodos de tiempo, este marco de trabajo aportó orden, constancia y seguimiento para el completo desarrollo del proyecto, además, permite obtener los resultados esperados, como la creación del instructivo y la satisfacción de la necesidad del lector de ejecutar automáticamente pruebas unitarias sobre un proyecto Java.

Palabras claves: TestNg, Jenkins, SonarQube, Automatización de pruebas, pruebas unitarias, *pipeline*.

ABSTRACT

The automation of processes in software development is an increasing trend since it is a way to capitalize on resources. Therefore, it is important to mention that in several of the multiple existing development methodologies, one of the key stages is the execution of unit tests, which can also be automated. However, in many cases this stage is done manually or it is not done for different reasons such as a low project budget, poor planning, the person in charge forgets the execution of the tests, or the lack of guidelines for the automatic execution of unit tests. Consequently, the objective of this project is to develop a guide to create a system of interconnected tools, like a pipeline, with the ability to automatically run the unit tests of a Java project in order to avoid a manual execution and satisfy the inexistence of this type of guide. The Scrum framework is used to divide and define tasks to be developed increasingly during periods of time. This framework provided order, constancy and monitoring for the complete development of the project. In addition, it makes it possible to achieve the expected results, such as the creation of the guide and meeting the reader's need to automatically run unit tests on a Java project.

Key words: TestNg, Jenkins, SonarQube, automation of tests, unit tests, pipeline.

ÍNDICE DE CONTENIDOS

ÍNDICE DE GRÁFICOS

Gráficos

Tablas

INTRODUCCIÓN

El desarrollo de *software* observado desde un punto filosófico es algo maravilloso porque durante la historia pocas actividades han logrado ser tan revolucionarias e importantes como esta, aunque es una actividad relativamente nueva, su producto final el *software*, se encuentra en todos los ámbitos humanos, lo que resulta increíble y lo hace tan especial, es que el *software* es intangible pero tiene la capacidad de afectar la realidad del ser humano, su diario vivir y el entorno en el que se desarrolla.

El proyecto se enfoca en usar herramientas con las cuales se logre automatizar parte del proceso de aseguramiento de calidad del *software*, para que, al finalizar la actividad del desarrollo de *software*, este posea la capacidad de afectar la realidad como fue solicitado por el usuario.

Como antecedentes teóricos y prácticos dentro del desarrollo de *software* y aseguramiento de calidad, es pertinente comprender la importancia de *software* y como es mencionado por Pressman, R. S., & Troya, J. M. (1988) el *software* es un sistema con la capacidad de enviar, modificar, crear y almacenar información de poca y alta complejidad, dentro de esa breve definición se encuentra una palabra clave, la cual es uno de los activos más importantes en la época actual y da a entender en parte, por qué algo intangible es capaz de alterar la realidad y esta palabra es la información.

Frente al hecho de que un sistema pueda utilizar este activo (la información), demuestra su importancia en la sociedad, un ejemplo de esto es la información gubernamental, que sucedería si los secretos de estado como ubicaciones, planos detallados y cronogramas de actividades de bases militares es captada por una entidad u gobierno cuya intención es su utilización con fines bélicos y todo esto se dio porque el *software* que manipulaba esta información tuvo un fallo.

Con los antecedentes anteriores, es pertinente hacer elusión de sucesos históricos para profundizar en el tema, dentro de esto se encuentra la situación de la cual se publicó un artículo del autor Andrews (s/f) por medio de la página web del canal de

televisión *History*, en este artículo se relata como en la mañana del 9 de noviembre de 1979 el sistema NORAD encargado de vigilar el espacio aéreo de Estados Unidos dio la alerta a los técnicos encargados del sistema, que los soviéticos habían lanzado una serie de misiles a Estados Unidos, pensando que un ataque nuclear era inminente la fuerza aérea estadounidense inicio con su plan de defensa y ataque desplegando 10 aviones interceptores, alertando al presidente que iniciara con el plan "*doomsday*" y preparando sus misiles nucleares para la retaliación, poco tiempo después se hizo una verificación del sistema NORAD y los satélites, con lo que encontraron que un técnico ejecutó por error una simulación preprogramada de un ataque soviético sobre estados unidos, esto se dio al parecer porque el sistema no tenía una validación con la cual diferenciar entre las simulaciones y ataques reales, así que envió la alerta como si él ataque fuera real, este suceso tuvo repercusiones internacionales y pudo iniciar una guerra nuclear, pero por suerte no paso a mayores causas y es un antecedente histórico que permite entender de mejor manera la importancia del *software*, la información y la calidad que necesita el *software*.

La realización de un análisis más detallado de los sucesos ocurridos durante el incidente con el sistema NORAD se deduce que al ser un sistema que maneja tanta información y de índole tan delicada, se desarrolló con los más altos estándares de calidad de *software* para esa época e inclusive después de ese proceso se encontraron errores en producción, según lo expuesto, el error fue generado por una simulación, esto significa que para crear el sistema NORAD se corrieron pruebas simuladas para observar y asegurarse de que el sistema respondería adecuadamente y cumpliera con la calidad esperada.

Con el análisis descrito en el párrafo antepuesto, se sabe de un modo empírico que la calidad del *software* era alta, pero la calidad es un atributo o descripción que se le da a algo y cuyo significado para Asale. (s/f) es una: "Propiedad o conjunto de propiedades inherentes a algo, que permiten juzgar su valor", esta definición afirma la idea de que la calidad es relativa, son propiedades y características de algo, y estas palabras dan cabida a la pregunta ¿Qué características o propiedades indican que algo tiene mala o buena calidad?.

La calidad de *software* se define por las características y propiedades que posee, las cuales le confieren el atributo de alta o baja calidad, y como es descrito por Ahad, A., Ullah, Z., Tariq, L., & Niaz, S. (2017), si después de la verificación del *software* cumple algunas características como funcionalidad, confiabilidad, usabilidad, eficiencia, mantenibilidad, portabilidad y otras, se puede indicar si el *software* tiene calidad o no.

En relación a lo expuesto, nace otra pregunta, ¿cómo se verifica que el *software* posee las características mencionadas?, históricamente esto se realiza por medio de pruebas de multidisciplinarios aspectos, los autores Hooda, I., & Chhillar, R. S. (2015), dan una breve introducción a algunas de estas pruebas ejemplificado como las pruebas de carga sirven para observar cómo funciona el sistema bajo fuertes cargas de trabajo, las pruebas de rendimiento analizan la velocidad de respuesta y procesamiento del sistema o también pruebas de seguridad que comprueban que el sistema no tenga vulnerabilidades que puedan ser aprovechadas por personal no autorizado para utilizar el sistema y estas son solo algunas de las pruebas que se realizan.

El desarrollo de *software* se realiza siguiendo modelos tradicionales que tienen entre sus fases o etapas, el aseguramiento de calidad, esta etapa verifica que el *software* contenga las características anteriormente expuestas, pero estas metodología o modelos de desarrollo usualmente crean barreras internas en el proyecto, al parecer obligan al desarrollador de código a mantenerse inmiscuido solo en esta tarea y alejados de la organización y operaciones del proyecto, mientras que los profesionales de TI/Operaciones encargados del despliegue y mantenimiento del código separados del proceso de desarrollo de *software*.

Aunque todos los encargados de crear, desplegar, mantener y probar el *software* trabajaban en el mismo proyecto, al final resultaban tener objetivos distintos, direcciones de departamentos independientes, indicadores para evaluación de rendimiento diferentes, desconocimiento del trabajo de los demás que después de un tiempo se traducía en fricciones entre equipos de trabajo y rivalidades de trabajo, el desarrollador veía como un problema personal el reporte de errores por parte de los

tester, mientras que el área de operaciones se despreocupa de los errores causados por el código y la falta o mala ejecución de pruebas.

Los equipos de trabajo encasillados en metodologías de desarrollo tradicionales, fueron los primeros afectados por los problemas que acarreaba usualmente esta forma de trabajo, exigiendo y buscando una solución, dieron inicio a un movimiento revolucionario dentro de las comunidades de desarrollo de *software* y TI/Operaciones, el cual es llamado DevOps que según lo descrito por Brunnert, A., van Hoorn, A., Willnecker, F., Danciu, A., Hasselbring, W., Heger, C., ... & Koziolek, A. (2015) se puede resumir en que DevOps es más una filosofía o forma de trabajo y no una metodología, DevOps está enfocada en combinar Dev (*Development*/Desarrollo) con Ops(Operaciones/TI) y la automatización de estos procesos por medio de herramientas que al ser combinadas se conocen como *pipelines*, esta filosofía promete promover entregas del producto con mayor velocidad, menos embotellamientos en autorizaciones y papeleos, para mejorar la productividad al convertir las áreas de trabajo en un equipo de trabajo. Por lo que, en referencia a la filosofía de trabajo DevOps se puede sintetizar como un conjunto de herramientas y prácticas que buscan la automatización de procesos los cuales permitan desarrollar, asegurar la calidad, solucionar errores, desplegar y monitorear el producto de forma más eficiente y rápida.

La situación problemática es que existen nuevas tecnologías, técnicas y filosofías para la automatización de la ejecución de pruebas, pero estas no son implementadas dentro de los proyectos y aparentemente por el desconocimiento de las herramientas que pueden ser utilizadas, sin mencionar que en los casos que si utilizan herramientas para pruebas, se usan de forma manual, lo que aparentemente no influye en el punto de vista del usuario que recibe o utiliza el sistema, al final este espera un sistema estable, rápido y que realice correctamente la tarea que necesita, ni tampoco influye en la visión del desarrollador la cual es dar un producto que cumpla las expectativas del cliente y el crear, entregar y mantener este producto, se encuentre dentro del presupuesto pactado, pero en definitiva los resultados esperados pueden verse afectados por la no automatización de un proceso como la ejecución de pruebas, es desperdiciar la oportunidad de mejorar el proceso de aseguramiento de calidad, tener abierta la

posibilidad de olvidar ejecutar las pruebas manuales y permitir la inyección de errores en el sistema, además del uso de un recurso humano el cual ejecute manualmente las pruebas que podría utilizar su tiempo en otra tarea.

El planteamiento del problema es no ejecutar de manera automática pruebas unitarias del *software* por desconocimiento de cómo hacerlo, puede dar cabida a una abundante cantidad de problemas durante el proceso de desarrollo y entrega del producto al cliente, como un sistema inestable que necesita una fuerte inversión o gasto de tiempo en mantenimientos, que resulta en un proyecto que puede sobrepasar el presupuesto pactado dado los gastos extras que se realizan en corregir errores e incumpliendo con los resultados esperados por el usuario.

La hipótesis planteada es la creación de un instructivo con los pasos necesarios para crear un validador con la capacidad de ejecutar automáticamente las pruebas unitarias de *software* podría evitar los posibles problemas de ejecutar las pruebas unitarias de forma manual, además es destacable mencionar que Burnstein, I. (2006), también reflexiona que esta es una inversión de dinero, tiempo y esfuerzo que finalmente ahorrarán recursos durante todo el proyecto.

Para evitar los problemas que puede generar el olvidar la ejecución manual de pruebas, este proyecto tiene como objetivo general la creación de un instructivo cuyos pasos permitan la conexión entre herramientas que finalmente resulta en un validador para la ejecución de pruebas unitarias de manera automática, todo esto apoyado de la filosofía DevOps.

Para lograr el objetivo general se plantearon los siguientes objetivos específicos:

- Fundamentar teórica y metodológicamente aspectos relacionados al tema de estudio.
- Realizar un análisis comparativo de herramientas para el desarrollo de *pipelines*
- Implementar el validador de requerimientos apoyado en las herramientas seleccionadas para *pipelines*.

- Validar la propuesta mediante una prueba ejecutada sobre un prototipo de un sistema

La metodología utilizada en el proyecto es Scrum, que es catalogada como ágil y así como es descrita en Fuentes, J. R. L. (2015) tiene ciertas características beneficiosas para el proyecto, tales como son su agilismo y flexibilidad frente al cambio, es iterativa y está enfocada a las entregas después de cada iteración, por medio de esta metodología para el desarrollo del proyecto, se pretende que, a través de la ejecución automática de las pruebas unitarias sobre el código, se logre verificar que una parte de los requerimientos funcionales solicitados para el *software* se cumplan y en caso de no hacerlo se pueda alertar al desarrollador de la inyección del error, de manera complementaria esto disminuirá los posibles problemas que puede causar el no realizar pruebas en el *software*.

El proyecto se justifica bajo la premisa en que la automatización de la ejecución de pruebas sobre *software* para el aseguramiento de calidad, tiene más beneficios que las pruebas manuales y como es expuesto en la investigación de Kasurinen, J., Taipale, O., & Smolander, K. (2010), la inversión de recursos en la automatización de los procesos de pruebas sobre *software* han incrementado durante el tiempo, se ha demostrado que permiten enfocar el talento humano en tareas menos repetitivas, crear procesos estandarizados y repetibles de aseguramiento de calidad, mejorar tiempos de entrega, evitar errores humanos tales como olvidar ejecutar las pruebas necesarias y entre otros beneficios la reducción de costos por solucionar errores en etapas maduras del desarrollo de *software,* con estos puntos expuestos el proyecto encuentra su validez para ser desarrollado y posible utilidad para solucionar la problemática planteada.

CAPITULO I. ESTADO DEL ARTE

1.1. Metodologías de desarrollo

El desarrollo de *software* es una actividad, que usualmente se compone de múltiples etapas y equipos de trabajo de más de una persona, resulta a cierto grado complicado y engorroso de realizar, por lo cual durante el tiempo algunas empresas y actores dentro de la industria han desarrollado metodologías de trabajo, todo esto con la intención de guiar el proceso de desarrollar *software*, organizarlo y estandarizarlo, por medio de buenas prácticas e ideologías de trabajo que han funcionado anteriormente en otros proyectos, y así finalmente simplificar el trabajo necesitado para futuros proyectos.

Dentro de las metodologías de *software* se encuentran dos grandes ramas, las metodologías tradicionales y las metodologías ágiles.

1.1.1. Metodologías tradicionales de desarrollo de *software*

Las metodologías tradicionales de desarrollo de *software* están más enfocadas a la planificación del proyecto y el cumplir esa planificación por lo cual no logran ser flexibles al cambio y usualmente el entorno se vuelve volátil, si se presenta un error en etapas finales, desestabiliza fuertemente el proyecto en su totalidad, tal y como fue mencionado por Figueroa, R. G., Solís, C. J., & Cabrera, A. A. (2008). Uno de los principales problemas de estas metodologías es la entrega en grandes paquetes de *software* por lo cual la detección de errores y corrección de los mismo solo se logra en etapas finales, todas estas desventajas normalmente se traducen en sobrecostos, tiempo y desperdicio de los recursos.

Dentro de algunas de las metodologías de desarrollo tradicional se encuentran:

RUP (*Rational Unified Process*): En la investigación de Pérez A. O. (2011), se indica que esta metodología está basada en modelos cascada y se encuentra dirigido por los casos de uso, arquitectura e iteraciones. Esta caracterizado por:

- **Casos de uso:** Es la descripción de la necesidad del usuario y que solicita que el sistema realice para solventarla, se realiza como una secuencia de procesos o interacciones entre el *software* y el usuario.
- **Centrado en la arquitectura:** El estar centrado en la arquitectura, se refiere a las herramientas y la conexión técnica entre ellas, por lo cual permite tener una visión más global del proyecto, la reutilización de componentes y conocer el cómo se conectan todas las partes.
- **Iterativa:** Se realizan iteraciones de desarrollo para hacer entregas parciales del producto.

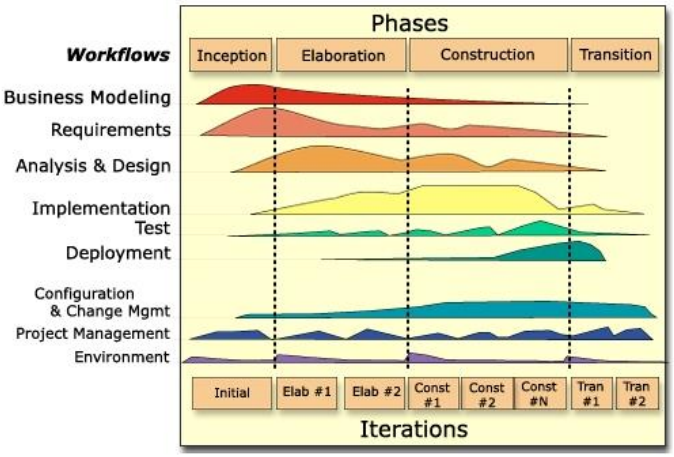
Conjuntamente a las características anteriores, la metodología RUP se compone por fases y según lo descrito por Martínez, A., & Martínez, R. (2014), estas se pueden sintetizar en:

- **Inicio**: En esta fase se recolecta toda la información justa y relevante para realizar un análisis de la situación y llegar a la conclusión de continuar o no con el proyecto, esta etapa es de corta duración de tiempo. En ocasiones durante esta etapa se hacen estimaciones de tiempo, costo y riesgo.
- **Elaboración**: La fase de elaboración sirve para establecer concretamente la arquitectura del proyecto y desarrollar el plan creado para el proyecto. En esta etapa se crea un prototipo de la arquitectura general del proyecto que será afectado durante las iteraciones.
- **Construcción**: Se espera que en esta fase se realice la creación, desarrollo e implementación de todos los requisitos y características de manera iterativa e incremental, para que el usuario tenga el producto y pueda utilizarlo en una versión beta.

- **Transición**: En la fase de transición se entrega el producto en versiones actualizadas para que el cliente utilice, se realiza entrenamiento al usuario, ajustes generales, y corrección de errores detectados

Gráficamente la metodología RUP es representada del siguiente modo:

Gráfico 1. *Rational Unified Process.*



Fuente: eRA: RUP Fundamentals Presentation. (s. f.)

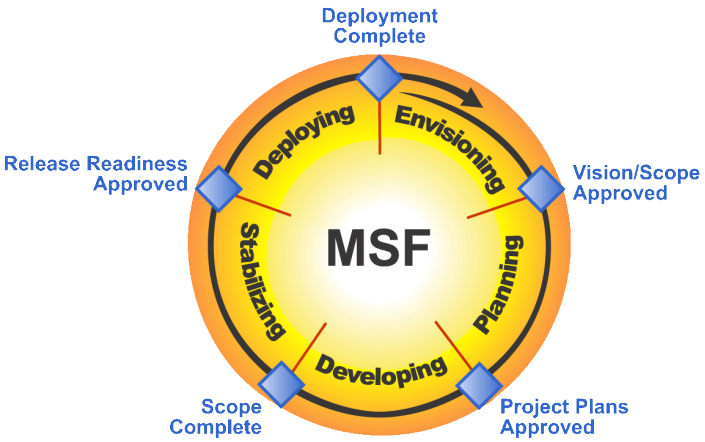
MSF (*Microsoft Solutions Framework*): La metodología *Microsoft Solutions Framework* creada por Microsoft, tiene un enfoque a la planificación por riesgos, es iterativa y hace entregas en cada iteración. Esta metodología se compone por fases y puede ser adoptada en múltiples organizaciones, no solo está diseñada para el desarrollo de *software*. Según la información detallada en Mendoza, L. E., Grimán, A., & Pérez, M. (s/f), entre las fases que componen esta metodología están:

- **Visión**: se realiza la definición del producto, objetivos y requerimientos que el cliente solicita, su entregable es el alcance y la visión aprobados por el negocio y el cliente.
- **Planeación**: En esta fase se espera que se logre la creación de un plan de trabajo, con cronogramas y actividades para cumplir con la meta trazada en la visión, el entregable de esta fase es el Plan del proyecto aprobado.

- **Desarrollo**: Se desarrolla el alcance y las actividades necesarias para la creación del producto, se realiza por hitos y entregables que se integran unos con otros para observar su funcionamiento como un todo, el entregable de esta fase es el Alcance completo.
- **Estabilización**: Durante esta fase se realizan las pruebas en un ambiente real, para observar su funcionamiento, el entregable de esta fase es el *Release Readiness* aprobado.
- **Implantación**: Se realiza la implantación del proyecto en el cliente, capacitándolo, adecuando el ambiente, y finalmente la fase es culminada con la Implantación completa y el visto bueno por parte del cliente.

Gráficamente el proceso de la metodología se puede observar en la siguiente imagen:

Gráfico 2. *Microsoft Solutions Framework*.



Fuente: Archiveddocs. (s. f.)

1.1.2. Metodologías agiles de desarrollo de *software*

Estas metodologías ágiles nacen como respuesta frente a los múltiples cambios, impedimentos y modificaciones que ocurren durante el desarrollo de *software* y que no eran suplidos por las metodologías tradicionales que son poco flexibles y no logran

adaptarse a cambios, el proyecto sufre fuertes impactos si estos cambios se dan durante etapas o fases maduras del proyecto.

Estos problemas que afectan de manera directa los proyectos, impulso a que varias organizaciones y empresas iniciaran el proceso de creación de metodologías más flexibles las cuales tengan la capacidad de adaptarse a las necesidades cambiantes del proyecto y del cliente.

La gran mayoría de las metodologías agiles están basadas en los 12 principios del manifestó para el desarrollo ágil de *software* por Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... & Kern, J. (2001), donde se describen bases como: el cliente es la prioridad número uno durante el proyecto, las entregas son continuas y en periodos cortos de tiempo, la mejor manera de medir la eficiencia del proyecto y del equipo es por la entrega de *software* funcional, la importancia de la simplificación de tareas y maximización de las actividades a realizar, y más bases con las cuales el *software* se desarrolla de modo ágil.

Algunas de las metodologías que se encuentran dentro de la categoría de agiles son:

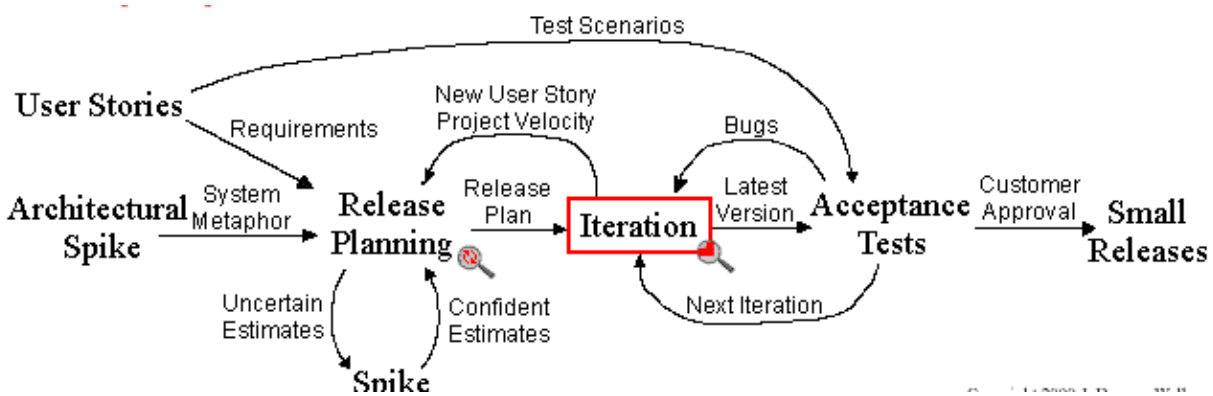
XP (*EXTREME PROGRAMMING*): Con la información recopilada de Beck, K., & Gamma, E. (2000), se factible decir que la metodología ágil de *extreme programming* es usualmente utilizada en proyectos cuyos requisitos de negocio son cambiantes continuamente y que en ocasiones no se encuentran demasiado claros, su enfoque está basado en la disminución de documentación y el aumento de la comunicación cara a cara y verbal, para transmitir de mejor manera la información, además hace un énfasis en el trabajo en equipo y la buena relación interpersonal, realiza continuamente capacitaciones a los integrantes del equipo. Para mejorar el dinamismo de trabajo y maximizarlo esta metodología solicita el acompañamiento constante y disponibilidad total por parte del cliente, con la intención de disminuir confusiones y resolver dudas de manera oportuna.

En el texto producido por Letelier, P. (2006). Se detalla que las fases de la metodología *extreme programming* son:

- **Exploración:** Esta fase es para la familiarización del equipo de trabajo con el proyecto, las herramientas, tecnologías que se utilizaran y la creación de prototipos con las historias de usuario que fueron mencionadas como centrales o iniciales por el cliente.
- **Planificación de la entrega:** Se llega a un acuerdo entre las historias de usuario que el cliente prioriza y lo que puede realizar el equipo en una iteración, adicionalmente el programador asigna puntos de esfuerzo a cada historia para hacer un seguimiento de la velocidad de desarrollo y de este modo tener información base para planear la cantidad de historias y tiempo necesario para futuras iteraciones.
- **Iteraciones:** Se realiza la planeación de que se entregara al finalizar la iteración y se realiza el desarrollo de las historias de usuario acordadas y priorizadas entre el cliente y el equipo de trabajo, adicionalmente se hace un plan de trabajo para la asignación de las tareas de programación a cada historia de usuario, usualmente estas tareas de programación son realizadas por parejas de programadores.
- **Producción:** En esta fase se revisa el sistema por medio de pruebas para validar el correcto funcionamiento del sistema antes de ser implementado en ambientes productivos, además estas revisiones proporcionan información de características nuevas que puedan ser necesarias para implementación.
- **Mantenimiento:** Frente a las entregas continuas, el nuevo desarrollo de funcionalidades y el sistema que ya se encuentra en funcionamiento en ambiente productivo, es necesario realizar soporte y mantenimiento para atender los problemas que se pueden presentar.
- **Muerte del proyecto:** Finalmente no se realizan más cambios en el sistema o arquitectura, no se tienen más historias de usuario por parte del cliente y el enfoque del equipo de trabajo está en aumentar la confianza del cliente en el sistema y mejorar el rendimiento.

Gráficamente la metodología *extreme programming* se representa del siguiente modo:

Gráfico 3. Metodología *extreme programming*.



Fuente: XP flow Chart. (s. f.).

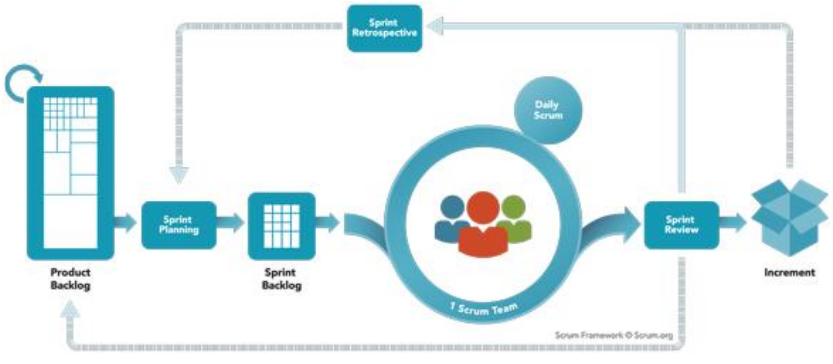
SCRUM: Aunque esta metodología ágil inicialmente se creó para ser implementada como una guía del proceso de desarrollo de *software*, es utilizada de manera exitosa en múltiples empresas tanto de desarrollo de *software* como de otras áreas, tiene la capacidad de aportar prácticas para el desarrollo de proyectos por su flexibilidad y bases que de cierto modo son genéricas y reutilizables no solo en el ámbito del *software*.

En la reconocida página *What is Scrum*? (2019), sintetiza que Scrum no es precisamente una metodología, es más una recopilación de buenas prácticas o marco de trabajo empírico, que se apoya del trabajo en equipo para la resolución de problemas imprevistos, la adaptación y entrega continua de productos funcionales para el cliente.

Scrum se realiza por medio de eventos o también llamados ceremonias y según Ken Schwaber & Jeff Sutherland (2016), se pueden resumir en:

- ***Sprint***: Es el periodo de tiempo que durara cada iteración, en este tiempo se realizan las actividades necesarias para finalmente entregar producto funcional al cliente.
- ***Sprint Planning***: Esta ceremonia se realiza al inicio del *sprint*, el *scrum master* y el equipo de trabajo priorizan las tareas a realizarse del *(product backlog)* durante el *sprint*, se les da un puntaje de esfuerzo y cada desarrollador se asigna las tareas del *product backlog* que puede realizar durante el sprint.
- ***Daily Scrum***: Diariamente se realiza una pequeña reunión de máximo 15 minutos con todo el equipo de trabajo, durante esta reunión se mencionan los inconvenientes o bloqueos que tengan para realizar sus asignaciones, así de este modo el *Scrum master* conoce que tareas se pueden atrasar, para evitar estos atrasos inicia con el proceso de eliminación de impedimentos.
- ***Sprint Review*:** Al finalizar el *sprint* se realiza una reunión en la cual se muestra al cliente, el trabajo realizado durante el *sprint* y el producto funcional a entregar para ambiente productivo. Adicionalmente se verifica que el cliente se encuentre satisfecho con el producto y los cambios que puedan ser necesarios en el producto entregado.
- ***Sprint Retrospective***: Este evento usualmente se realiza después del *sprint review* con los miembros del equipo de trabajo y el *Scrum master*, durante la misma se validan que se hizo bien durante el *sprint*, que no se volverá a hacer y se lleva un control de las lecciones aprendidas para futuros *sprints*.

Gráfico 4. Ciclo de vida en Scrum.



Fuente: What is Scrum? (2019).

1.1.3. Comparación entre RUP, MSF, SCRUM y XP

Las diferentes metodologías tratadas con anterioridad muestran algunas similitudes, por lo que existe motivo suficiente para comparar entre varias de las características que comparten en diferentes niveles y estas pueden ser:

Tabla 1. Comparación entre RUP, MSF, SCRUM y XP.

	Documentación Generada	Iterativa	Enfoque	Adaptabilidad al cambio	Facilidad
SCRUM	Baja	Alto	Administrativo	Alta	Alta
XP	Baja	Alto	Programación	Alta	Media
RUP	Alta	Medio	Arquitectura	Media	Baja
MSF	Alta	Medio	Riesgos	Media	Baja

Fuente: elaboración propia.

Las metodologías Scrum y XP están enfocadas más a la realización del proyecto que en documentarlo, mientras que RUP y MSF tienen como parte del proceso de desarrollo la documentación exhaustiva del proyecto y esto puede dispersar de cierto modo el esfuerzo y dilatar los tiempos necesarios para entregar producto funcional, es posible observar esta característica como una debilidad, adicionalmente el tener una adaptabilidad al cambio menor a SCRUM y XP las ponen en desventaja ante los proyectos actuales cuya característica común es el cambio continuo. Entre Scrum y XP cabe destacar que, aunque tiene gran similitud, Scrum supera por su facilidad de uso a XP, conjuntamente la implantación en múltiples tipos de proyectos no enfocados en *software* habla de la flexibilidad que posee esta metodología para organizar, dirigir y potenciar proyectos.

1.2. Calidad de *software*

La calidad de *software*, mencionada por múltiples autores como Crosby, P. B. (1979), donde da a entender que la calidad es, que tanto el sistema se adapta a los requerimientos solicitados por el cliente, mientras que Juran, J. M. (1988) piensa que la calidad se refiere a como el sistema logra la satisfacción de las necesidades del cliente, aunque no sea específicamente los requisitos solicitados por el mismo. Otra de las definiciones es la de Pressman, R. S. (2000), en la que detalla que calidad es la entrega de *software* con las características especificadas del cliente y que cumpla con lo solicitado por el mismo, además se agrega que el desarrollo y producto final estará alineado con las buenas prácticas de ingeniería de *software* en general.

Con la información anterior, se extracta, que la calidad es un conjunto de características que el *software* tiene, la inclusión de buenas prácticas que todo proyecto de *software* tiene la necesidad de poseer y finalmente estas características cumplirán con las expectativas que el cliente solicito.

1.2.1. Aseguramiento de calidad en el *software*

El desarrollo de *software* conlleva múltiples etapas para finalizar con un producto funcional para el cliente, pero tal como dicen Lin, L., He, J.,Zhang, Y., & Song, F (2015), en ocasiones se realizan entregas a ambiente productivo sin el respectivo escrutinio de los escenarios o rutas que el sistema tiene más allá de los que el desarrollador por intuición puede probar, esto puede conllevar a sistemas defectuosos, con inconsistencias o errores que pudieron ser evitados por medio de pruebas. Entre algunos de los motivos por los cuales se ignoran las pruebas que pueden asegurar la calidad, están los presupuestos ajustados o tiempos de entrega mal planeados donde olvidaron tener en cuenta esta etapa.

Para realizar el aseguramiento de calidad que comprueba la existencia de las características que tendría el *software*, se realizan pruebas según Galin, D. (2004), en dos tipos de clases:

- **Pruebas de caja blanca:** Durante estas pruebas se revisa el flujo de pasos que realiza una funcionalidad internamente para llegar a un resultado esperado.
- **Pruebas de caja negra:** Con estas pruebas se observa la funcionalidad por medio de entradas y que las salidas después de ejecutado el proceso sean las correctas, no se verifica como se realiza el proceso interno.

Adicionalmente a la división en clases, también existe una clasificación en tipos de pruebas como es detallado por Graham, D., Van Veenendaal, E., & Evans, I. (2008) donde se describen:

- **Pruebas unitarias:** Este tipo de pruebas esta enfocadas a verificar el funcionamiento de un componente aislado y único.
- **Pruebas de integración:** En el momento de que múltiples componentes se relacionan entre ellos, se realizan pruebas de integración, las cuales verifican que todos los componentes en unión funcionen correctamente, a diferencia de las pruebas unitarias que solo validan el componente A, en este caso se valida como el componente A, B, C… funcionan al estar integrados.
- **Pruebas de seguridad:** Mediante estas pruebas se pone a prueba la seguridad del sistema, posibles vulnerabilidades y fallos que puedan permitir la intromisión, robo o alteración de la información por agentes externos o internos no autorizados
- **Pruebas de carga:** Se realiza la simulación de altos volúmenes de peticiones para encontrar puntos de fallos y límites del sistema frente a situaciones imprevistas pero que pueden suceder.
- **Pruebas de rendimiento:** Del sistema se espera cierto tiempo de respuesta ante el uso normal y continuo, así que estas pruebas miden como se utilizan los recursos para realizar cierta tarea, e igualmente si su velocidad de respuesta es

la esperada, finalmente estos datos recopilados durante un periodo de tiempo permiten la toma de decisiones para optimizaciones en futuras entregas.

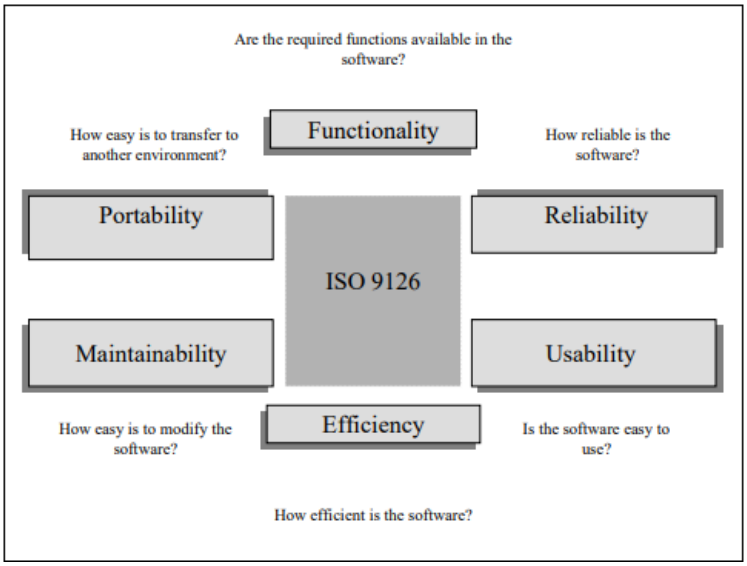
- **Pruebas de estrés:** Con estas pruebas se lleva al límite el sistema por medio de la inyección o simulación de la cantidad máxima de registros, usuario, accesos o información que el sistema use, además en condiciones extremas como falta de memoria en servidor, múltiples tareas de alto consumo de recursos funcionando simultáneamente, volúmenes altos de transacciones y demás, de este modo se observa su funcionamiento bajo estas condiciones, se analiza como el sistema responde frente esta situación y como vuelve a su estado normal de funcionamiento.
- **Pruebas de usabilidad:** Se realizan pruebas donde se observa que tan fácil o difícil es para el usuario utilizar el sistema, permite encontrar puntos de mejora de diseño de procesos tanto internamente como de interfaz para mejorar la experiencia del usuario.

Entre las características propuestas para evaluar por medio de pruebas, se encuentran las de la ISO 9126 que fueron detalladas en la investigación de Chua, B. B., & Dyson, L. E. (2004). Estas características son:

- **Funcionalidad**: Se refiere a las funciones que satisfacen los requerimientos explicita e implícitamente.
- **Fiabilidad**: Es la capacidad del sistema en este caso del *software* de mantener la prestación del servicio en ciertas condiciones durante el tiempo.
- **Usabilidad**: Es la facilidad o dificulta para el usuario el utilizar el *software*.
- **Eficiencia**: Es la relación que existe entre el desempeño del *software* con los recursos necesitados para realizar cierta tarea durante un periodo de tiempo.
- **Mantenibilidad**: Se entiende como la facilidad o dificultad para agregar, quitar, editar funcionalidades en el *software*, la facilidad de análisis del código, funcionamiento lógico del *software* internamente y el impacto a su estabilidad o la prestación del servicio al realizarse modificaciones.

- **Portabilidad**: Está relacionada con la capacidad de cambiar el *software* de plataforma y ser implementado en otras tecnologías.

Gráfico 5. ISO 9126.



Fuente: Chua, B. B., & Dyson, L. E. (2004)

1.3. DEVOPS

Como breve introducción se aprecia que DevOps es una filosofía enfocada en la automatización de procesos por medio del uso de herramientas conectadas las cuales agilizan el trabajo de creación, despliegue y mantenimiento de *software*, a continuación, se profundiza en el tema.

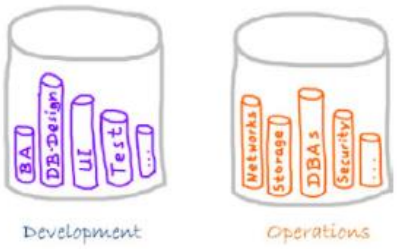
1.3.1. Inicios de DEVOPS

DevOps inicia como un movimiento revolucionario durante los años 2007 y 2008 según Atlassian (2019), en foros de internet y conferencias a nivel mundial, en el momento que las personas encargadas de trabajar en áreas de desarrollo de *software*, infraestructura y soporte de *software* se dieron cuenta de las falencias de trabajo en equipo dentro de sus organizaciones.

Estas personas manifestaron su descontento a los principales problemas que se presentaban, tales como el aislamiento de equipos de trabajo dentro de un mismo proyecto, en este caso en específico, el área de desarrollo (*Development / DEV)* y el área de Operaciones (*Operations / OPS*), tal y cual es redactado en Bass, L., Weber, I., & Zhu, L. (2015), se generan fricciones entre estas dos áreas de trabajo debido a los puntos de vista de lo que consideran como un trabajo éxitos, mientras que *DEV* tiene el pensamiento que un trabajo exitoso es el cambio constante por la creación de nuevas funcionalidades y despliegues constantes, por otra parte *OPS* espera estabilidad en los servicios, entonces no desean cambios de desarrollo que puedan alterar la estabilidad del sistema.

Esta forma de pensar finalmente generaba la separación de áreas, trabajos, métricas, objetivos y equipos de trabajo, que al contrario a trabajar separados tienen que estar apuntando a la misma meta del proyecto.

Gráfico 6. Separación de áreas.



Fuente: Bass, L., Weber, I., & Zhu, L. (2015).

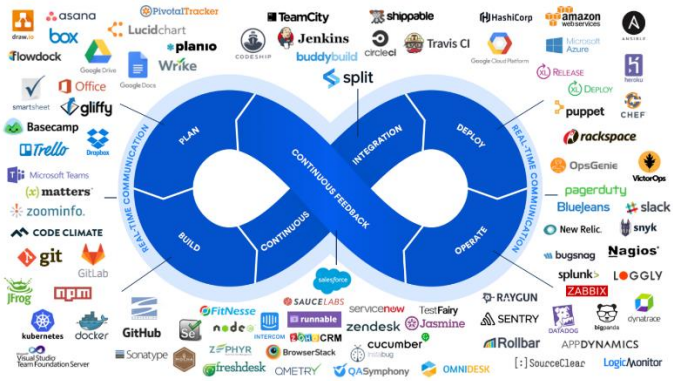
1.3.2. ¿Qué es DEVOPS?

DevOps no se define como metodología sino como filosofía o conjunto de buenas prácticas de trabajo que cambian la cultura organizacional, como es expuesto por Walls, M. (2013), donde puntualiza que la mentalidad de los involucrados dentro de la empresa se ve modificada para mejorar el trabajo en equipo y automatizar procesos que anteriormente se realizaban manualmente por medio del uso de herramientas y

así lograr el desarrollo, prueba y publicación de *software* más rápido, eficiente y confiable.

DevOps es usualmente representado gráficamente del siguiente modo:

Gráfico 7. DevOps y algunas de sus herramientas.

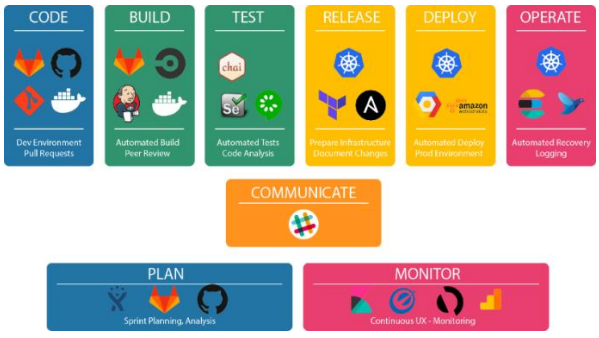


Fuente: #1 Marketplace for DevOps apps. (s. f.)

1.3.3. ¿Qué es un *PIPELINE* en *DevOps*?

Un *pipeline* (tubería) en DevOps se define según Verona, J. (2016), como la conexión de múltiples herramientas las cuales permiten realizar una tarea de manera automática y que siga un flujo de trabajo continuo. Un ejemplo de esto es la siguiente imagen:

Gráfico 8. Posibles secciones de un *pipeline*



Fuente: Leicher, A. (2018, julio 20)

Para la creación de un *pipeline* con la información de la imagen anterior se puede realizar la unión de una herramienta del bloque *CODE* (GitHub) con otra herramienta del bloque *BUILD* (Jenkins) y una del bloque *TEST* (Selenium), para generar un *pipeline* pequeño el cual versionaría el código con GitHub, luego Jenkins lo compilaría para finalmente ejecutar de manera automática las pruebas de Selenium y en caso de fallas realizar reportes o alertar a los encargados para sus arreglos, este *pipeline* podría estar integrado con otros más grandes que realicen despliegues en ambientes pre-productivos o productivos y realicen el monitoreo de la estabilidad del sistema, todo de manera automática, en conclusión esto cumpliría con el cometido principal, la unión de desarrollo con operaciones, automatización de procesos y la realización de entregas continuas, eficientes y confiables.

1.4. Herramientas para versionamiento de código

El versionamiento de código permite crear repositorios donde se encuentra el código fuente del *software*, desde este repositorio se puede descargar o realizar actualizaciones del código, con un respaldo de versiones de como se ha modificado, creado, eliminado durante el tiempo y como es explicado por Orozco, G., & Righi, L. G. (2018), su principal ventaja es que permite a un grupo de personas trabajar de manera simultánea en un proyecto sin la necesidad de esperar que otra persona termine de realizar sus modificaciones.

En la documentación expuesta por GitHub Glossary - GitHub Help. (2019) explican la terminología utilizada dentro de la mayoría de los sistemas de versionamiento de código, esto se puede sintetizar, en que usualmente estos sistemas permiten la creación de un repositorio de código que se encuentra en la nube, este repositorio tiene una estructura ramificada, en la cual desde una rama (*branch)* central se puede crear una nueva rama idéntica a la anterior. Todo el repositorio se puede descargar a la maquina local del desarrollador para que pueda realizar los cambios deseados, y en el momento que el desarrollador o encargado de los cambios desee sincronizar sus

cambios al repositorio en la nube, realiza un *commit* que primero sincroniza sus cambios con el repositorio en su máquina para finalmente realizar un *push* que remite y sincroniza los cambios en su repositorio local con el repositorio en la nube. Adicionalmente existe la opción de bajar cambios que realizo otra persona o sincronizar el repositorio local con el repositorio en la nube, al realizar un *pull.*

Dado la existencia de una gran cantidad de herramientas que permiten el versionamiento de código, solo se realizará la investigación sobre las 2 herramientas que comúnmente son más usadas, que son GitHub y GitLab.

1.4.1. GITHUB

En la página oficial de GitHub Build software better, together. (s. f.), se detalla como esta empresa inicio en octubre del 2007 con su primer *commit* y su crecimiento por medio de inversiones de dinero, cambios continuos en su cultura organizacional y esfuerzo, lograron para el 2019 tener más de 31 millones de usuarios registrados, aproximadamente 100 millones de repositorios de proyectos y ser una empresa reconocida internacionalmente por su excelente servicio.

GitHub en un principio prestaba únicamente el servicio de versionamiento de código, pero con el pasar de los años y su evolución constante, crecieron su portafolio de servicios por lo que actualmente también proporcionan:

Revisión de código: Los integrantes del equipo de trabajo del proyecto o usuarios de la comunidad de GitHub pueden expresar sus opiniones para mejorar el código con buenas prácticas o diferentes formas de programar para maximizar el rendimiento y la utilización de todas las posibilidades

Gestión de proyecto: Por medio del tablero que proporciona GitHub se puede gestionar, asignar y hacer seguimiento de las tareas pendientes, en proceso y finalizadas del equipo de trabajo.

Integración: Uno de los puntos más fuertes de GitHub es la gran cantidad de herramientas externas con las cuales se puede integrar, obteniendo más funcionalidades y al mismo tiempo pone a disposición su robustez a sistemas externos.

Documentación: Mientras se crean nuevas funcionalidades y versiona el código se puede agregar la documentación del proyecto con la creación de *wikis*, esto brinda a los interesados la información necesaria para integrarse al equipo de trabajo, realizar modificaciones o aprender sobre el proyecto.

1.4.2. GITLAB

GitLab se posiciona como un líder a nivel mundial debido a que es un producto que durante los últimos años se ha enfocado en acoplarse y brindar soporte para la implementación de DevOps en los proyectos de *software*, por ser un sistema de código abierto y gratuito con versiones de pago las cuales tiene un mejor soporte y mayores funcionalidades los ha convertido en una de las elecciones favoritas de las empresas de *software* para versionar sus proyectos.

Dentro de la historia relatada en About GitLab. (s. f.), hacen referencia a sus humildes orígenes en el 2011, cuando Dmitriy en su casa en Ucrania pensó en crear una herramienta poderosa que le ayudara a trabajar mejor en equipo mientras desarrollaba *software*. Lo interesante de la historia es que es necesario considerar que en su hogar no tenían agua por tubería, y es increíble el pensar que priorizo la creación de una herramienta para versionamiento de código, que en solucionar la falta de agua en su hogar y evitar la caminata diaria hasta el pozo de agua comunitario.

Las poderosas herramientas que tiene GitLab para la creación de un ciclo de vida entero de DevOps (The DevOps Lifecycle with GitLab. ,s. f.), lo diferencia frente a otras herramientas de versionamiento de código, entre algunas de sus funcionalidades esta:

Gestión: GitLab da métricas de la velocidad del desarrollo del proyecto, estas métricas permiten observar las personas del equipo que tan rápido o lento trabajan o en que tareas han centrado sus esfuerzos durante el tiempo.

Planeación: Por medio de los tableros que tiene GitLab se pueden crear tareas, asignarles tiempos, recursos, encargados y priorizarse para organizar el equipo de trabajo y enfocarlo en las actividades importantes, además estos tableros permiten la visibilidad del proyecto de inicio a fin.

Verificación: La *suit* de utilidades de GitLab poseen la capacidad de integrarse con herramientas de ejecución automática de pruebas para realizar verificaciones sobre el código, estas verificaciones proveen un rápido *feedback* para corregir o mejorar el código.

Seguridad: Con pruebas de tipo *Static Apllication Security Testing* (SAST), *Dynamic Application Security Testing* (DAST) y más, ayudan a la entrega segura de aplicaciones.

Despliegues: GitLab tiene la capacidad de realizar despliegues automáticos del *software* en servidores todo esto sin intervención humana.

Configuración: Kubernetes y GitLab se pueden integrar para la configuración del ambiente necesario para las aplicaciones a desplegar, este tipo de funcionalidad ayudan en el concepto de definir infraestructura por medio de código.

Monitoreo: A la necesidad de conocer que sucede en los proyectos, el monitoreo automático de GitLab brinda métricas para conocer como los cambios en el código impactan los ambientes pre-producitvos y productivos.

1.4.3. Tabla comparativa de GITHUB y GITLAB

Ambos servicios de versionamiento de código tienen características similares, por lo que resulta pertinente resumir sus leves diferencias o completa similitud para tener una

visión más clara de que servicio se puede adaptar mejor a las necesidades de un proyecto en específico, a continuación, se presentan algunas de las principales funcionalidades gratuitas de ambos servicios y como se diferencian entre ellas:

Tabla 2. Comparación entre GitHub y GitLab.

Funcionalidad Gratuita	GitLab	GitHub
Repositorios privados	Si	Si
Numero de colaboradores	Ilimitado	Ilimitado en repositorio público y solo 3 en repositorios privados
Wiki	Si	Si en repositorio público, no en repositorio privado.
Páginas web	Si	Si
Capacidad por proyecto	10GB	1GB
Capacidad por archivo	20MB	100MB

Fuente: elaboración propia.

Como conclusión es aceptable decir que ambos servicios de versionamiento de código posen funcionalidades similares en sus versiones gratuitas y si bien GitLab inicio como un proyecto versionado en la plataforma GitHub, logro separarse y convertirse en un poderoso aliado para el desarrollo de *software* y competidor de Github. Uno de los ejes fundamentales que atrae a los usuarios a utilizar GitLab es la posibilidad de crear un ilimitado número de repositorios tanto privados como públicos y agregar colaboradores sin un límite establecido a diferencia de GitHub, pero esto no disminuye que GitHub es la empresa con más usuarios y repositorios a nivel mundial por ser un pionero en el servicio de versionamiento de código, esto también significa una larga trayectoria y

comunidad que pueden resolver rápidamente las dudas de cómo utilizar las funcionalidades o conectarlas con otras herramientas.

1.5. Herramientas para pruebas unitarias

Con la intención de delimitar el caso de estudio, solo se describirá información relevante de dos herramientas para pruebas unitarias, Junit y TestNg.

1.5.1. JUNIT

Con la información recopilada de la página oficial de JUnit 5. (2019), es factible decir que Junit versión Junit5 es un *framework* para pruebas unitarias a código escrito en el lenguaje de programación JAVA, este funciona por medio de la escritura de palabras claves y *tags* que indican que esta sección del código es parte de una prueba y no es código de funcionalidades de la aplicación, estas pruebas están generalmente asociadas a una funcionalidad en específico, más no, a pruebas de integración entre múltiples funcionalidades

Algunas de las palabras más usadas en las pruebas son:

@Test: Esta palabra clave se agrega antes de la creación de un método, sirve para indicar que el siguiente método es una prueba unitaria de una funcionalidad del sistema.

Assert: La palabra *assert* se utiliza para el llamado de un método del *framework* de Junit5, que permite la validación de un valor y en caso de no ser el valor esperado, la prueba falle

Un ejemplo básico de una prueba unitaria con el uso estas dos palabras es:

Gráfico 9. Ejemplo de una clase y método en Java para ser evaluados.

```
Public class Calculator {
    public int evaluate(String expression){
        int sum = 0;
        for (String summand: expression.split("\\+"))
            sum += Integer.valueOf(summand);
        return sum;
    }
}
```

Fuente: elaboración propia.

En la clase *Calculator* existe un método llamado *evaluate* , este recibe un *String* el cual dividirá cada vez que encuentre el símbolo +, y finalmente retornará la sumatoria de todos los valores divididos.

Gráfico 10. Ejemplo de un *test* en Junit

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class CalculatorTest {
  @Test
  public void evaluatesExpression() {
    Calculator calculator = new Calculator();
    int sum = calculator.evaluate("1+2+3");
    assertEquals(6, sum);
  }
}
```

Fuente: elaboración propia.

La prueba se realiza con la escritura de la palabra clave *@Test* esto indica que el siguiente método es una prueba unitaria, dentro del método se instancia la clase *Calculator* en la que anteriormente se agregó el método *evaluate,* al hacer un llamado del método *evaluate(1+2+3)* y recibir el valor de retorno se valida con el método *assertEquals* que el valor obtenido sea igual a 6.

1.5.2. TestNG

Este *framework* para pruebas de código JAVA, es considerado uno de los principales competidores de Junit y como es descrito en TestNG - Welcome. (2019) su creador se inspiró en Junit para construir un mejor *framework*, porque en su experiencia laboral encontró problemas de ejecución por hilos en Junit y quiso mejorarlo.

Entre algunas de las mejoras que tiene TestNg versión 6.14.3, se encuentran agrupación de pruebas, configuración de ejecución en hilos, manejo seguro de hilos, estas características permiten la configuración flexible de pruebas y la creación de pruebas de manera más rápida y robusta.

Un ejemplo básico de una prueba con el *framework* de TestNg versión 6.14.3 es:

Gráfico 11. Ejemplo de un *test* con TestNg.

```
package example1;

import org.testng.annotations.*;

public class SimpleTest {

 @BeforeClass
 public void setUp() {
   // code that will be invoked when this test is instantiated
 }

 @Test(groups = { "fast" })
 public void aFastTest() {
   System.out.println("Fast test");
 }

 @Test(groups = { "slow" })
 public void aSlowTest() {
    System.out.println("Slow test");
 }

}
```

Fuente: TestNg – Welcome. (2019).

En este caso se usan la anotación *@BeforeClass* para indicar que el siguiente método escrito se ejecutara una vez antes de todas las demás pruebas y se usa la palabra *@Test* como una anotación para que el *framework* comprenda que el siguiente método es una prueba, finalmente la palabra *groups* se utiliza para agrupar la prueba con otras pruebas del grupo "*fast"* o *"slow".*

Aunque Junit actualmente permite la ejecución de pruebas en hilos, TestNg es un *framework* que maneja mejor la configuración de una ejecución de este tipo, optimiza el uso de recursos en *multi-threaded execution test*, lo cual para el proyecto se vuelve en un ahorro de tiempo por la velocidad que simboliza ejecutar múltiples pruebas de forma paralela, con una configuración mínima.

1.5.3. Comparación entre JUNIT y TESTNG

TestNg y Junit son los principales *frameworks* utilizados para prueba unitarias de los sistemas programados utilizando el lenguaje JAVA y brindan funcionalidades similares, por lo que es necesario describir algunas de sus similitudes y leves diferencias para futuras tomas de decisiones, entre estas funcionalidades se encuentra:

Tabla 3. Comparación entre TestNg y Junit 5

FUNCIONALIDADES	TestNG 6.14.3	Junit5
Soporta anotaciones, ejemplo (@)	Si	Si
Prueba de excepciones	Si	Si
Pruebas de agotamiento de tiempo	Si	Si
Ignorar pruebas	Si	Si
Pruebas parametrizadas (valores primitivos)	Si	Si
***Suit* de pruebas**	Si	Si
Grupos de pruebas	Si	No
Pruebas dependientes de otras	Si	No

Pruebas parametrizadas (objetos)	Si	No

Fuente: elaboración propia.

En la comparación anterior se visualiza que TestNg posee una suite de funcionalidades más completa que puede realizar agrupamientos de pruebas, hacer pruebas dependientes de otras y parametrizarlas, que finalmente conllevan a una configuración más robusta de pruebas unitarias y un mayor control sobre la ejecución de estas.

1.6. SONARQUBE

SonarQube es una potente plataforma *open-source* con la capacidad de analizar código de más de 25 diferentes lenguajes de programación, para encontrar vulnerabilidades, malas prácticas de programación y más.

Las principales funcionalidades descritas por Enhance Your Workflow with Continuous Code Quality | SonarQube. (2019) son:

Inspección continua: Brinda una imagen general de la salud del proyecto, donde se incluyen los *bugs* encontrados, las malas prácticas de programación, y posibles errores de código.

Mejoramiento de calidad por etapas: Permite configurar etapas de calidad en las cuales se añaden reglas que se activan durante la revisión del código y reportan si el código es lo suficientemente bueno como para pasar a producción.

Revisión de código en versionamiento: Su integración con plataformas de versionamiento de código permite revisiones de múltiples *branches* en la nube o locales para obtener *feedback* lo más pronto posible después de un cambio.

Profundización en los errores: En la página *Issues* da la información pertinente a los *issues* encontrados, quien los inyecto, cuando y la localización dentro del código.

Información histórica: SonarQube posee la capacidad de brindar información histórica de como la calidad del proyecto en términos de código ha variado durante el tiempo.

SonarQube tiene una extensa base de datos de prácticas de escritura de código aceptadas mundialmente como correctas o mejores, por lo que ejecuta una revisión del código del *software* para encontrar donde no se están cumpliendo estas prácticas, además puede validar si la lógica es demasiado compleja por su escritura y en varios casos proporciona ejemplos de cómo mejorar la escritura.

Con todos los antecedentes expuestos de SonarQube, el proyecto que implementa esta herramienta puede hacer un seguimiento de la calidad de su código, mejorarlo en caso de ser necesario, observar métricas durante el tiempo sobre cómo ha mejorado o empeorado, encontrar los culpables, crear planes de acción para mejorar el proyecto, encontrar errores que no se encuentran a simple vista, cerrar algunas brechas de seguridad, cuenta con la capacidad de integrarse a plataformas de versionamiento de código para dar *feedback* lo más pronto posible y adicionalmente se puede integrar como una herramienta de ejecución automática para los *pipelines* en DevOps.

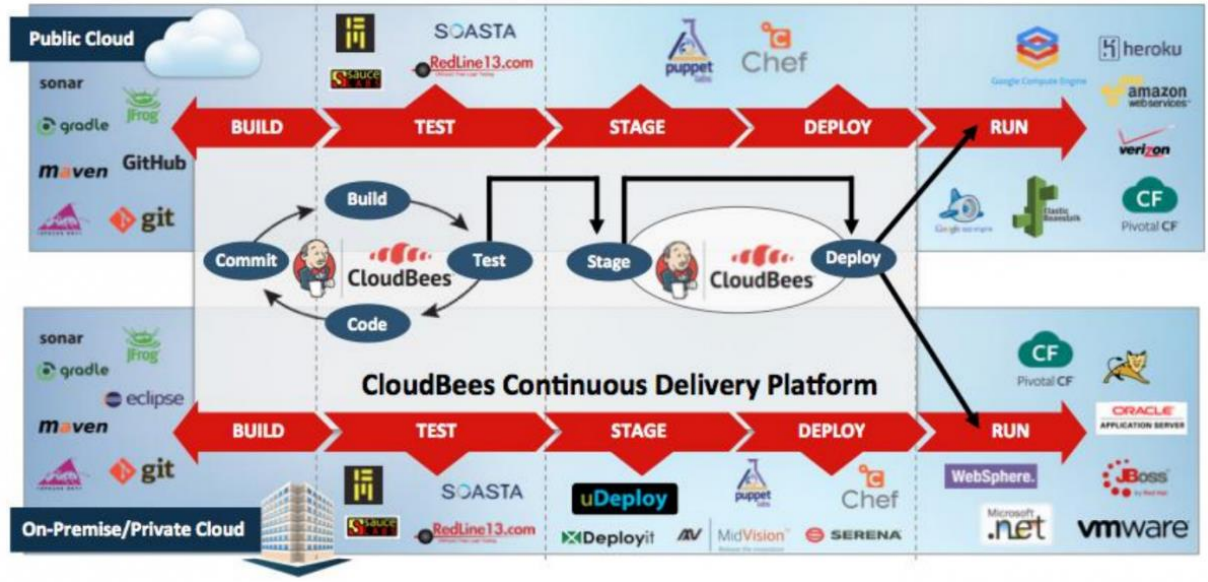
1.7. JENKINS

Jenkins es un servidor de integración continua y como se detalla en Jenkins User Documentation. (s. f.), este es uno de los servidores *open-source* más utilizados a nivel mundial, por la gran cantidad de *plugins* para construcción, despliegue, pruebas y automatización de proyectos de software.

La dos principales características de Jenkins es que a pesar de ser gratuito tiene una inmensa comunidad que brindan soporte y responden dudas de cómo utilizarlo, además posee una gran variedad de *plugins* que pueden integrarse a Jenkins y abren una infinidad de posibilidades para trabajar. Usualmente Jenkins se utiliza como el servidor en el que se construyen los *pipelines* de DevOps para la integración y el

despliegue continuos, por medio de sus *plugins,* se pueden agregar nuevas funcionalidades y conectarse con otras herramientas.

Gráfico 12. Jenkins con algunas de las herramientas utilizadas para DevOps.



Fuente: Continuous Delivery with Jenkins -- Part 2. (2016, abril 27).

CAPITULO II. METODOLOGÍA

En el capítulo en mención esta inicialmente el marco conceptual de las metodologías de investigación teóricas y empíricas, así mismo la forma de empleo en el proyecto y finalmente la información pertinente a la metodología de desarrollo utilizada.

2.1. Métodos Teóricos

Analítico-Sintético: como fue mencionado por el autor Rodríguez Jiménez, A., Jacinto, P., & Omar, A. (2017), se puede entender a este método del modo en que se realiza el proceso de análisis de la información adquirida por medio de fuentes para definir un concepto sintetizado y utilizable.

Este método en particular se utilizó al adquirir información y conocimiento de medios textuales y visuales como son *google academics*, páginas oficiales de reconocidas empresas de *software* y cualquier tipo de información de fuentes previamente comprobadas como confiables, todo esto con la finalidad de ser utilizados en el desarrollo del proyecto.

Inductivo-deductivo: En el texto de Dávila Newman, G. (2006) hace referencia en que parte del proceso inductivo es el análisis de fenómenos o conclusiones particulares a eventos, donde buscan variables comunes y se realiza un proceso lógico de asociación para obtener una conclusión generalizada.

Se utilizo este método de investigación en la revisión de los motivos dentro de las empresas, para no tener un proceso riguroso de calidad, estos hechos se presentaban diferentes unos con otros, pero se logró obtener una idea generalizada del problema.

Análisis-documental: Con los pensamientos recopilados Arias, F. G. (2012), es posible describir el análisis documental como un proceso en el que se recopila, analiza, detalla, interpreta y se realizan consideraciones sobre investigaciones de otros autores para la adquisición y diseño de nuevos conocimientos.

Con el análisis documental se obtuvieron referencias e ideas principales de documentos académicos para su posterior análisis, con la intención de lograr una base de conceptos teóricos en la explicación de las herramientas, metodologías y más.

Además, se realizó la investigación documental durante enero y junio del 2019 en diferentes bases de datos indexadas por *JSTOR Journals, Google Academics, Springer Journals* con filtro de búsqueda de documentación por idioma inglés y español, fechas de publicación entre enero 2007 y junio 2019, para las palabras claves en español como “Ejecución automática de pruebas unitarias por medio de un pipeline en Jenkins”, “Instructivo creación de Job en Jenkins para la ejecución de pruebas unitarias usando TestNg”, “Jenkins pruebas unitarias TestNg Java Maven” , ”pruebas unitarias de TestNg ejecutadas automáticamente desde Jenkins” y en inglés frases o palabras como “Jenkins *unit test* TestNg” , “Jenkins *unit test* Email SonarQube” , “*Instructive pipeline* Jenkins TestNg SonarQube” , ”*Unit Testing Automation* Jenkins TestNg”.

Dentro de los resultados de la búsqueda no se encontraron documentos o proyectos iguales a este, por lo cual se justificó el desarrollo del proyecto por su no existencia.

2.2. Métodos Empíricos

Observación directa: En la definición expuesta por Díaz Sanjuán, L. (2011) refiere a la observación directa como una metodología que permite tomar sucesos vívidos, experiencias y convertirlos en conocimiento aplicable y replicable para el futuro, esto se da porque el investigador conoce adecuadamente las partes involucradas y necesarias del tema en estudio.

Es por ello por lo que la observación directa fue de vital importancia durante el proceso de investigación, porque se tomó como base del tema, la experiencia vivida a nivel empresarial y que actividades se hicieron para mitigar las falencias existentes referentes al proceso de validación de calidad en *software*.

2.3. Metodología Desarrollo

A continuación, se encuentra la información teórica pertinente a la metodología de desarrollo que se utilizó en el proyecto, las fases y ceremonias que se recomiendan seguir.

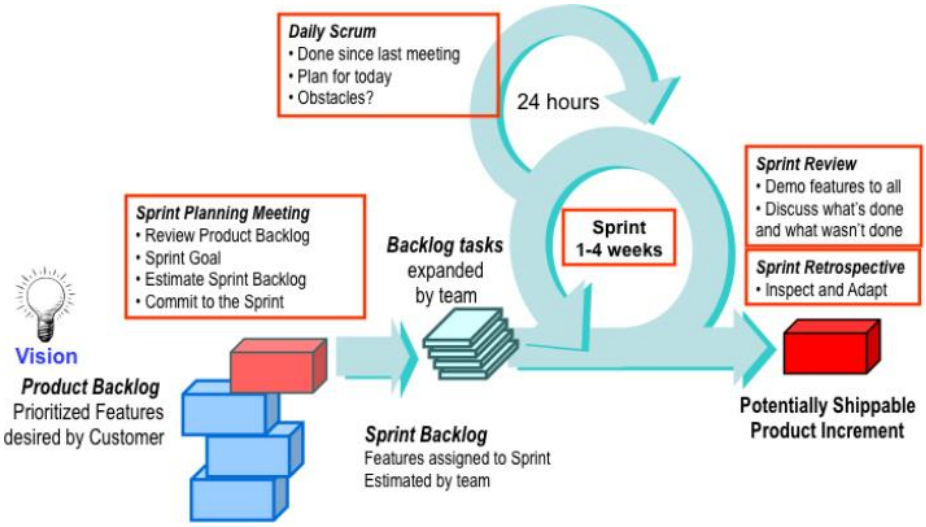
SCRUM

Scrum es definida por Schwaber, K., & Sutherland, J. (2017) como un proceso, técnica de trabajo o un marco de trabajo el cual contiene múltiples actividades e involucrados, los cuales buscan mejorar continua e incrementalmente el producto y el proyecto, además este puede ser aplicado en diferentes ámbitos y ramas de trabajo, gestiona un proyecto y producto, más no, un proyecto de solo un área específica o un producto en particular.

Al observar Scrum desde el punto de metodología de desarrollo de *software* es importante mencionar la entrevista sintetizada de Cohn, M. (2015) donde afirma que es considerada una metodología ágil, está enfocada en la entrega continua e iterativa de producto útil al cliente o interesado.

La metodología Scrum se compone por iteraciones cortas y continuas las cuales tienen ceremonias para el planeamiento de actividades, control y seguimiento que son ejecutadas por las personas del equipo que tienen ciertos roles, esta metodología se puede observar en el gráfico 13.

Gráfico 13. Proceso de Scrum Gráficamente.



Fuente: Sliger, M. (2011).

Las ceremonia, roles y artefactos son:

El Scrum *Master* (Rol): Con la información en mención por Sliger, M. (2011) es posible definir al Scrum *master* como la persona con la obligación de cuidar el equipo, protegerlo y evitar distracciones, al mismo tiempo que remueven obstáculos y negocia con el personal externo al equipo.

El *Product Owner* (Rol): Al tener en cuenta la definición en Acerca del Product Owner.(s/f), el *product owner* es la persona encargada de organizar el *backlog*, priorizar y expresar de manera clara en que se enfocará el esfuerzo el equipo de trabajo

El equipo (Rol): Sliger, M. (2011) define al equipo con un grupo de 5 a 9 personas encargadas y responsables de la creación y entrega del producto.

Dado que el proyecto fue desarrollado por una sola persona, los roles anteriormente detallados no fueron definidos.

Las actividades o ceremonias y artefactos mencionados por Schwaber, K., & Sutherland, J. (2017), que son usados en Scrum, se pueden resumir del siguiente modo:

***Sprint* (Actividad):** Periodo de tiempo comprendido entre 1 a 4 semanas en la cual se realizarán las actividades planeadas para el control y entrega del producto

El *sprint* se definió con periodos de 10 días laborables o dos semanas sin contar los fines de semana, durante este tiempo se realizaron las actividades de desarrollo de historias.

***Planning* (Actividad):** Esta ceremonia se realiza antes de iniciar el *Sprint* y en la cual se planea que se realizará durante el *Sprint*, quién tiene que realizar las actividades, y qué se entregara al final, usualmente dura de 6 a 8 horas.

El *planning* se modificó para ser realizado por una persona y definir las historias de usuario que serían desarrolladas en cada *sprint*.

***Daily* (Ceremonia):** El *daily* son reuniones diarias en las cuales el equipo puede mencionar si tienen bloqueos que impidan el cumplimiento de sus responsabilidades o también para dar información importante para todo el equipo.

Esta ceremonia no se realizó dado el tamaño y cantidad de personas que desarrollaron el proyecto.

***Review* (Ceremonia):** Al finalizar el *Sprint* se realiza esta ceremonia la cual está enfocada en mostrar al *product owner*, los ejecutivos interesados en el producto y al equipo de trabajo, qué se realizó durante el *sprint* y qué no se realizó, cuáles fueron las metas alcanzadas y la reorganización del *backlog* con los productos finalizados y pendientes.

Esta ceremonia no se realizó dado el tamaño y cantidad de personas que desarrollaron el proyecto.

Retrospectiva (Ceremonia): La retrospectiva es una ceremonia que se realiza al finalizar la *review*, con la intención de encontrar oportunidades de mejoras, reconocer qué se hizo mal y qué no se volver repetir en el siguiente *Sprint*, las personas involucradas en esta ceremonia son el equipo de desarrollo y el *Scrum master.*

Como retrospectiva se realizó un *Sprint* completo en el cual se validó la satisfacción de todos los desarrollos realizados en los anteriores *Sprints*.

Backlog **(Artefacto):** Este es el principal artefacto de la metodología Scrum, aquí se encuentran definidas las prioridades, las necesidades y actividades que se van a realizar para cumplir con la entrega del producto, el *backlog* se retroalimente y prioriza cada *sprint* según la necesidad del proyecto.

El artefacto *backlog* se utilizó como una tabla organizativa, en la cual se listaron las historias de usuario que se desarrollaron por *Sprint*.

En conclusión, esta metodología ágil se utilizó para organizar, dividir y desarrollar el proyecto por medio de iteraciones con entregables visibles e incrementales.

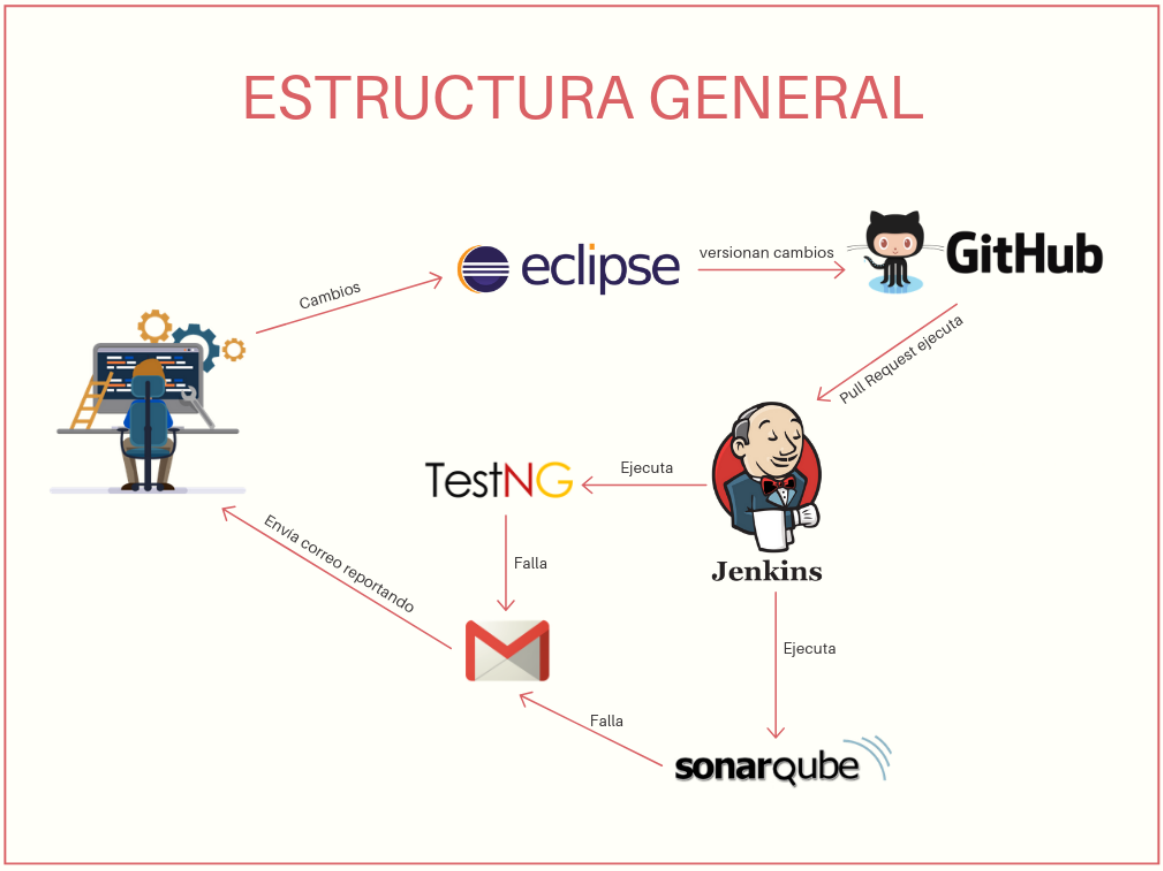
CAPITULO III. RESULTADOS

En el capítulo en cuestión, se detallan los resultados obtenidos de la aplicación de la metodología *Scrum* para el desarrollo del proyecto.

3.1. Creación de *Backlog*

Para el inicio del desarrollo del proyecto se plantearon las historias de usuario necesarias para desarrollar el instructivo a partir de la estructura general esperada.

Gráfico 14. Estructura general



Fuente: elaboración propia.

En el gráfico 14 se muestra como el desarrollador produce y continuamente versiona cambios en un proyecto Java en la rama desarrollo del repositorio de GitHub el cual contiene dos ramas , una llamada master y una llamada desarrollo , al finalizar sus cambios en la rama desarrollo crea un *pull request* contra la rama *master*, esta petición de *pull request* acciona la ejecución del servidor Jenkins el cual por medio de una serie de *plugins* configurados, tiene la capacidad de ejecutar las pruebas unitarias escritas en TestNg del código versionado en la rama desarrollo y de ejecutar la revisión por parte de la herramienta SonarQube, así con estas verificaciones en caso de existir un resultado no esperado en las pruebas de TestNg o no cumplir con la *Quality Gate* de SonarQube, el servidor Jenkins envía un correo informativo alertando los errores a las personas interesadas.

Dado el detalle anterior de la estructura general del proyecto, en la tabla 4 se muestra el *backlog* del proyecto que está compuesto por la historias de usuario propuestas para el desarrollo del proyecto, para la creación del *backlog* se observó el diagrama general del producto final esperado y se dividió en entregables funcionales, priorizados por dependencia de artefactos y estimado en puntos de esfuerzo con la secuencia 0, 0.5, 1, 2, 3, 5, 8, 13, 21, de Fibonacci y como recomienda Restrepo Herrón, A. A., & Gil, J. G. (2018), los valores se asignan de manera incremental según la dificultad de la historia de usuario, además como es usualmente realizado el estimado fue otorgado con base en experiencias anteriores del desarrollador.

Tabla 4. *Backlog*.

ID	Historia de usuario	Estimación
1	Creación de cuenta en *GitHub.com*	1
2	Instalación de *GitHub Desktop*	1
3	Creación de proyecto Java	2
4	Versionamiento de proyecto Java en *GitHub*	2

5	Instalación de *Jenkins*	1
6	Instalación de *plugins GitHub Pull Request, Email Extension Template, SonarQube Scanner y Sonar Quality Gate en Jenkins*	1
7	Instalación de *SonarQube*	1
8	Instalación de *Ngrok*	0,5
9	Configuración de *GitHub Webhook Pull Request*	1
11	Exponer *SonarQube* a Internet	0,5
11	Configuración de *plugins* en *Jenkins*	3
12	Disminuir la seguridad en la cuenta de *Gmail*	0,5
13	Configurar proyecto Java para que *SonarQube* pueda conocer su cobertura	3
14	Configuración del *Job* para la ejecución automática de pruebas unitarias, revisión de *SonarQube* y reporte por medio de correo electrónico.	5
15	Ejecución automática de *Jenkins* al crearse un *Pull request* en *Github*	0,5
	Total de historias de usuario	**15**
	Total de puntos de esfuerzo	**23**

Fuente: elaboración propia.

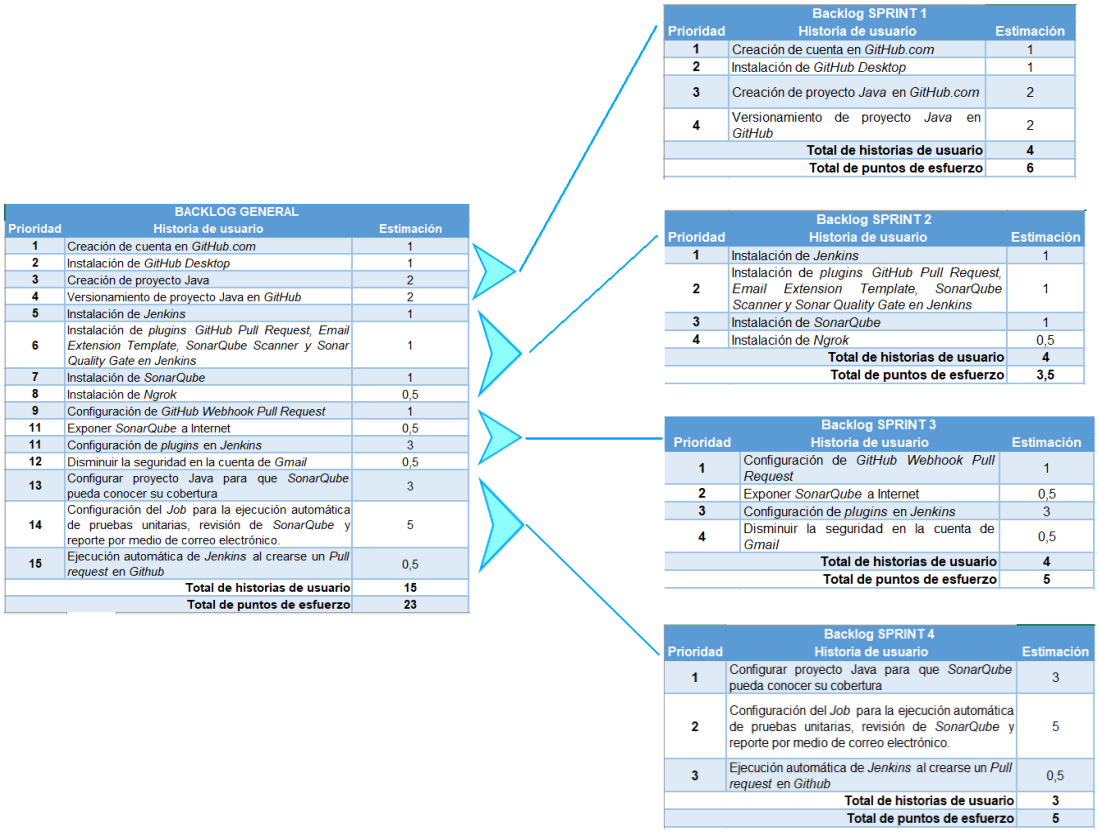
Es importante resaltar que el *backlog* general del proyecto puede variar según los *Sprint* y no representa un listado obligatorio de tareas a cumplir.

3.2. Asignación de historias de usuario a *Sprints*

Las historias de usuario del *Backlog* fueron asignadas a *Sprints* comprendidos por espacios de tiempo de 10 días laborables en los que se desarrollaron las historias.

Para la división de las historias de usuario a desarrollarse por *Sprints*, estas fueron seleccionadas según la prioridad y orden propuestos del *backlog* general, también se tomó en cuenta que la sumatoria de los puntos de esfuerzo de las historias por *Sprint* no superara el valor de 10, por experiencias anteriores del desarrollador, un valor inferior a 10 es prudente para desarrollarse en *Sprints* de esta duración. Con la información anteriormente expuesta, se crearon los *backlogs* de los *Sprints* del siguiente modo.

Gráfico 15. Backlog proyecto en general.



BACKLOG GENERAL		
Prioridad	Historia de usuario	Estimación
1	Creación de cuenta en *GitHub.com*	1
2	Instalación de *GitHub Desktop*	1
3	Creación de proyecto Java	2
4	Versionamiento de proyecto Java en *GitHub*	2
5	Instalación de *Jenkins*	1
6	Instalación de *plugins GitHub Pull Request, Email Extension Template, SonarQube Scanner y Sonar Quality Gate en Jenkins*	1
7	Instalación de *SonarQube*	1
8	Instalación de *Ngrok*	0,5
9	Configuración de *GitHub Webhook Pull Request*	1
11	Exponer *SonarQube* a Internet	0,5
11	Configuración de *plugins* en *Jenkins*	3
12	Disminuir la seguridad en la cuenta de *Gmail*	0,5
13	Configurar proyecto Java para que *SonarQube* pueda conocer su cobertura	3
14	Configuración del *Job* para la ejecución automática de pruebas unitarias, revisión de *SonarQube* y reporte por medio de correo electrónico.	5
15	Ejecución automática de *Jenkins* al crearse un *Pull request* en *Github*	0,5
	Total de historias de usuario	**15**
	Total de puntos de esfuerzo	**23**

Backlog SPRINT 1		
Prioridad	Historia de usuario	Estimación
1	Creación de cuenta en *GitHub.com*	1
2	Instalación de *GitHub Desktop*	1
3	Creación de proyecto *Java* en *GitHub.com*	2
4	Versionamiento de proyecto *Java* en *GitHub*	2
	Total de historias de usuario	**4**
	Total de puntos de esfuerzo	**6**

Backlog SPRINT 2		
Prioridad	Historia de usuario	Estimación
1	Instalación de *Jenkins*	1
2	Instalación de *plugins GitHub Pull Request, Email Extension Template, SonarQube Scanner y Sonar Quality Gate en Jenkins*	1
3	Instalación de *SonarQube*	1
4	Instalación de *Ngrok*	0,5
	Total de historias de usuario	**4**
	Total de puntos de esfuerzo	**3,5**

Backlog SPRINT 3		
Prioridad	Historia de usuario	Estimación
1	Configuración de *GitHub Webhook Pull Request*	1
2	Exponer *SonarQube* a Internet	0,5
3	Configuración de *plugins* en *Jenkins*	3
4	Disminuir la seguridad en la cuenta de *Gmail*	0,5
	Total de historias de usuario	**4**
	Total de puntos de esfuerzo	**5**

Backlog SPRINT 4		
Prioridad	Historia de usuario	Estimación
1	Configurar proyecto Java para que *SonarQube* pueda conocer su cobertura	3
2	Configuración del *Job* para la ejecución automática de pruebas unitarias, revisión de *SonarQube* y reporte por medio de correo electrónico.	5
3	Ejecución automática de *Jenkins* al crearse un *Pull request* en *Github*	0,5
	Total de historias de usuario	**3**
	Total de puntos de esfuerzo	**5**

Fuente: elaboración propia.

3.3. *SPRINT* 1

Para el *Sprint* 1 se desarrollaron las historias pertenecientes a la creación de una cuenta en la plataforma GitHub para versionar el futuro código del proyecto, la instalación de la herramienta GitHub desktop para versionar código por medio de un programa con *Front-end* en la cuenta anteriormente creada de GitHub, adicionalmente se desarrolló el proyecto base en Java que sirvió para demostrar la ejecución automática de las pruebas unitarias de la estructura creada, este proyecto Java se desarrolló con el programa Eclipse como IDE para programar, Maven para el manejo de dependencias y TestNg para la creación de pruebas unitarias sobre los métodos públicos, finalmente se versiono el código del proyecto Java creado, en la cuenta GitHub con la herramienta GitHub *Desktop*, esta información se puede observar de manera resumida en el siguiente gráfico.

Gráfico 16. Código Java y Github.

Fuente: elaboración propia.

3.3.1. *Backlog Sprint* 1

Después del proceso de revisión y priorización del *backlog*, se seleccionaron las siguientes historias para ser desarrolladas durante este *sprint*.

Tabla 5. *Backlog Sprint* 1.

Prioridad	Historia de usuario	Estimación
1	Creación de cuenta en *GitHub.com*	1
2	Instalación de *GitHub Desktop*	1
3	Creación de proyecto *Java*	2
4	Versionamiento de proyecto *Java* en *GitHub*	2
	Total de historias de usuario	**4**
	Total de puntos de esfuerzo	**6**

Fuente: elaboración propia.

3.3.2. Creación de cuenta en GitHub

La historia de usuario de creación de una cuenta en GitHub permitió describir e ilustrar los pasos necesarios para que el lector del instructivo posea una cuenta que pueda ser usada para el versionamiento de código del proyecto Java. La información acerca de la historia de usuario se detalla en la siguiente tabla.

Tabla 6. Detalle crear cuenta de GitHub

TITULO	**Creación de cuenta en GitHub.**
Descripción de Historia de usuario	Al lector del instructivo le surge la necesidad de conocer el proceso de creación de una cuenta gratuita en GitHub para poder realizar el versionamiento de código de un proyecto Java.
Encargado	Desarrollador.
Puntos de esfuerzo	1
Proceso de desarrollo	Conforme a lo solicitado en la historia de usuario se siguieron y detallaron lo pasos necesarios para la creación de una cuenta gratuita en GitHub usando una cuenta de Gmail, para su desarrollo se utilizó el método de observación directa de la información expuesta en Github.com. La ilustración y detalle de los pasos están en el anexo 1 sección 2.

Fuente: elaboración propia.

3.3.3. Instalación de GitHub *Desktop*

La Instalación de GiHub Desktop permitió el uso de la herramienta para versionar código evitando el uso de comandos por consola y facilitando la interacción entre el interesado en versionar código y la plataforma GitHub. La información acerca de la historia de usuario se detalla en la siguiente tabla.

Tabla 7. Detalle Instalación GitHub *Desktop*

TITULO	**Instalación de GitHub *Desktop*.**
Descripción de Historia de usuario	Se propuso el uso de una herramienta grafica para versionar código en la plataforma Github evitando el uso de comandos por consola.
Encargado	Desarrollador.
Puntos de esfuerzo	1
Proceso de desarrollo	Para cumplir con la historia de usuario solicitada se documentaron los pasos necesarios para la instalación de GitHub *Desktop* tomando como base la información detallada en https://help.github.com/en/desktop/getting-started-with-github-desktop/installing-github-desktop . La ilustración y detalle de los pasos se pueden observar en el anexo 1 sección 3.

Fuente: elaboración propia.

3.3.4. Creación de proyecto Java

El proyecto Java es un programa básico el cual posee varias clases con métodos públicos los cuales sirvieron de ejemplo para crear las pruebas unitarias que la estructura de herramientas ejecutó para observar su funcionamiento, este proyecto Java fue desarrollado bajo el *framework* Maven para el manejo de dependencias y TestNg como *framework* para la escritura de las pruebas unitarias. El programa creado es un eje inicial del proyecto, a partir de las modificaciones de código que sufre y su versionamiento, puede o no desencadenar las revisiones del sistema de herramientas que ejecuta las pruebas unitarias de manera automática. El detalle de la historia de usuario se encuentra en la tabla siguiente.

Tabla 8. Detalle proyecto Java.

TITULO	**Creación de proyecto Java.**
Descripción de Historia de usuario	La creación de un proyecto básico en Java es obligatoria dado que el sistema tiene que ejecutar las pruebas unitarias de manera automática sobre algún programa, para la creación del programa se recomienda el uso de Maven para mejorar el manejo de dependencias y la herramienta TestNg para el desarrollo de las pruebas unitarias.
Encargado	Desarrollador.
Puntos de esfuerzo	2
Proceso de desarrollo	Se creo una estructura básica de un programa Java utilizando el IDE Eclipse para su programación, además del *framework* Maven para el manejo de dependencias y TestNg para la creación de pruebas unitarias. Se tomo en cuenta la importancia de que el proyecto posea clases con métodos claramente definidos, entendibles y *testeables* para que el lector pueda implementarlos

	fácilmente. la ilustración y detalle de los pasos se pueden observar en el anexo 1 sección 4.

Fuente: elaboración propia.

3.3.5. Versionamiento de proyecto Java en GitHub

El versionamiento del proyecto Java en GitHub permitió a Jenkins recuperar el código donde se encuentran las pruebas unitarias que ejecutará automáticamente.

Tabla 9. Detalle GitHub y Java

TITULO	**Versionamiento de proyecto Java en GitHub.**
Descripción de Historia de usuario	El programa Java que se creó anteriormente es versionado utilizando el *software* GitHub Desktop y la cuenta de GitHub, también es necesario que se cree una nueva rama en el repositorio de GitHub, de este modo el programa se encontrará en la nube y será accesible para Jenkins, se tendrán respaldos de diferentes versiones del sistema y además podrá ser modificado por uno o más desarrolladores de manera simultánea.
Encargado	Desarrollador.
Puntos de esfuerzo	2

Proceso de desarrollo	Se realizo el versionamiento del proyecto Java en la cuenta de GitHub creada anteriormente y con la información encontrada y analizada de https://help.github.com/en/desktop/getting-started-with-github-desktop se utilizó el software GitHub Desktop para realizar el versionamiento y creación de una nueva rama llamada desarrollo, como consiguiente se ilustro y detallo los pasos necesarios para este proceso y que pueden ser observados en el anexo 1 sección 5

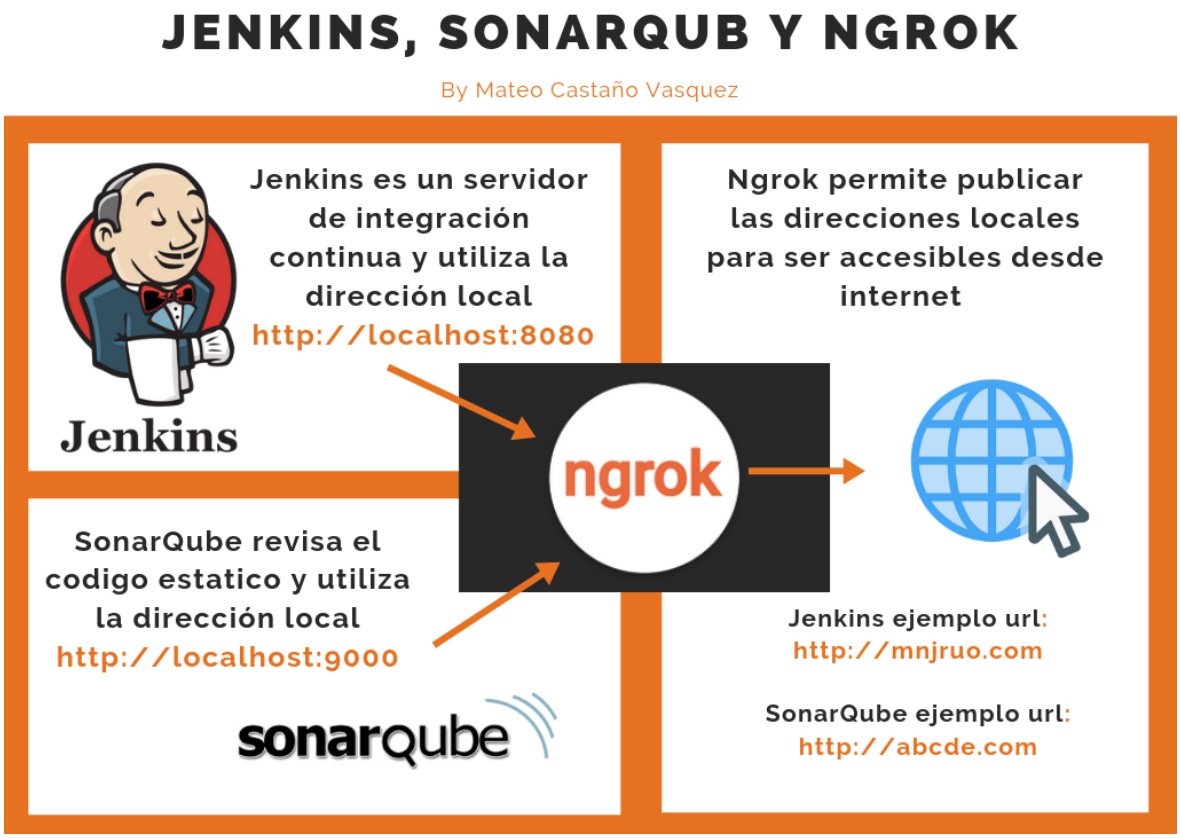
Fuente: elaboración propia.

3.4. *SPRINT* 2

En el *Sprint* 2 se desarrollaron las historias de instalación del servidor Jenkins el cual se encarga de conectarse con GitHub para descargar el código fuente del proyecto Java creado anteriormente, la ejecución de las pruebas unitarias sobre el código de manera automática al detectar un *pull request* en GitHub, la ejecución de SonarQube para la revisión de código estático y el envío de correos si las pruebas unitarias fallan o la *Quality Gate* de SonarQube no es superada, todas estas tareas las realiza por medio de plugins los cuales fueron instalados como parte de una historia de usuario de este sprint, además de la instalación de SonarQube.

Dado que el código del proyecto Java se encuentra versionado en GitHub, la cual es una herramienta en la nube y Jenkins como SonarQube se encuentran instaladas en una maquina local y usan *Urls* como http://localhost:8080, se desarrolló la historia de usuario para la instalación de la herramienta Ngrok, que tiene la capacidad de realizar un puente entre las *Url* locales y publicarlas en internet para que pueda existir comunicación entre herramientas en la nube y herramientas locales, tal y como se puede observar en el gráfico 17.

Gráfico 17. Jenkins, SonarQube y Ngrok separados



Fuente: elaboración propia.

3.4.1. *Backlog Sprint* 2

Después del proceso de revisión y priorización del *backlog*, se seleccionaron las siguientes historias para ser desarrolladas durante este *sprint*.

Tabla 10. *Backlog Sprint* 2.

Prioridad	Historia de usuario	Estimación
1	Instalación de *Jenkins*	1
2	Instalación de *plugins GitHub Pull Request, Email Extension Template, SonarQube Scanner y Sonar Quality Gate en Jenkins*	1
3	Instalación de *SonarQube*	1
4	Instalación de *Ngrok*	0,5
	Total de historias de usuario	**4**
	Total de puntos de esfuerzo	**3,5**

Fuente: elaboración propia.

3.4.2. Instalación de Jenkins

La instalación de Jenkins sirvió para crear una de las bases del proyecto, este servidor se encarga de la ejecución automática de las pruebas unitarias del código, también de la ejecución de SonarQube para revisión de código y además el reporte de errores en caso de suceder, en la tabla 11 se detalla el desarrollo de esta historia de usuario.

Tabla 11. Detalle instalación Jenkins

TITULO	Instalación de Jenkins.
Descripción de Historia de usuario	Existe la necesidad por parte del lector de conocer el proceso de instalación del servidor Jenkins en una maquina local con sistema operativo Windows, con este servidor ejecutara automáticamente las pruebas unitarias del código del proyecto Java creado anteriormente.
Encargado	Desarrollador.
Puntos de esfuerzo	1
Proceso de desarrollo	Para el desarrollo de la historia de usuario en cuestión, se investigó la descarga e instalación de Jenkins en una maquina local con sistema operativo Windows tomando como base la información expuesta en https://wiki.jenkins.io/display/JENKINS/Thanks+for+using+Windows+Installer , además se ilustro el paso a paso para cumplir con la meta de instalar Jenkins tal y como se muestra en el anexo 1 sección 6.

Fuente: elaboración propia.

3.4.3. Instalación de *plugins* GitHub *Pull Request*, *Email Extension Template*, SonarQube *Scanner* y Sonar Quality *Gate* en Jenkins

El servidor Jenkins se conectara a diferentes herramientas para realizar el proceso deseado en este proyecto y para esto se instalaron los *plugins* necesarios que permitirán la conexión con GitHub y la notificación de la creación de *pull request* en el proyecto versionado, también se podrá comunicar con SonarQube para iniciar el escaneo del código estático por lo que se instaló SonarQube *Scanner* y *Sonar Quality Gate*, adicionalmente se instaló el *plugin Email Extension Template* para el envío de

correos electrónicos cuando el proyecto falle. En la tabla 12 se detalla más información sobre el desarrollo.

Tabla 12. Instalación *plugins*.

TITULO	**Instalación de *plugins GitHub Pull Request, Email Extension Template, SonarQube Scanner* y *Sonar Quality Gate* en Jenkins**
Descripción de Historia de usuario	El servidor Jenkins realizara conexiones con otras herramientas por lo cual el lector tiene que conocer los *plugins* necesarios para poder enviar correos electrónicos, iniciar la ejecución de SonarQube y realizar conexión con un proyecto versionado de GitHub cada vez que un *pull request* sea creado.
Encargado	Desarrollador.
Puntos de esfuerzo	1
Proceso de desarrollo	En el anexo 1 sección 7 se detalla la instalación de los *plugins* deseados y para su desarrollo se utilizó como material de apoyo la información expuesta sobre los *plugins* en la página https://plugins.jenkins.io/

Fuente: elaboración propia.

3.4.4. Instalación de SonarQube

El *software* SonarQube se instaló para ser utilizado en el análisis de código estático del proyecto Java versionado en GitHub, la ejecución de SonarQube será disparada por Jenkins como parte de la ejecución automática de las pruebas unitarias. En la tabla 13 se puede observar con mayor detalle el proceso de desarrollo de la historia de usuario.

Tabla 13. Detalle instalación SonarQube.

TITULO	Instalación de SonarQube
Descripción de Historia de usuario	Se propone como complemento para el lector, el desarrollo de una historia de usuario en la cual se realice la instalación de SonarQube en un maquina local con sistema operativo Windows, por motivos de que este *software* se utilizará como una herramienta adicional para la revisión de código estático y estará conectada a Jenkins en la ejecución de las pruebas unitarias de manera automática.
Encargado	Desarrollador.
Puntos de esfuerzo	1
Proceso de desarrollo	Tal y como fue solicitado en la historia de usuario, se ilustro y detallo el proceso de descarga e instalación del *software* SonarQube en una maquina local con sistema operativo Windows y esto se puede observar en el anexo 1 sección 8. Parte de su desarrollo se realizó por medio de la investigación de información sobre SonarQube y su instalación, y se utilizó como guía lo redactado en el siguiente *link* *https://docs.sonarqube.org/latest/setup/get-started-2-minutes/*

Fuente: elaboración propia.

3.4.5. Instalación de *Ngrok*

La instalación de Ngrok se realizó para exponer la *Url* local de Jenkins y SonarQube a internet y ser visibles para herramientas en la nube como GitHub. Parte del desarrollo se puede observar en la tabla 14.

Tabla 14. Detalle instalación Ngrok.

TITULO	Instalación de Ngrok
Descripción de Historia de usuario	El lector necesita realizar una conexión entre GitHub y Jenkins, pero el sistema Jenkins se encuentra instalado de manera local y no es accesible por parte de GitHub, por lo que se propone utilizar el *software* Ngrok para publicar temporalmente el servidor Jenkins a internet y que se visible por parte de GitHub.
Encargado	Desarrollador.
Puntos de esfuerzo	0,5
Proceso de desarrollo	Con las indicaciones proporcionadas en la historia de usuario se realizó la investigación del proceso de descarga, instalación y uso , se utilizó la información encontrada en https://ngrok.com/product como base del proceso de ilustración y descripción que puede observarse en anexo 1 sección 9.

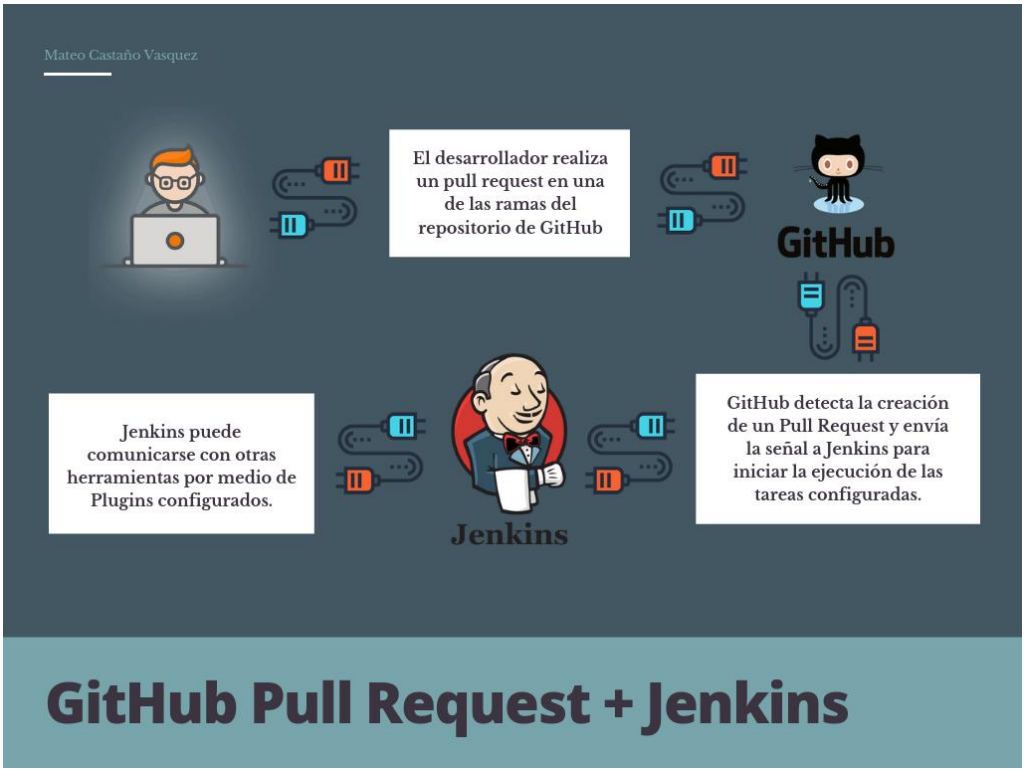
Fuente: elaboración propia.

3.5. *SPRINT* 3

El *Sprint* 3 se enfocó principalmente en la configuración de algunas de las herramientas previamente instaladas, tal y como se describe en el gráfico 18 , se configuraron los *plugins* instalados en Jenkins para que este pueda comunicarse con GitHub, SonarQube y enviar correos electrónicos de manera correcta, adicionalmente se disminuyó la seguridad de la cuenta de correo Gmail utilizada por Jenkins para así habilitar el envío de correos y se expuso el *software* Jenkins y SonarQube a internet para lograr configurar correctamente en GitHub la opción de GitHub *pull request* que

será la encargada de enviar la señal de ejecución de Jenkins cada vez que se cree un nuevo *pull request* en GitHub.

Gráfico 18. Jenkins y GitHub *webhook Pull Request* conectados.



Fuente: elaboración propia.

3.5.1. *Backlog Sprint* 3

Después del proceso de revisión y priorización del backlog, se seleccionaron las siguientes historias para ser desarrolladas durante este sprint.

Tabla 15. *Backlog Sprint* 3.

Prioridad	Historia de usuario	Estimación
1	Configuración de *GitHub Webhook Pull Request*	1
2	Exponer *SonarQube* a Internet	0,5

3	Configuración de *plugins* en *Jenkins*	3
4	Disminuir la seguridad en la cuenta de *Gmail*	0,5
	Total de historias de usuario	**4**
	Total de puntos de esfuerzo	**5**

Fuente: elaboración propia.

3.5.2. Configuración de GitHub *Webhook Pull Request*

La opción *GitHub Webhook Pull Request* permite a GitHub notificar a Jenkins sobre la creación de nuevos Pull Request dentro de sus ramas, lo que resultó sumamente necesario para que Jenkins conozca en qué momento iniciar la ejecución de las pruebas unitarias del código versionado en GitHub. En la tabla 16 se encuentra un mayor detalle de la historia de usuario y su detalle.

Tabla 16. Configuración Webhook pull request.

TITULO	**Configuración de GitHub Webhook Pull Request**
Descripción de Historia de usuario	Se sugiere la ilustración y descripción de los pasos necesarios para la configuración de la opción GitHub *Webhook Pull Request*, así se notifica Jenkins sobre la creación de nuevos *Pull Request* en GitHub y puede desencadenar la ejecución automática de pruebas unitarias del proyecto versionado, adicionalmente para el desarrollo de la historia de usuario es necesario tener en cuenta que mientras Jenkins se encuentre con una *Url* local, GitHub no podrá comunicarse con él.
Encargado	Desarrollador.

Puntos de esfuerzo	1
Proceso de desarrollo	El desarrollo de la historia de usuario se dio con la exposición del Jenkins local a internet por medio de la herramienta Ngrok usando la información expuesta en https://ngrok.com/docs y después configurando la opción GitHub *Webhook Pull Request* con base a lo mencionado en https://developer.github.com/webhooks/ , finalmente se ilustraron y describieron los pasos para esta configuración, tal y como se observa en el anexo 1 sección 10.

Fuente: elaboración propia.

3.5.3. Exponer *SonarQube* a Internet

La historia de usuario se creó con motivo de que el *software* SonarQube se instaló previamente en una máquina local y posee la *Url* http://localhost:9000 la cual no es accesible desde GitHub y tampoco su reporte es visible al enviar esta *Url* en un correo electrónico, con estos antecedentes se dio la necesidad de exponer SonarQube a Internet con ayuda del *software* Ngrok. La historia de usuario se encuentra más detallada en la tabla 17.

Tabla 17. Detalle SonarQube en internet.

TITULO	**Exponer SonarQube a Internet**
Descripción de Historia de usuario	El *software* SonarQube se encuentra localmente instalado y las herramientas en la nube no tiene acceso al mismo se propone la ilustración y descripción de los pasos necesario para exponer temporalmente SonarQube al Internet por medio de la herramienta Ngrok.

Encargado	Desarrollador.
Puntos de esfuerzo	0,5
Proceso de desarrollo	La información expuesta en https://ngrok.com/docs fue base para el desarrollo de la historia de usuario propuesta , finalmente ilustrando y detallando los pasos solicitados, tal y como se puede ver en el anexo 1 sección 11.

Fuente: elaboración propia.

3.5.4. Configuración de *plugins* en Jenkins

En una historia de usuario previamente desarrollada fueron instalados los *plugins* necesarios para la correcta conexión entre herramientas y Jenkins, por lo cual durante esta historia de usuario los *plugins GitHub Pull Request, Email Extension Template, SonarQube Scanner y Sonar Quality Gate* fueron configurados. El detalle de la historia de usuario se puede observar en la tabla 18.

Tabla 18. Detalle configuración *plugins*.

TITULO	**Configuración de plugins en Jenkins**
Descripción de Historia de usuario	Se cree necesaria el detallar e ilustrar la configuración necesaria de los plugins *GitHub Pull Request, Email Extension Template, SonarQube Scanner y Sonar Quality Gate* instalados previamente, para así complementar la información previamente redactada. Con esta configuración se espera que Jenkins pueda realizar la conexión con GitHub *webhook pull request* , SonarQube y el envío de correos electrónicos.
Encargado	Desarrollador.

Puntos de esfuerzo	3
Proceso de desarrollo	La historia de usuario se desarrolló ilustrando los pasos y configuración necesaria para que los *plugins* previamente instalados funcionen correctamente, estas configuraciones fueron investigadas en fuentes como https://docs.sonarqube.org/display/SCAN/Analyzing+with+SonarQube+Scanner+for+Jenkins , https://plugins.jenkins.io/ghprb , https://plugins.jenkins.io/sonar-quality-gates y https://plugins.jenkins.io/emailext-template siendo esta información publicada por los propios autores de los *plugins* y Jenkins, además en el anexo 1 sección 12 se puede observar el resultado del desarrollo de esta historia de usuario.

Fuente: elaboración propia.

3.5.5. Disminuir la seguridad en la cuenta de Gmail

La configuración de SMTP para el envío de correos con el *plugin Email Extension Template* instalado previamente, necesita de una cuenta de correo que sirva como emisor y en este proyecto se utilizó una cuenta de Gmail, la cual será configurada y disminuida su seguridad para ser utilizada con este propósito. El detalle de la historia de usuario se puede observar en la tabla 19.

Tabla 19. Detalle disminución seguridad Gmail.

TITULO	Disminuir la seguridad en la cuenta de *Gmail*
Descripción de Historia de usuario	Se recomienda detallar los pasos necesarios para disminuir la seguridad de una cuenta de Gmail, no disminuirla puede generar bloqueos en el envío de correos electrónicos desde Jenkins al utilizar una cuenta de Gmail como emisor de correos, es importante recalcar que el envío de correos es necesario para la notificación de errores.
Encargado	Desarrollador.
Puntos de esfuerzo	0,5
Proceso de desarrollo	La historia de usuario se desarrolló siguiendo la información encontrada en https://support.google.com/accounts/answer/6010255?hl=en y detallando de menara visual los pasos necesarios para disminuir la seguridad de una cuenta de Gmail, estas ilustraciones se pueden observar en el anexo 1 sección 13.

Fuente: elaboración propia.

3.6. *SPRINT* 4

Los anteriores *Sprints* fueron marcados con historias de usuario enfocadas en la configuración y preparación de las herramientas y ambiente de trabajo, por lo tanto, en este *Sprint* se realizó la interconexión entre las herramientas y una ejecución practica del conjunto de herramientas para así lograr observar el funcionamiento del proyecto como un todo.

3.6.1. *Backlog Sprint* 4

Después del proceso de revisión y priorización del *backlog*, se seleccionaron las siguientes historias para ser desarrolladas durante este *sprint*.

Tabla 20. *Backlog Sprint* 4.

Prioridad	Historia de usuario	Estimación
1	Configurar proyecto Java para que *SonarQube* pueda conocer su cobertura	3
2	Configuración del *Job* para la ejecución automática de pruebas unitarias, revisión de *SonarQube* y reporte por medio de correo electrónico.	5
3	Ejecución automática de *Jenkins* al crearse un *Pull request* en *Github*	0,5
	Total de historias de usuario	**3**
	Total de puntos de esfuerzo	**5**

Fuente: elaboración propia.

3.6.2. Configurar proyecto Java para que SonarQube pueda conocer su porcentaje de cobertura

El código del proyecto Java posee un porcentaje de cubrimiento en pruebas unitarias, y este porcentaje proporciona información importante del estado del proyecto, con los antecedentes mencionados, se creó una historia de usuario en la que se configuró el proyecto Java para poder encontrar ese porcentaje y aprovecharlo. En la tabla 21 se detalla la historia de usuario.

Tabla 21. Detalle Java y SonarQube.

TITULO	Configurar proyecto Java para que SonarQube pueda conocer su cobertura
Descripción de Historia de usuario	El *software* SonarQube posee una métrica llamada *Quality Gate* la cual tiene configurada entre sus reglas el porcentaje de cubrimiento del código del proyecto, para hacer uso de esta métrica, se recomienda ilustrar y detallar los pasos necesarios para agregar el *plugin* JaCoCo en el proyecto Java, dado que este *plugin* tiene la capacidad de generar un reporte que es leído por SonarQube.
Encargado	Desarrollador.
Puntos de esfuerzo	3
Proceso de desarrollo	La historia de usuario se desarrolló con parte de la información disponible en https://docs.sonarqube.org/display/PLUG/Usage+of+JaCoCo+with+SonarJava, después de esto se procedió a realizar el detalle e ilustración de las modificaciones necesarias para usar este *plugin* en el proyecto creado anteriormente en Java. El resultado de la historia de usuario se puede encontrar en el anexo 1 sección 14.

Fuente: elaboración propia.

3.6.3. Configuración del Job para la ejecución automática de pruebas unitarias, revisión de SonarQube y reporte por medio de correo electrónico

La historia de usuario en mención fue la unión de todas las configuraciones y herramientas anteriormente instaladas, se realizó la creación de un *Job* en Jenkins el cual cada vez que se crea un *pull request* en una rama especifica del proyecto

versionado en GitHub, inicia el proceso de ejecución automática de pruebas unitarias sobre el código versionado, valida la *Quality Gate* con la métrica de cobertura de código en SonarQube y notifica por correo electrónico a los interesados en caso de un fallo.

Tabla 22. Detalle creación de Job.

TITULO	**Configuración del *Job* para la ejecución automática de pruebas unitarias, revisión de *SonarQube* y reporte por medio de correo electrónico.**
Descripción de Historia de usuario	El lector del instructivo tiene la necesidad de conocer la creación de un *Job* en Jenkins el cual ejecute automáticamente las pruebas unitarias sobre el código versionado, valide las métricas configuradas en SonarQube y notifique fallas cada vez que falle la ejecución en Jenkins, de este modo se unirían finalmente todas las configuraciones de herramientas previamente realizadas.
Encargado	Desarrollador.
Puntos de esfuerzo	5
Proceso de desarrollo	El desarrollo de esta historia de usuario se dio por medio del análisis y síntesis de la información obtenida en las historias de usuario previamente desarrolladas, concluyendo con la ilustración y detalle de los pasos necesarios para la configuración del *Job* en Jenkins el cual cumpla con los requerimientos solicitados, el producto final de esta historia de usuario se puede encontrar en el anexo 1 sección 15.

Fuente: elaboración propia.

3.6.4. Ejecución automática de Jenkins al crearse un *Pull request* en Github

Por medio de esta historia de usuario se realizó una prueba práctica sobre el producto final del instructivo, con la que el lector pudo observar el funcionamiento del sistema. El detalle de la historia de usuario se puede encontrar en la tabla 23.

Tabla 23. Detalle ejecución automática de Jenkins.

TITULO	**Ejecución automática de Jenkins al crearse un *Pull request* en Github.**
Descripción de Historia de usuario	Es necesario para el lector del instructivo, observar como el sistema se ejecuta las pruebas unitarias de manera automática al crear un *pull request* en GitHub, por lo que se solicita la ilustración y detalle de los pasos necesarios para esta tarea.
Encargado	Desarrollador.
Puntos de esfuerzo	0,5
Proceso de desarrollo	Siguiendo la petición de la historia de usuario se detalló lo solicitado y se utilizó en parte la información expuesta en https://help.github.com/en/articles/creating-a-pull-request para crear un *pull request* con GitHub *Desktop* en la rama desarrollo del proyecto versionado en GitHub que finalmente desencadenara la ejecución del *Job* de Jenkins y permitirá observar los resultados arrojados. El producto terminado de esta historia de usuario se puede encontrar en el anexo 1 sección 16.

Fuente: elaboración propia.

3.7. *SPRINT* 5

Durante el *Sprint* 5 se realizó la validación del producto final del proyecto con la técnica V.A iadov que permitió obtener un indicador de satisfacción.

La técnica V.A iadov consiste en realizar 2 preguntas abiertas y 3 cerradas a los interesados, donde las 3 preguntas cerradas proporcionan después de su tabulación un Índice de Satisfacción Grupal, mientras que las 2 preguntas abiertas brindan la información necesaria de causas o motivos para la satisfacción o insatisfacción sobre el producto en cuestión.

Las tres preguntas cerradas se relacionan por medio del cuadro lógico propuesto por Kuzmina, N. V. (1970), en el cual según la respuesta obtenida por las preguntas se ubican en una casilla específica, de este modo la última pregunta proporciona un valor con el cual se calculará el ISG (Índice de Satisfacción Grupal). Para entender mejor este proceso se puede ejemplificar del siguiente modo:

Pregunta 1. Posee respuesta SI, NO, NO SE
Pregunta 2. Posee respuesta SI, NO, NO SE
Pregunta 3. Tiene una escala de respuestas similares a la siguiente:

- Me gusta mucho
- Me gusta lo suficiente
- Ni me gusta ni me disgusta
- Me disgusta más de lo que me gusta
- No me gusta
- No tengo una opinión.

Según la respuesta de la pregunta esta se ubican en el Cuadro Lógico de Iadov del siguiente modo:

- Respuesta pregunta 1: No
- Respuesta pregunta 2: Si
- Respuesta pregunta 3: Me gusta lo suficiente

Tabla 24. Cuadro lógico iadov.

	PREGUNTA 1								
	NO			NO SÉ			SI		
	PREGUNTA 2								
PREGUNTA 3	Si	No sé	No	Si	No sé	No	Si	No sé	No
Me gusta mucho	1	2	6	2	2	6	6	6	6
Me gusta lo suficiente	2	2	3	2	3	3	6	3	6
Ni me gusta ni me disgusta	3	3	3	3	3	3	3	3	3
Me disgusta más de lo que me gusta	6	3	6	3	4	4	3	4	4
No me gusta	6	6	6	6	4	4	6	4	5
No tengo una opinión.	2	3	6	3	3	3	6	3	4

Fuente: elaboración propia.

La tabla se llenó teniendo en cuenta que la pregunta 1 poseía la respuesta “NO” por lo que la siguiente respuesta se ubicara en ese bloque, luego la pregunta 2 tenía la respuesta “SI” entonces la siguiente respuesta se ubicara en esa columna, finalmente la pregunta 3 tenía la respuesta “Me gusta lo suficiente” por lo que se ubica en la celda correspondiente a la columna de la respuesta anterior y esta celda posee un valor numérico “2” que pertenece a la escala de satisfacción propuesta en la técnica iadov y detallada en la tabla 25.

Tabla 25. Equivalencia de valores en cuadro lógico iadov.

Valor de cuadro iadov	Satisfacción	Valor de satisfacción
1	Clara satisfacción	1
2	**Mas satisfecho que insatisfecho**	**0.5**
3	No definida	0

4	Mas insatisfecho que satisfecho	-0.5
5	Clara insatisfacción	-1
6	Contradictoria	0

Fuente: elaboración propia.

El número 2 obtenido al finalizar las tres preguntas representan un valor de 0.5 de satisfacción que es usado en la fórmula de ISG.

$$ISG = \frac{A(1) + B\ (0.5) + C(0) + D(-0.5) + E(-1)}{N}$$

Esta fórmula posee las variables A, B, C, D, E, que representan el total de personas que respondieron las preguntas y cuyo valor en el cuadro lógico iadov era 1, 2, 3, 4, 5, 6, por ejemplo, en el caso de que 10 personas respondieran el cuestionario y se obtuviesen los siguientes resultados:

Tabla 26. Ejemplo de equivalencia en cuadro lógico iadov.

Personas	Valor del cuadro lógico iadov	valor de satisfacción
6	1	1
2	2	0.5
2	4	-0.5

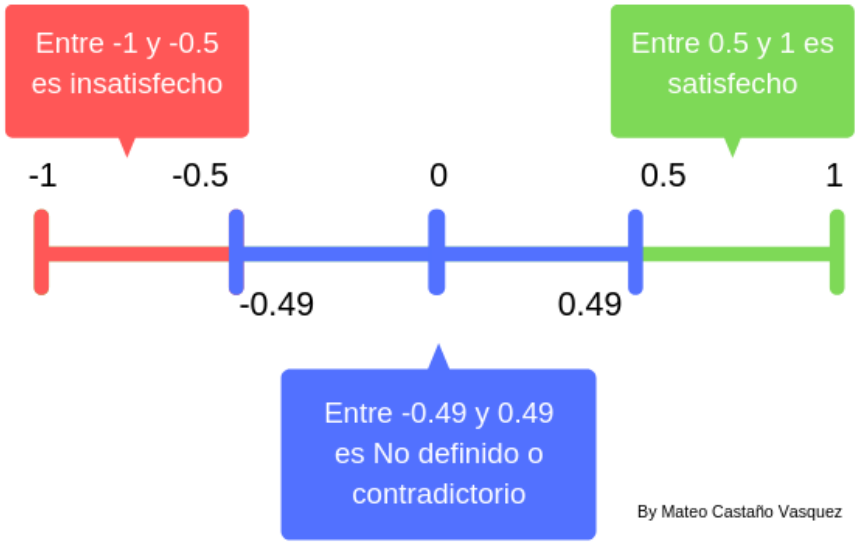
Fuente: elaboración propia.

Al remplazar las variables, teniendo en cuenta que N es el total de personas que respondieron el cuestionario y los datos expuestos en el cuadro anterior la fórmula se observaría así:

$$ISG = \frac{6(1) + 2\ (0.5) + 0(0) + 2(-0.5) + 0(-1)}{10} = 0.6$$

El valor obtenido final de 0.6 representa “satisfacción” en la escala propuesta por la técnica iadov tal y como se puede observar en el gráfico 19.

Gráfico 19. Escala de grado de satisfacción.



Fuente: elaboración propia.

El propósito de la información anteriormente expuesta era realizar un preámbulo demostrativo de cómo funciona la técnica iadov, con la cual después de su aplicación se obtuvieron los siguientes resultados:

Tabla 27. Resultados obtenidos después de realizar iadov.

Personas	Variable	Valor del cuadro lógico iadov	valor de satisfacción
8	A	1	1
1	B	2	0.5
Total personas N: 9			

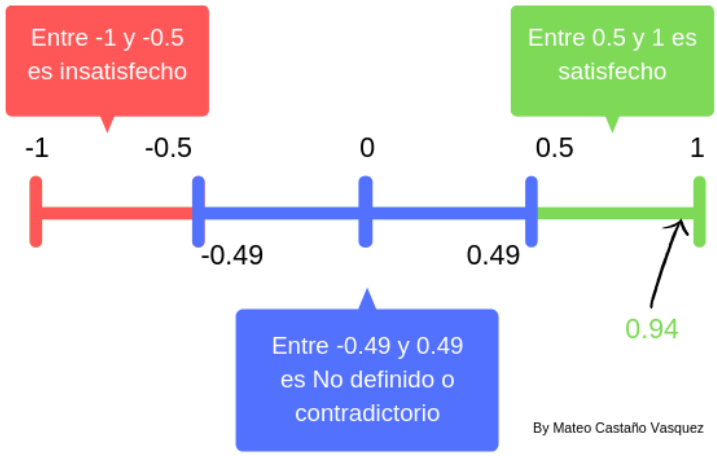
Fuente: elaboración propia.

Al remplazar las variables en la fórmula de ISG con los valores de la tabla 27 la cual es un resumen de los datos extraídos de la aplicación de la técnica a 9 personas, se

obtuvo un total de 0.94 que indica un nivel de “satisfacción” para los lectores del instructivo según la escala de valores propuestos por la técnica iadov.

$$ISG = \frac{8(1) + 1(0.5) + 0(0) + 0(-0.5) + 0(-1)}{9} = 0.94$$

Gráfico 20. Grado de satisfacción obtenido.



Fuente: elaboración propia.

Como resultado general a la validación por medio de la técnica iadov que se aplicó a los 9 participantes seleccionados, con la condición de que el instructivo generado por este proyecto puede ser usado en su vida profesional y personal, son desarrolladores de Java que desean automatizar la ejecución de pruebas unitarias, se encontró un valor ISG de 0.94 que se ubica en “Satisfecho” según el gráfico 20, demuestra que aún existen oportunidades de mejora, pero que el instructivo satisface la mayoría de necesidades definidas en este proyecto para el posible lector.

CONCLUSIONES

- La investigación para la fundamentación teórica y metodológica fue una guía para la selección y uso de las herramientas necesarias para completar el proyecto.
- El uso de la metodología Scrum, permitió la división y orden del proyecto en entregables, mejor definidos para ser desarrollados por *Sprints* o iteraciones en un tiempo limitado.
- La validación de resultados permitió conocer el grado de satisfacción que proporcionan las funcionalidades del proyecto.

RECOMENDACIONES

- Se recomienda conectar más herramientas y detallar la forma de realizarlo, proporcionará más funcionalidades que son de utilidad en proyectos donde la automatización es importante.
- Sería aconsejable implementar la ejecución automática de pruebas de *front-end* con herramientas como *Appium* o *Selenium* proveerían un mejor aseguramiento de calidad.
- Podría ser aconsejable utilizar herramientas pagas para algunas tareas, podrían agregar funcionalidades más potentes al proyecto.
- En lo posible investigar la información proporcionada por empresas o desarrollador de una herramienta siempre brinda detalles útiles de su uso.

BIBLIOGRAFÍA

#1 Marketplace for DevOps apps. (s. f.). Recuperado 11 de junio de 2019, de Atlassian Marketplace website: https://marketplace.atlassian.com/categories/devops

About GitLab. (s. f.). Recuperado 9 de junio de 2019, de GitLab website: https://about.gitlab.com/company/

Acerca del Product Owner. (s. f.). Recuperado 5 de mayo de 2019, de Scrum.org website: https://www.scrum.org/resources/blog/acerca-del-product-owner

Ahad, A., Ullah, Z., Tariq, L., & Niaz, S. (2017). Software Inspections and Their Role in Software Quality Assurance. American Journal of Software Engineering and Applications, 6(4), 105-110.

Andrews, E. (s. f.). 5 Cold War Close Calls. Recuperado 5 de mayo de 2019, de HISTORY website: https://www.history.com/news/5-cold-war-close-calls

Archiveddocs. (s. f.). Chapter 1 - Introduction to the Microsoft Solutions Framework. Recuperado 11 de junio de 2019, de https://docs.microsoft.com/en-us/previous-versions/tn-archive/bb497060(v%3dtechnet.10)

Arias, F. G. (2012). El proyecto de investigación. Introducción a la metodología científica. 6ta. Fidias G. Arias Odón.

ASALE, R.-. (s. f.). «Diccionario de la lengua española» - Edición del Tricentenario. Recuperado 5 de mayo de 2019, de «Diccionario de la lengua española» - Edición del Tricentenario website: http://dle.rae.es/

Atlassian. (2019). ¿Qué es DevOps? Recuperado 9 de junio de 2019, de Atlassian website: https://es.atlassian.com/devops

Bass, L., Weber, I., & Zhu, L. (2015). DevOps: A software architect's perspective. Addison-Wesley Professional.

Beck, K., & Gamma, E. (2000). Extreme programming explained: embrace change. addison-wesley professional.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... & Kern, J. (2001). Manifesto for agile software development.

Brunnert, A., van Hoorn, A., Willnecker, F., Danciu, A., Hasselbring, W., Heger, C., ... & Koziolek, A. (2015). Performance-oriented DevOps: A research agenda. arXiv preprint arXiv:1508.04752.

Build software better, together. (s. f.). Recuperado 9 de junio de 2019, de GitHub website: https://github.com

Burnstein, I. (2006). Practical software testing: a process-oriented approach. Springer Science & Business Media.

Chua, B. B., & Dyson, L. E. (2004, December). Applying the ISO 9126 model to the evaluation of an e-learning system. In Proc. of ASCILITE (pp. 5-8).

Cohn, M. (2015). Product Backlog Refinement (Grooming).

Continuous Delivery with Jenkins -- Part 2. (2016, abril 27). Recuperado 11 de junio de 2019, de CloudBees website: https://www.cloudbees.com/blog/continuous-delivery-jenkins-part-2

Crosby, P. B. (1979) Quality is Free, McGraw-Hill, New York.

Dávila Newman, G. (2006). El razonamiento inductivo y deductivo dentro del proceso investigativo en ciencias experimentales y sociales. Laurus, 12, 180-20

Díaz Sanjuán, L. (2011). La observación.

Enhance Your Workflow with Continuous Code Quality | SonarQube. (2019). Recuperado 9 de junio de 2019, de https://www.sonarqube.org/features/enhance-your-workflow/

eRA: RUP Fundamentals Presentation. (s. f.). Recuperado 11 de junio de 2019, de https://era.nih.gov/docs/rup_fundamentals.htm

Figueroa, R. G., Solís, C. J., & Cabrera, A. A. (2008). Metodologías tradicionales vs. metodologías ágiles. Universidad Técnica Particular de Loja, Escuela de Ciencias de la Computación.

Fuentes, J. R. L. (2015). Desarrollo de Software ÁGIL: Extreme Programming y Scrum. IT Campus Academy.

Galin, D. (2004). Software quality assurance: from theory to implementation. Pearson Education India.

GitHub Glossary - GitHub Help. (2019). Recuperado 9 de junio de 2019, de https://help.github.com/en/articles/github-glossary

Graham, D., Van Veenendaal, E., & Evans, I. (2008). Foundations of software testing: ISTQB certification. Cengage Learning EMEA.

Hooda, I., & Chhillar, R. S. (2015). Software test process, testing types and techniques. International Journal of Computer Applications, 111(13).

Jenkins User Documentation. (s. f.). Recuperado 9 de junio de 2019, de Jenkins User Documentation website: https://jenkins.io/doc/index.html

JUnit 5. (2019). Recuperado 9 de junio de 2019, de https://junit.org/junit5/

Juran, J. M. (1988) Juran's Quality Control Handbook, 4th edn, J. M. Juran, Editor in Chief; I. M. Gryne, Associate Editor. McGraw-Hill, New York.

Kasurinen, J., Taipale, O., & Smolander, K. (2010). Software test automation in practice: empirical observations. Advances in Software Engineering, 2010.

Ken Schwaber y Jeff Sutherland (2016). La Guía de Scrum TM

Kuzmina, N. V. (1970). Metódicas investigativas de la actividad pedagógica. Editorial Leningrado.

Leicher, A. (2018, julio 20). Delivery Pipelines as enabler for a DevOps culture. Recuperado 11 de junio de 2019, de Hacker Noon website: https://hackernoon.com/delivery-pipelines-as-enabler-for-a-devops-culture-ebc45963f703

Letelier, P. (2006). Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP).

Lin, L., He, J., Zhang, Y., & Song, F. (2015). Quality Assurance through Rigorous Software Specification and Testing: A Case Study. Procedia Computer Science, 62, 257-265.

Martínez, A., & Martínez, R. (2014). Guía a rational unified process. Albacete, España.

Mendoza, L. E., Grimán, A., & Pérez, M. (s. f.). Especialización de MSF para el desarrollo basado en componentes de sistemas colaborativos.

Orozco, G., & Righi, L. G. (2018). Integración continua: solución a los problemas de productividad y calidad del código en un entorno ágil testigo.

Pressman, R. S. (2000) Software Engineering – A Practitioner's Approach, European adaptation by D. Ince, 5th edn, McGraw-Hill International, London.

Pressman, R. S., & Troya, J. M. (1988). Ingeniería del software (No. 001.64 P74s.). McGraw Hill.

Pérez A. O. (2011). Cuatro enfoques metodológicos para el desarrollo de Software RUP – MSF – XP - SCRUM. INVENTUM, 6(10), 64-78. https://doi.org/10.26620/uniminuto.inventum.6.10.2011.64-78

Restrepo Herrón, A. A., & Gil, J. G. (2018). Marco para implementar el sistema de gestión de seguridad de la información con marco de trabajo ágil Scrum alcaldía de Calarcá.

Rodríguez Jiménez, A., Jacinto, P., & Omar, A. (2017). Métodos científicos de indagación y de construcción del conocimiento. Revista EAN, (82), 179-200.

Schwaber, K., & Sutherland, J. (2017). The Scrum GuideTM. The definitive guide to scrum: The rules of the game. November 2017.

Sliger, M. (2011). Agile project management with Scrum. Paper presented at PMI® Global Congress 2011—North America, Dallas, TX. Newtown Square, PA: Project Management Institute.

TestNG - Welcome. (2019). Recuperado 9 de junio de 2019, de https://testng.org/doc/

The DevOps Lifecycle with GitLab. (s. f.). Recuperado 9 de junio de 2019, de GitLab website: https://about.gitlab.com/stages-devops-lifecycle/

Verona, J. (2016). Practical DevOps. Packt Publishing Ltd.

Walls, M. (2013). Building a DevOps culture. " O'Reilly Media, Inc.".

What is Scrum? (2019). Recuperado 9 de junio de 2019, de Scrum.org website: https://www.scrum.org/resources/what-is-scrum

XP flow Chart. (s. f.). Recuperado 11 de junio de 2019, de http://www.extremeprogramming.org/map/project.html

ANEXOS

ANEXO 1: INSTRUCTIVO

INSTRUCTIVO

INSTRUCTIVO PARA LA CREACIÓN DE UN VALIDADOR DE REQUERIMIENTOS PARA LA EJECUCIÓN AUTOMÁTICA DE PRUEBAS UNITARIAS MEDIANTE UN PIPELINE

PUCESA

JUNIO 2019

ÍNDICE

1. AMBIENTE DE DESARROLLO

Requisitos Míninos:

- Sistema operativo: Windows 10 Pro 64 bits (10.0, compilación 17134)
- Idioma del sistema operativo: Español (Configuración regional: español)
- Procesador: Intel® Core™ i5-4200M CPU @ 2.50GHz (4 CPUs), ~2.5GHz
- Memoria RAM: 8192MB
- Disco duro: 500gb
- Modelo dispositivo: Hewlett Packard HP ProBook 440 G1

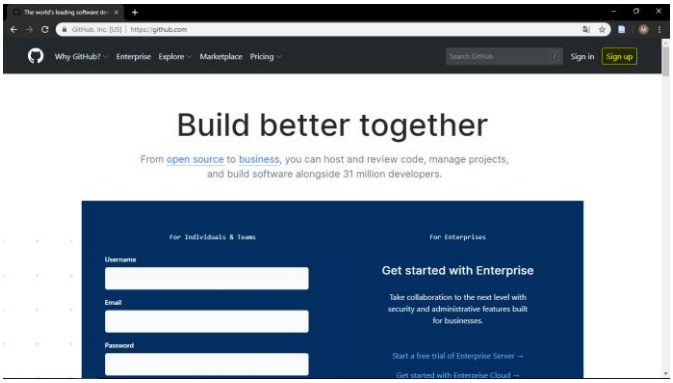
1.1. **Precondiciones**:

- Ambiente de desarrollo instalado para aplicaciones JAVA: Eclipse Neon versión 2018-09 (4.9.0)
- Apache Maven versión 3.6.0
- Java versión 1.8.0_211

2. CREACIÓN DE CUENTA EN GITHUB

Para la creación de la cuenta GitHub se utilizará la versión expuesta en la página oficial de https://github.com/ con fecha al 4 de mayo del 2019.

2.1. Ingrese a https://github.com/ y seleccione la opción Sign Up.

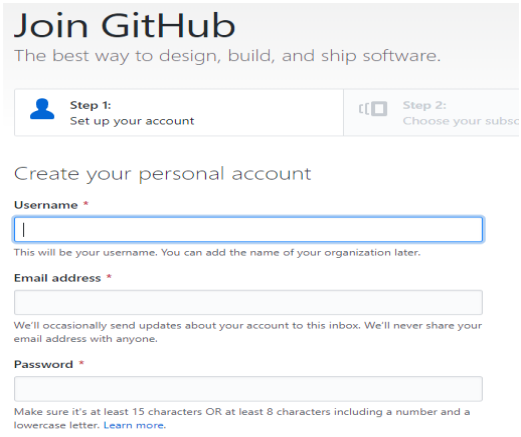


2.2. Complete los datos solicitados

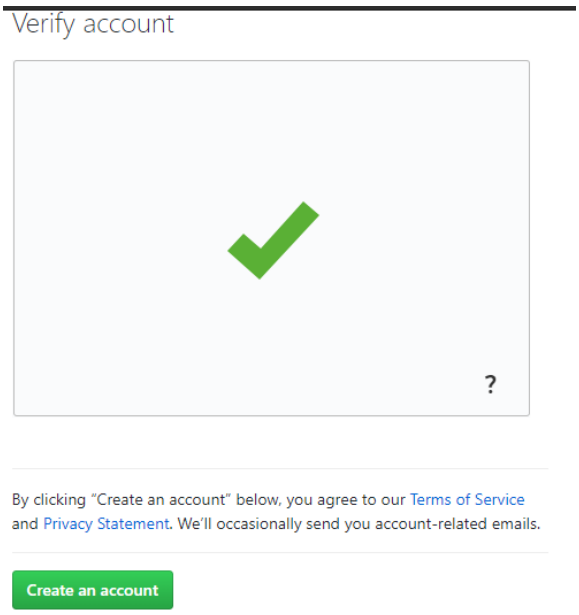
- **Username**: Usuario válido con el que se identificará al entrar en github
- **Email address**: Ingrese un e-mail valido
- **Password**: Agregue la contraseña con la cual ingresará a su cuenta.

En este ejemplo se utilizaron los siguientes datos

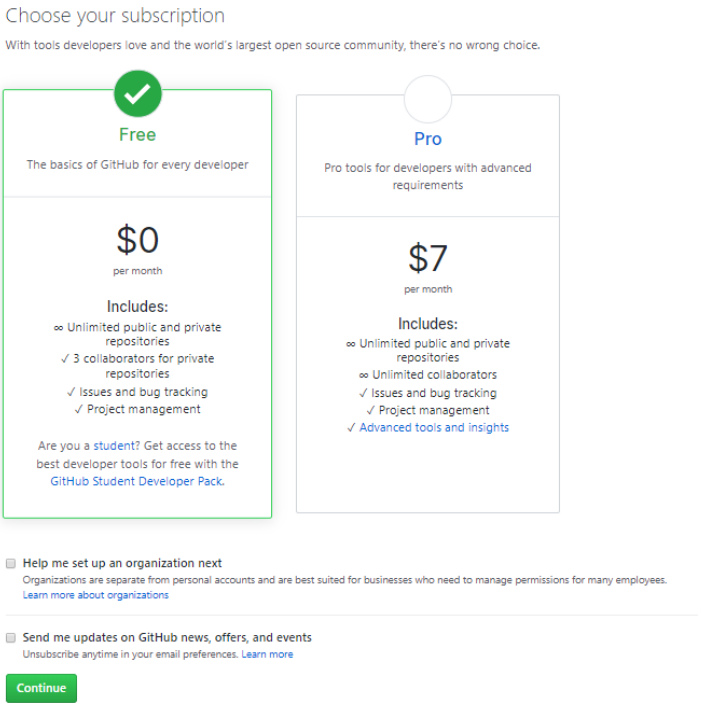
- **Usuario**: Demostracion1
- **Contraseña**: Demostracion1
- **Correo**: ddemostracion6@gmail.com



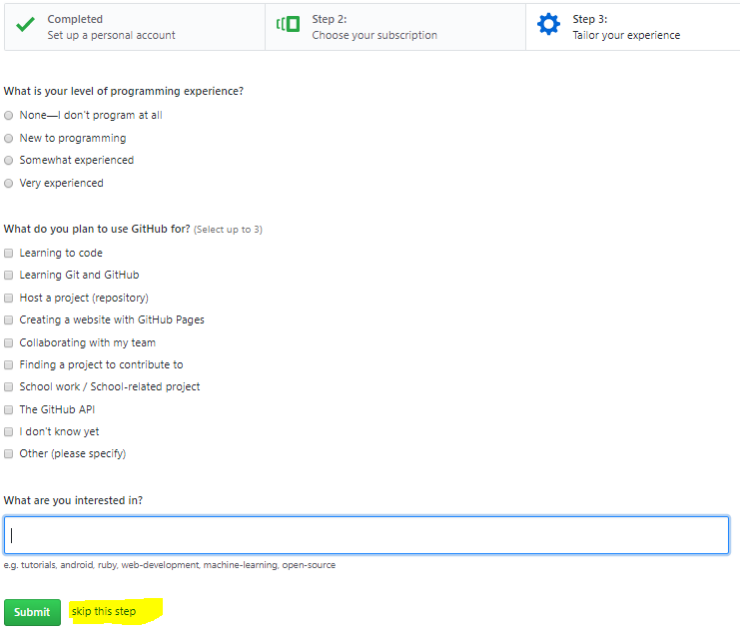
2.3. Verifique la cuenta como es solicitado y realice click sobre el botón “Create an account”.



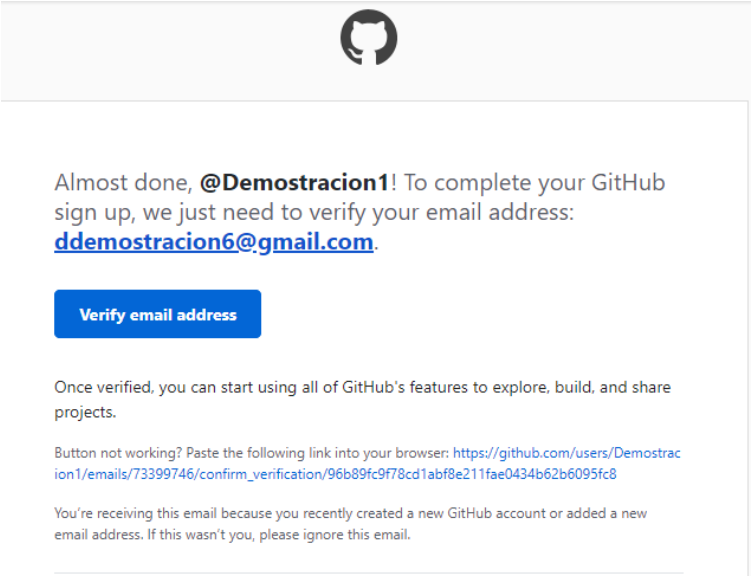
2.4. Seleccione el plan que mejor se acople a sus necesidades, en este caso se seleccionará FREE → click en “Continue”



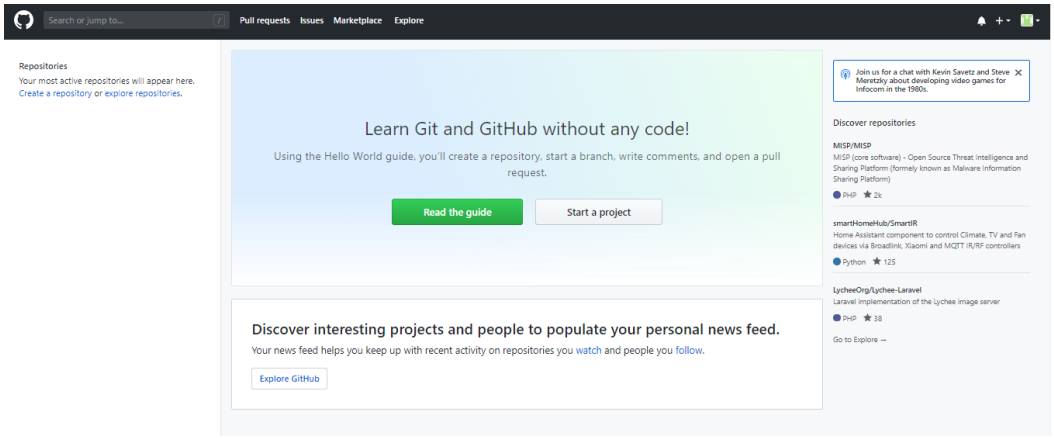
2.5. Complete la encuesta o si desea puede saltar este paso, en este caso práctico se seleccionará “skip this step”.



2.6. Ingrese al correo electrónico inscrito anteriormente y verifique su cuenta.



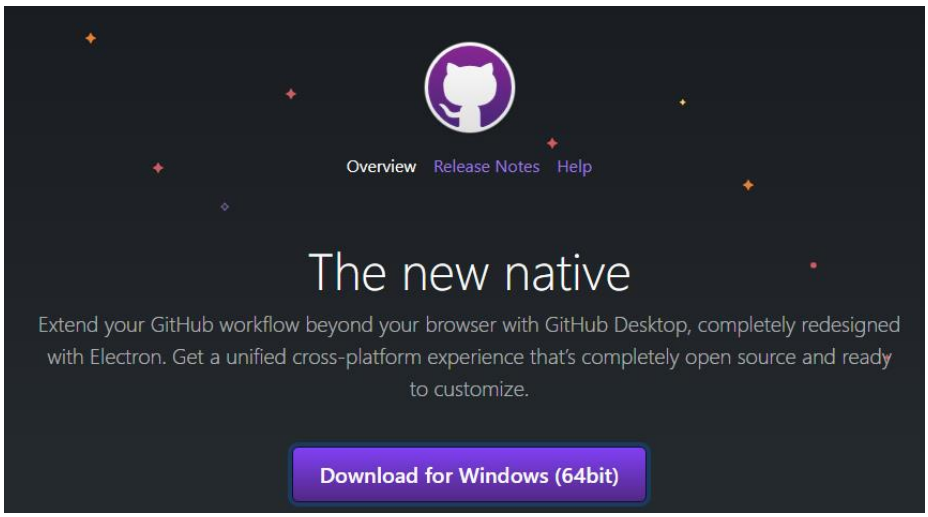
2.7. Finalmente se encuentra en el panel principal de GITHUB



3. INSTALACIÓN DE GITHUB DESKTOP

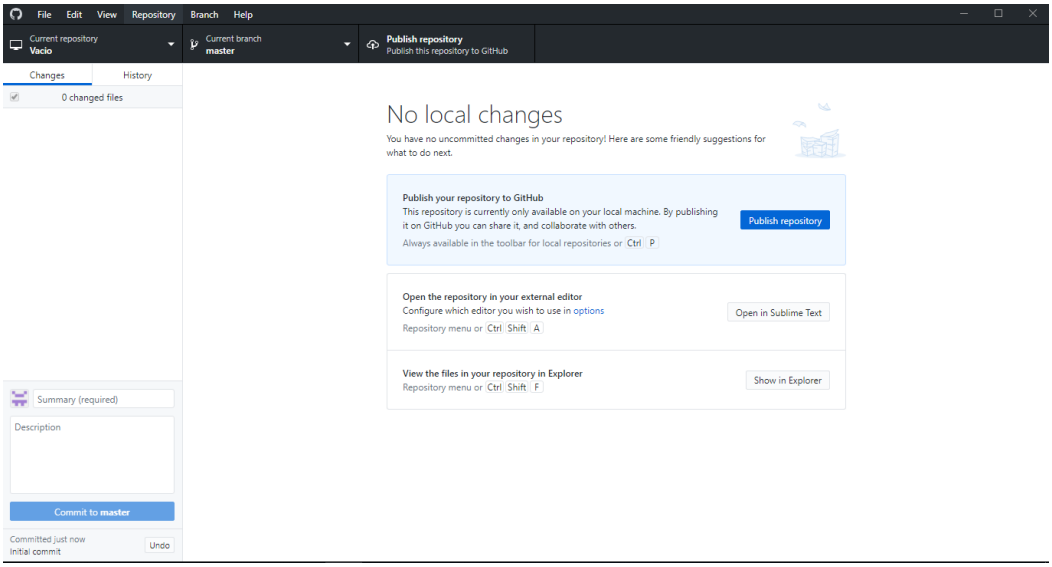
La aplicación GitHub Desktop versión 1.6.5 será utilizada para realizar el versionamiento de código del proyecto desde una interfaz gráfica.

3.1. Ingrese al link https://desktop.github.com/ desde el cual puede descargar GitHub Desktop y presione el botón “Download for Windows (64 bit)”



3.2. Después de finalizar la descarga del archivo, realice click sobre el instalador y espere hasta la finalización de la instalación

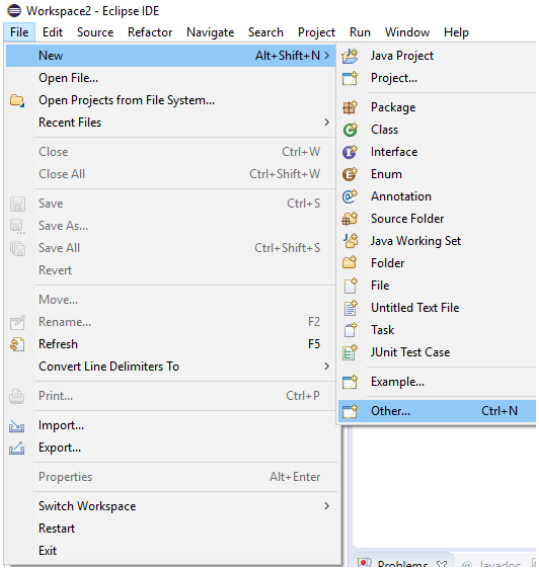
3.3. Finalmente, GitHub Desktop se iniciará automáticamente y mostrará una ventana similar a la siguiente.



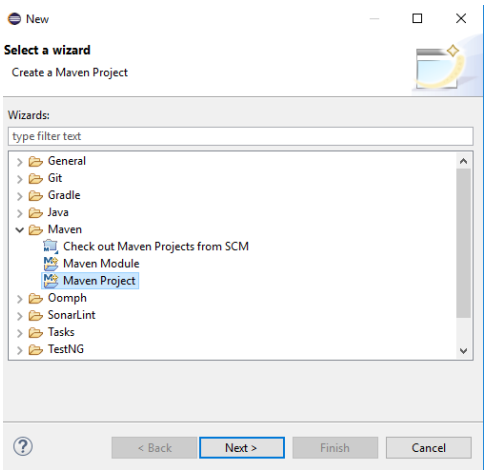
4. CREACIÓN DE PROYECTO JAVA

Para la creación del proyecto Java se utilizará la versión del entorno de desarrollo Eclipse Eclipse Neon Versión: 2018-09 (4.9.0) utilizando MAVEN para el manejo de dependencias y el Framework TestNg para la creación de pruebas unitarias.

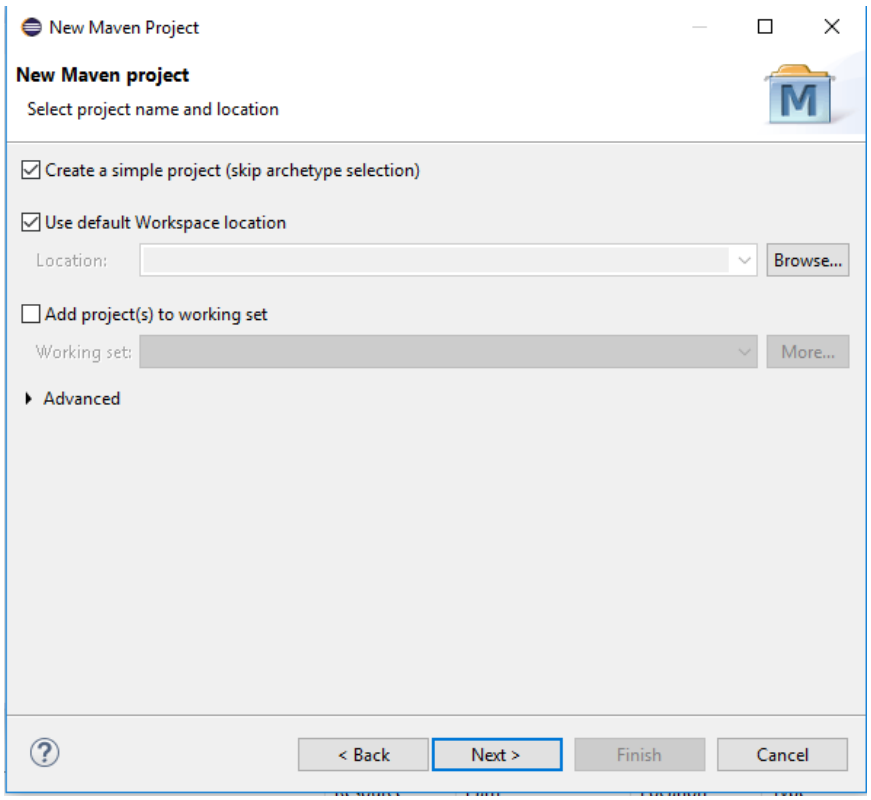
4.1. Ejecute el programa Eclipse y seleccione la opción **File** en la esquina superior izquierda → **New** → **Other**



4.2. En la ventana emergente selecciones la carpeta con nombre **Maven** → **Maven Project** → **Next**



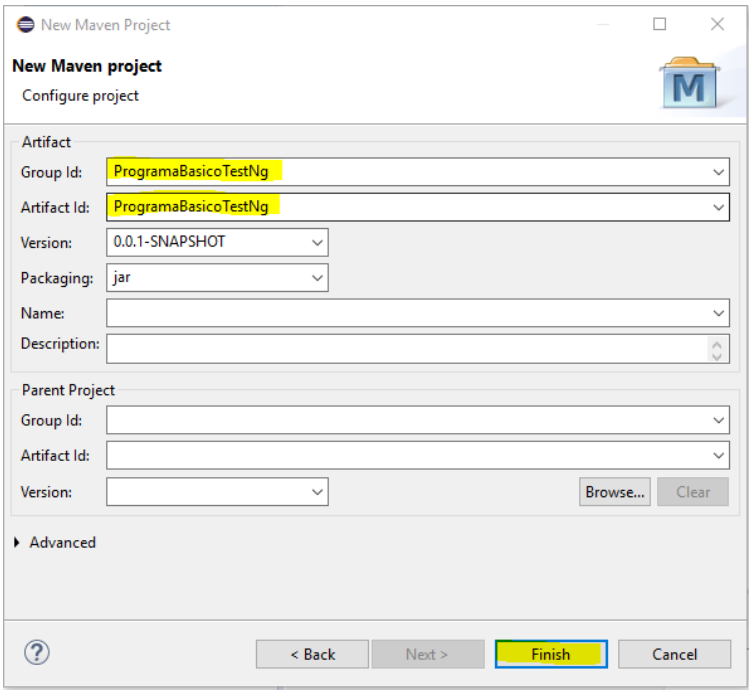
4.3. Seleccione la casilla **create a simple Project (skip archetype selection)** → realice click en el botón **Next**



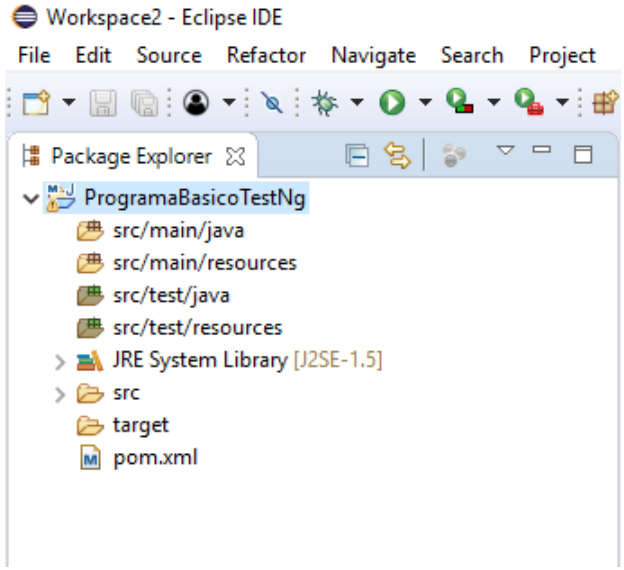
4.4. Ingrese los siguientes datos en la sección **Artifact:**

- Group Id: ProgramaBasicoTestNg
- Artifact Id: ProgramaBasicoTestNg

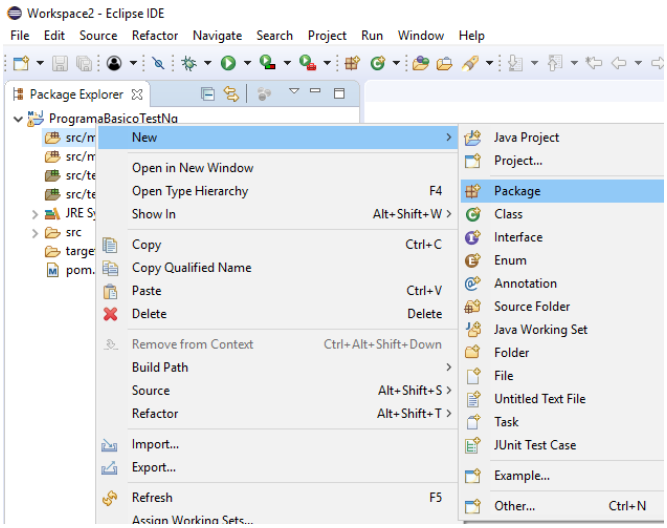
Donde **Group Id** es el nombre del grupo del proyecto y **Artifact Id** el nombre del proyecto, luego realice click en **Finish.**



4.5. Finalmente obtendrá una estructura similar a la siguiente.

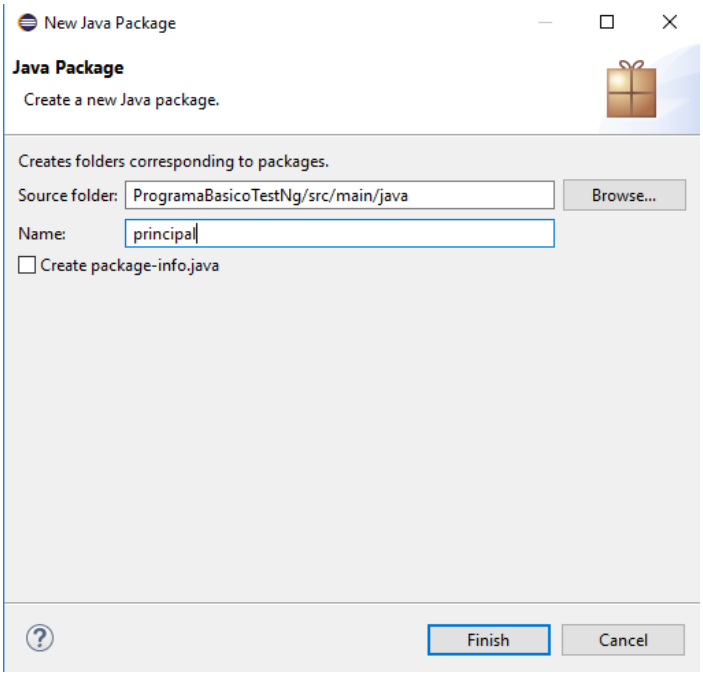


4.6. Despliegue la estructura anteriormente creada y en la carpeta src/main/java realice click derecho → New→ Package

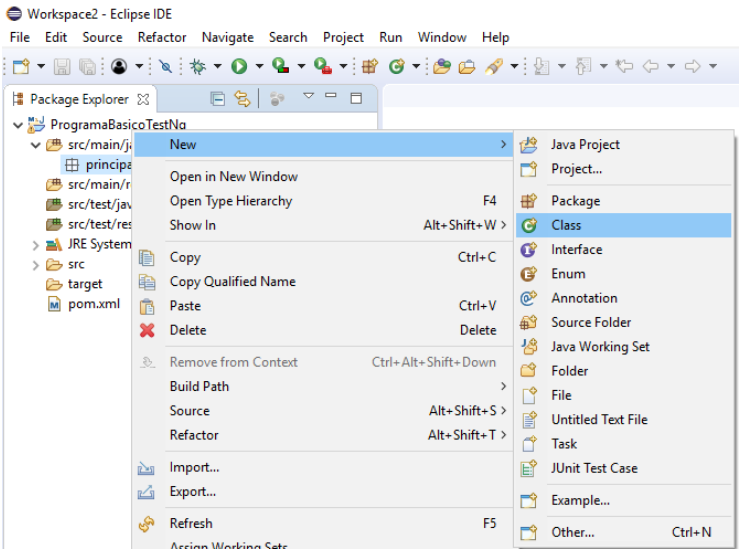


4.7. En la siguiente ventana agregue el nombre del paquete en el campo Name, en este caso el paquete se nombrará “principal”, después de agregar el nombre realice click en Finish.

Name: principal.



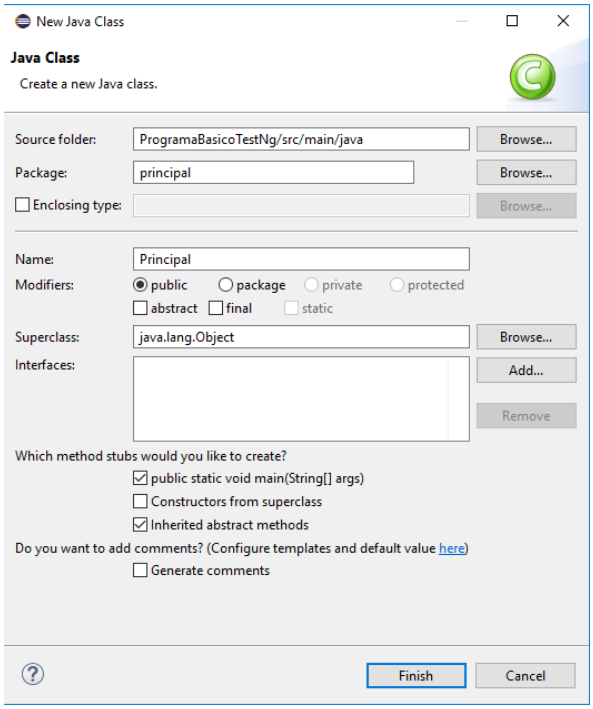
4.8. En el paquete creado, en este caso llamado “principal”, realice click derecho -> New -> Class



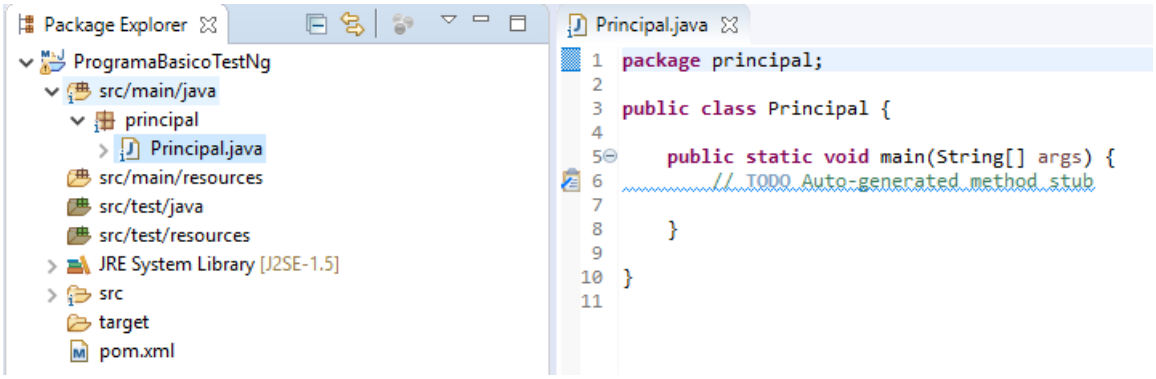
4.9. En la ventana emergente, agregue el nombre de la clase en el campo Name, en este caso se llamará Principal, luego seleccione la casilla “public static void main(String[] args)”, de este modo se agregarán las sentencias necesarias para indicar que este es el método main del programa Java, finalmente realice click en Finish

Name: Nombre de la clase.

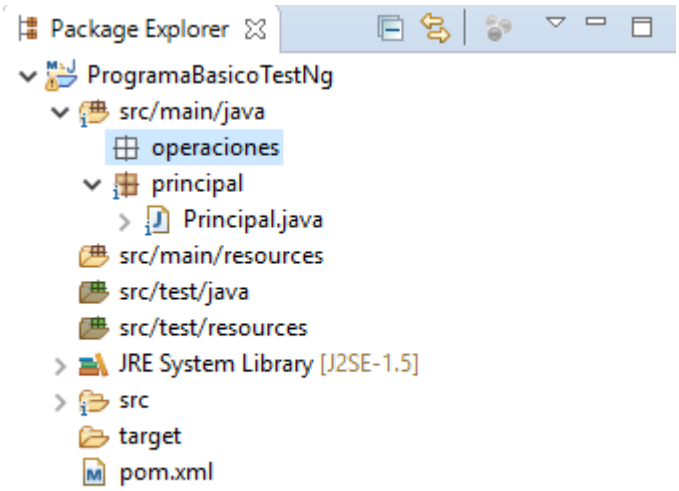
Casilla “public static void main(String[] args)”: Si se activa la casilla significa que se agregará el código necesario para indicar que esta es la clase main del programa Java.



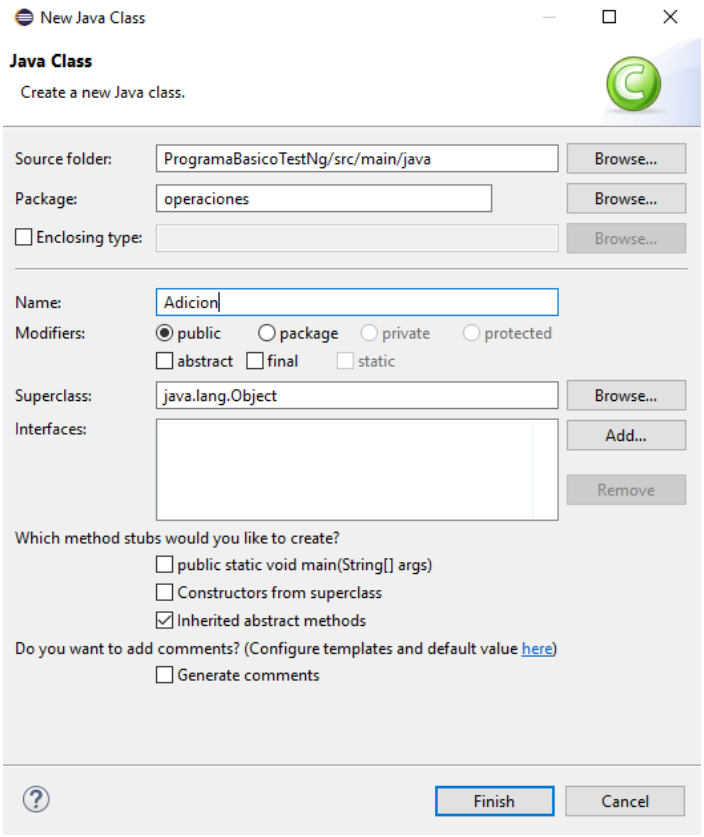
4.10. Después de creada la clase Principal.java, contará con la siguiente estructura y clases.



4.11. Cree un nuevo paquete en la carpeta src/main/java siguiendo los pasos 4.6 y 4.7, pero ahora el paquete se llamará “operaciones”.



4.12. Dentro del paquete “operaciones” anteriormente creado, cree una clase llamada Adicion.java, en este caso NO seleccione la casilla “public static void main(String[] Args)” y realice click sobre Finish.

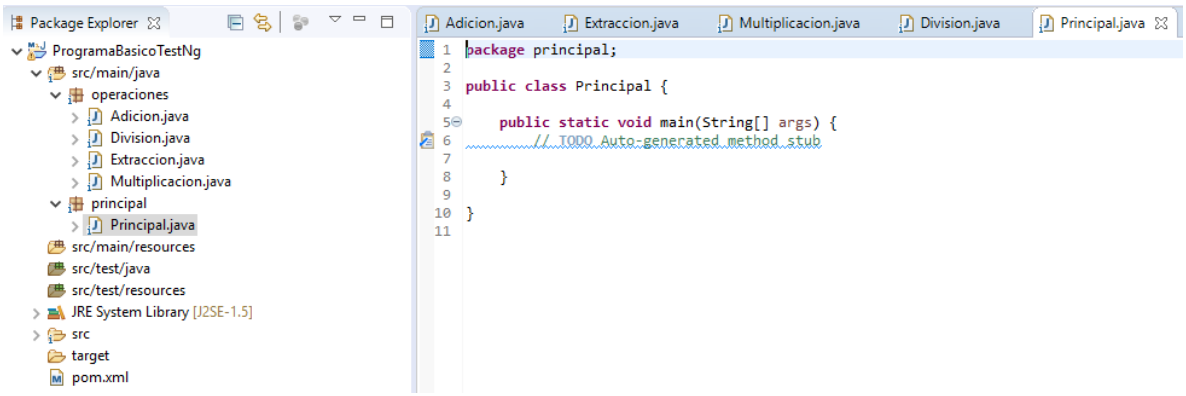


4.13. Dentro del paquete “operaciones”, cree una clase llamada Extraccion.java, en este caso NO seleccione la casilla “public static void main(String[] Args)” y realice click sobre Finish.

4.14. Dentro del paquete operaciones, cree una clase llamada Multiplicacion.java, en este caso NO seleccione la casilla “public static void main(String[] Args)” y realice click sobre Finish.

4.15. Dentro del paquete operaciones, cree una clase llamada Division.java, en este caso NO seleccione la casilla “public static void main(String[] Args)” y realice click sobre Finish.

4.16. Después de la creación de clases, la estructura será similar a la siguiente

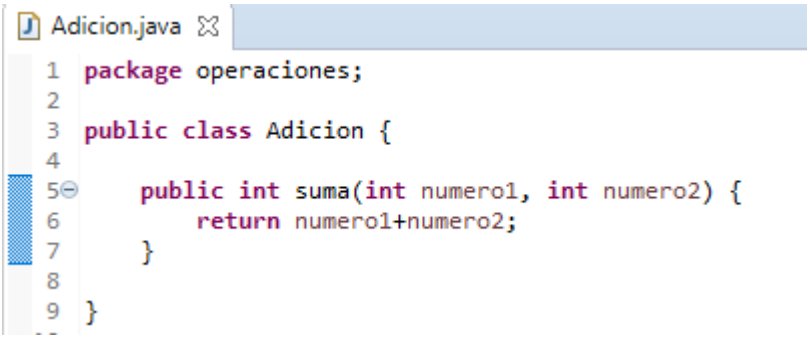


4.17. Realice doble click sobre la clase Adicion.java y agregue el siguiente código:

```
public int suma(int numero1, int numero2)
{
return numero1+numero2;
}
```

Después de agregar el código guarde los cambios con la opción guardar en la barra de herramientas superior u oprimiendo Ctrl + s.

La clase lucirá del siguiente modo:

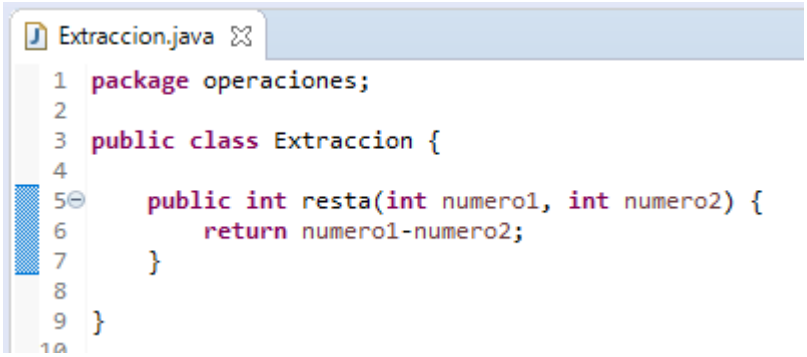


4.18. Realice doble click sobre la clase Extraccion.java y agregue el siguiente código:

```
public int resta(int numero1, int numero2)
{
return numero1-numero2;
}
```

Después de agregar el código guarde los cambios con la opción guardar en la barra de herramientas superior u oprimiendo Ctrl + s.

La clase lucirá del siguiente modo:



4.19. Realice doble click sobre la clase Division.java y agregue el siguiente código:

```
public int division(int numero1, int numero2) {
        if(numero2==0)
        {
                throw new IllegalArgumentException("Argumento 'divisor' es 0");
        }
        else
        {
                return numero1/numero2;
        }

}
```

Después de agregar el código guarde los cambios con la opción guardar en la barra de herramientas superior u oprimiendo Ctrl + s.

La clase lucirá del siguiente modo:

Division.java

```
1  package operaciones;
2
3  public class Division {
4
5      public int division(int numero1, int numero2) {
6          if(numero2==0) {
7              throw new IllegalArgumentException("Argumento 'divisor' es 0");
8          }else {
9              return numero1/numero2;
10         }
11
12     }
13
14 }
15
```

4.20. Realice doble click sobre la clase Multiplicacion.java y agregue el siguiente código

```
public int multiplicacion(int numero1, int numero2) {
        return numero1*numero2;
}
```

Después de agregar el código guarde los cambios con la opción guardar en la barra de herramientas superior u oprimiendo Ctrl + s.

La clase lucirá del siguiente modo:

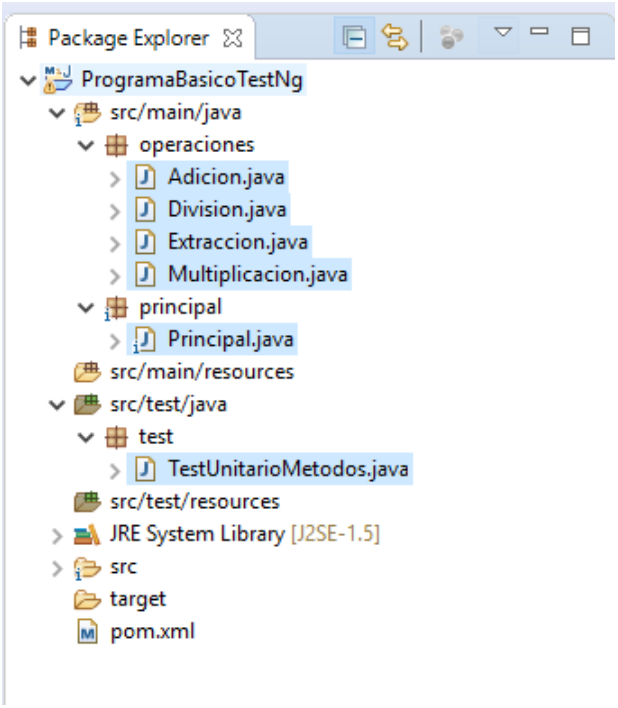
*Multiplicacion.java

```
1  package operaciones;
2
3  public class Multiplicacion {
4
5      public int multiplicacion(int numero1, int numero2)
6      {
7          return numero1*numero2;
8      }
9
10 }
```

4.21. En la carpeta src/test/java cree un paquete llamado test, puede seguir los pasos 4.6 y 4.7.

4.22. Dentro del paquete test, cree una clase llamada TestUnitarioMetodos.java, esta clase será la que contiene las pruebas sobre los métodos y como buena práctica contendrá en su nombre la palabra Test, después de nombrarla, NO seleccione la casilla “public static void main(String[] Args) y realice click sobre Finish, puede apoyarse del paso 4.12.

4.23. Después de crear la clase, tendrá una estructura similar a la siguiente.



4.24. Agregue el siguiente código dentro de la clase TestUnitarioMetodos.java

```
package test;

import java.util.logging.Level;

import operaciones.Adicion;
import operaciones.Division;
import operaciones.Extraccion;
import operaciones.Multiplicacion;
import org.testng.Assert;
import org.testng.annotations.AfterClass;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;
import org.testng.log4testng.Logger;

public class TestUnitarioMetodos {

	// Objeto de tipo Logger con el cual se escribira en la consola
	private final static Logger =
Logger.getLogger(Thread.currentThread().getClass());

	/**
	 * La anotacion @BeforeClass indica que este metodo se ejecutara una
sola vez
	 * antes que todas las pruebas @Test
	 */
	@BeforeClass
	public void beforeClass() {
		LOGGER.info("El beforeClass se ejecuta una vez antes de todas
las pruebas");
	}



	/**
	 * La anotacion @BeforeTest indica que este metodo se ejecutara antes
de que un
	 * metodo @Test se ejecute
	 */
	@BeforeTest
	public void beforeTest() {
		LOGGER.info("El beforeTest se ejecuta antes de cada prueba");
	}
```

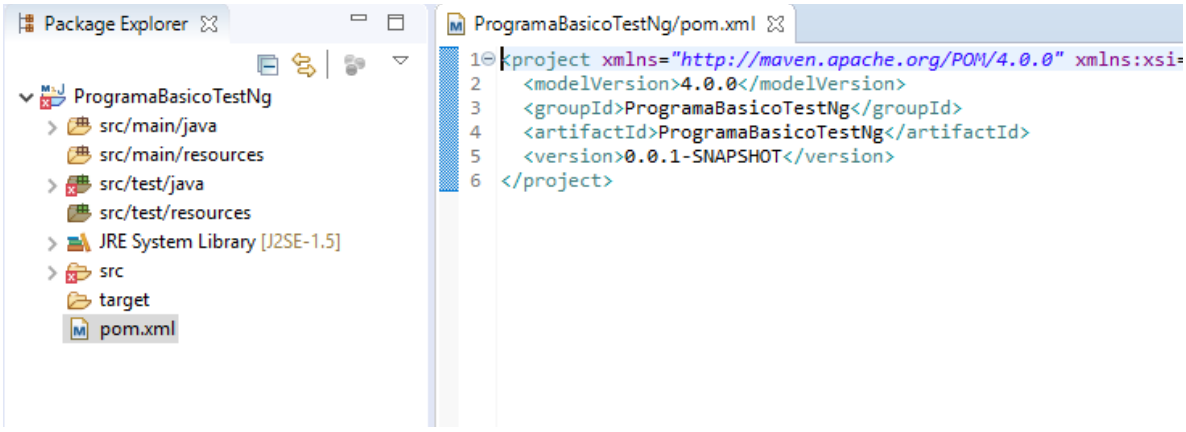
```
	/**
	 * Este metodo realiza la prueba sobre el metodo suma, que se encuentra en la clase Adicion.java
	 * Primero instancia un objeto de tipo Adicion y usa este objeto para hacer uso del metodo suma
	 * La anotacion @Test indica que este metodo es un metodo para pruebas
	 */
	@Test
	public void adicionTest() {
		LOGGER.info("prueba adicion");
		Adicion = new Adicion();
		Assert.assertEquals(3, adicion.suma(1, 2));
	}


	/**
	 * Este metodo realiza la prueba sobre el metodo resta, que se encuentra en la clase Extraccion.java
	 * Primero instancia un objeto de tipo Extraccion y usa este objeto para hacer uso del metodo resta
	 * La anotacion @Test indica que este metodo es un metodo para pruebas
	 */
	@Test
	public void extraccionTest() {
		LOGGER.info("prueba extraccion");
		Extraccion = new Extraccion();
		Assert.assertEquals(0, extraccion.resta(2, 2));
	}


	/**
	 * Este metodo realiza la prueba sobre el metodo multiplicacion, que se encuentra en la clase Multiplicacion.java
	 * Primero instancia un objeto de tipo Multiplicacion y usa este objeto para hacer uso del metodo multiplicacion
	 * La anotacion @Test indica que este metodo es un metodo para pruebas
	 */
	@Test
```

```
	public void multiplicacionTest() {
		LOGGER.info("prueba multiplicacion");
		Multiplicacion = new Multiplicacion();
		Assert.assertEquals(4, multiplicacion.multiplicacion(2, 2));
	}



	/**
	 * Este metodo realiza la prueba sobre el metodo division con valores válidos,
	 * este metodo se encuentra en la clase Division.java
	 * Primero instancia un objeto de tipo Division y usa este objeto para hacer uso del
	 * metodo division
	 * La anotacion @Test indica que este metodo es un metodo para pruebas
	 */
	@Test
	public void divisionValoresValidosTest() {
		LOGGER.info("prueba division valores validos");
		Division division = new Division();
		Assert.assertEquals(2, division.division(4, 2));

	}



	/**
	 * Este metodo realiza la prueba sobre el metodo division con un divisor 0,
	 * este metodo se encuentra en la clase Division.java
	 * Primero instancia un objeto de tipo Division y usa este objeto para hacer uso del
	 * metodo division
	 *
	 * La anotacion @Test(expectedExceptions = {
	 *                         IllegalArgumentException.class                 }, expectedExceptionsMessageRegExp = "Argumento 'divisor' es 0")
	 *
	 * Indica que este metodo es para pruebas y espera un error con mensaje
	 *
	 */
	@Test(expectedExceptions = {
			IllegalArgumentException.class                 }, expectedExceptionsMessageRegExp = "Argumento 'divisor' es 0")
```

```
	public void divisionParaZero() {
		LOGGER.info("prueba division valor divisor cero revisando excepcion");
		Division = new Division();
		division.division(1, 0);
	}


	/**
	 * La anotacion @AfterTest indica que este metodo se ejecutara después de que un
	 * metodo @Test se ejecute
	 */
	@AfterTest
	public void afterTest() {
		LOGGER.info("El afterTest se ejecuta despues de la ejecución de cada prueba");
	}



  /**
	* La anotacion @BeforeClass indica que este metodo se ejecutara una sola vez
	* despues que todas las pruebas @Test
	*/
	@AfterClass
	public void afterClass() {
		LOGGER.info("El afterClass se ejecuta una vez despues de todas las pruebas");

	}
}
```

Después de agregar el código guarde los cambios con la opción guardar en la barra de herramientas superior u oprimiendo Ctrl + s, las líneas de código que se muestran con error serán solucionadas en los siguientes pasos.

4.25. Realice doble click sobre el archivo llamado pom.xml

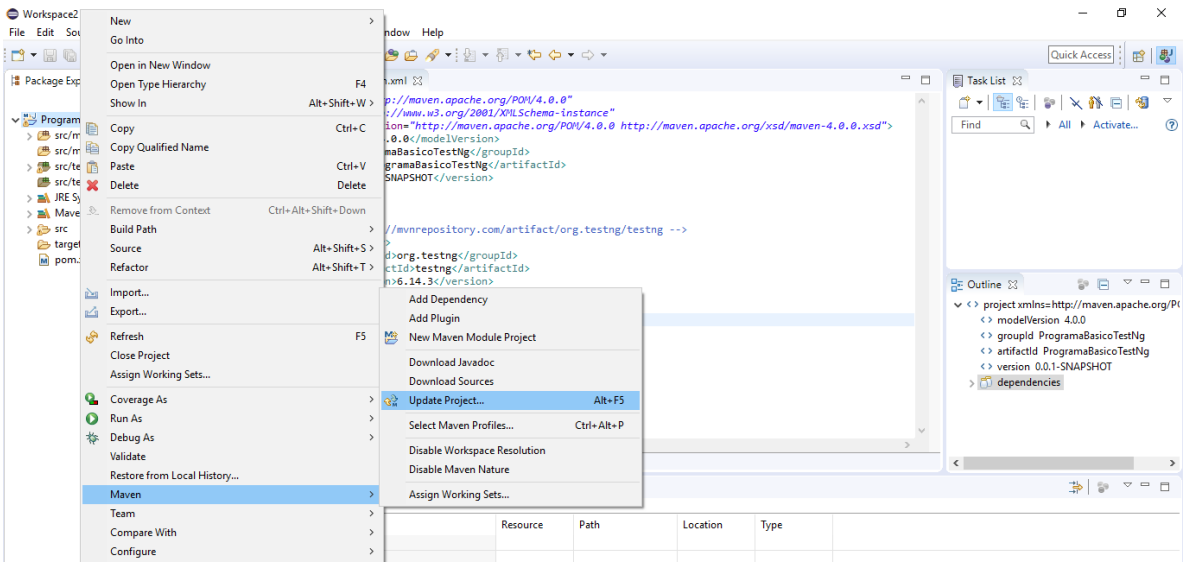


4.26. En el archivo pom.xml, agregue el siguiente código antes de la línea </Project>, así Maven identificará que el proyecto necesita hacer uso de las librerías de TestNg, por lo que estas se descargarán e incorporarán al proyecto

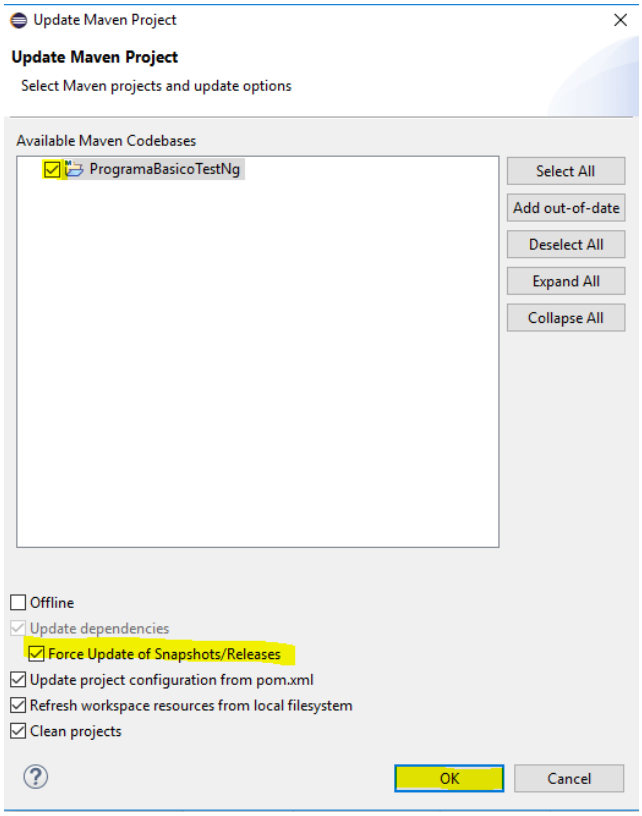
```
<dependencies>
		<!-- https://mvnrepository.com/artifact/org.testng/testng -->
		<dependency>
			<groupId>org.testng</groupId>
			<artifactId>testng</artifactId>
			<version>6.14.3</version>
			<scope>test</scope>
		</dependency>

</dependencies>
```

4.27. Después de agregar el código guarde los cambios con la opción guardar en la barra de herramientas superior u oprimiendo Ctrl + s, finalmente el archivo pom.xml lucirá del siguiente modo.

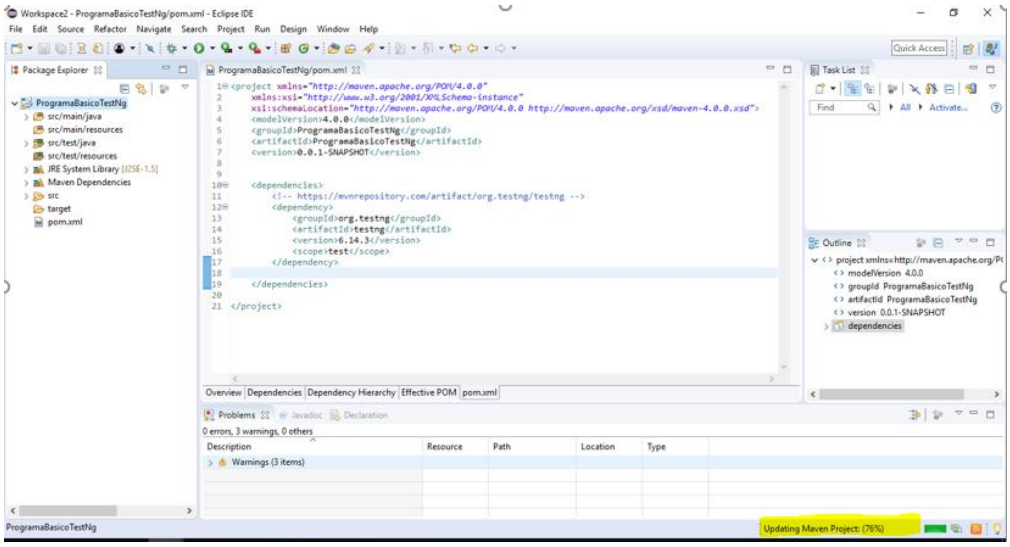
4.28. Realice click derecho sobre la carpeta principal del proyecto en este caso la carpeta principal se llama ProgramaBasicoTestNg, en el menú desplegado realice click sobre la opción Maven → en el siguiente menú desplegado seleccione Update Project , de este modo Maven iniciará el proceso de descarga de las librerías que fueron configuradas como dependencias en el archivo pom.xml.



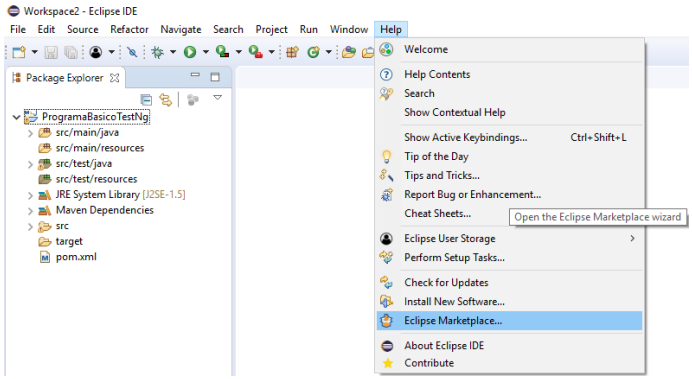
4.29. En la ventana emergente seleccione el check del proyecto, también seleccione el check de la opción “Force Update of Snapshots/Releases” y luego realice click sobre el botón OK.



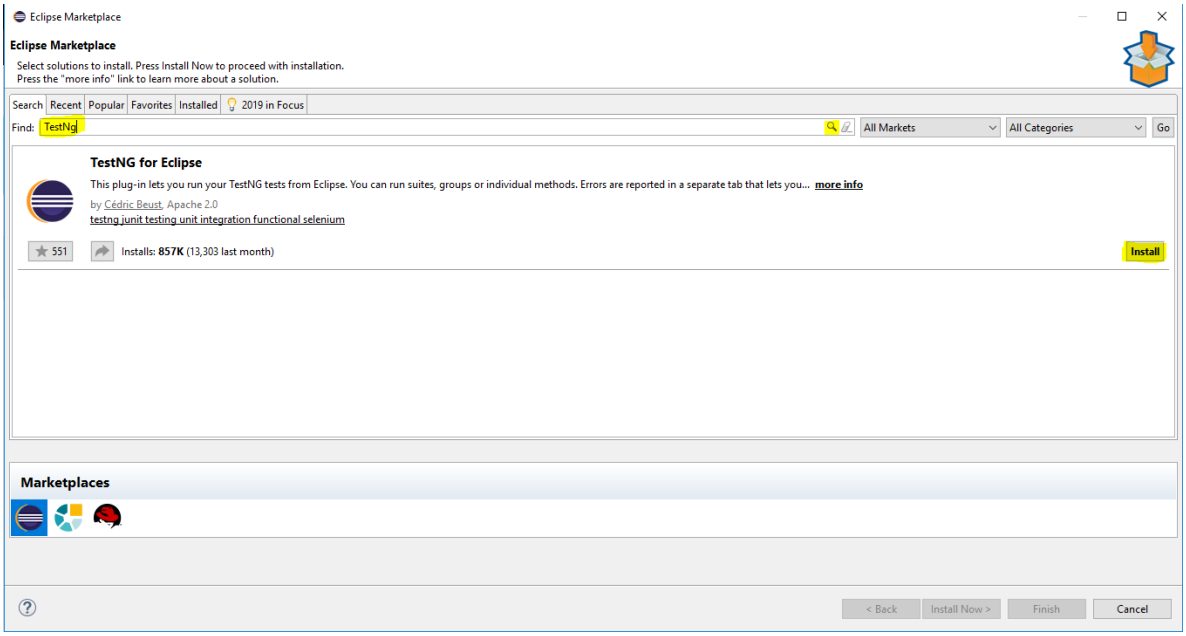
4.30. Espere a que Maven descargue las librerías necesarias para el programa.



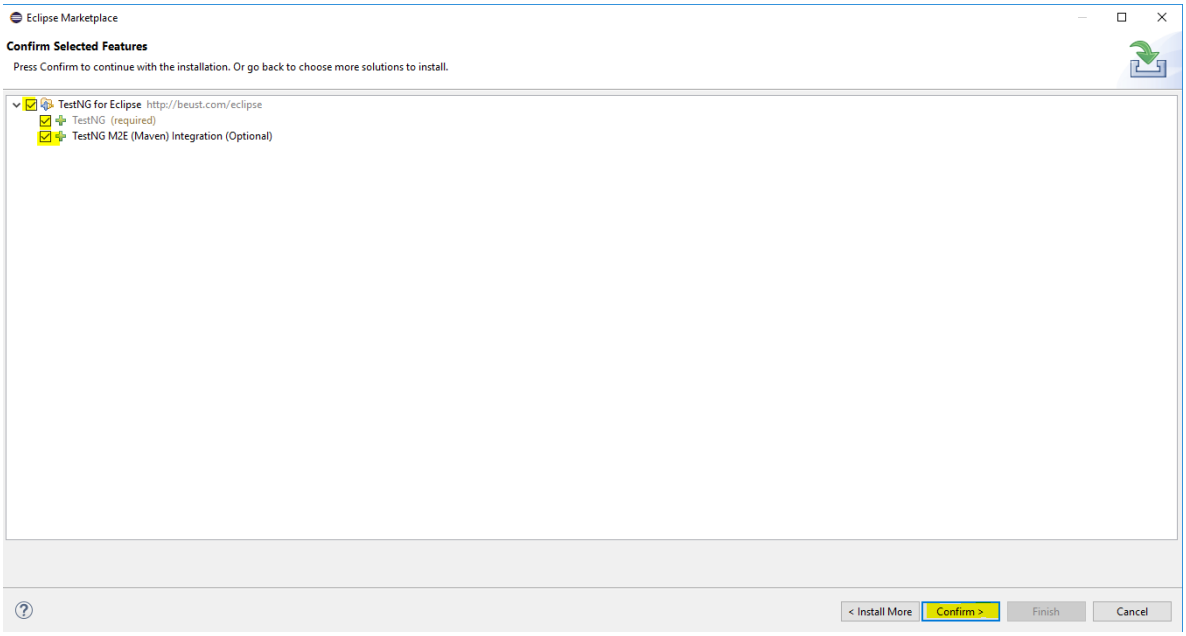
4.31. Realice click en el menú Help en la parte superior de la pantalla, luego seleccione la opción Eclipse Marketplace… para agregar el plugin de TestNg a Eclipse.



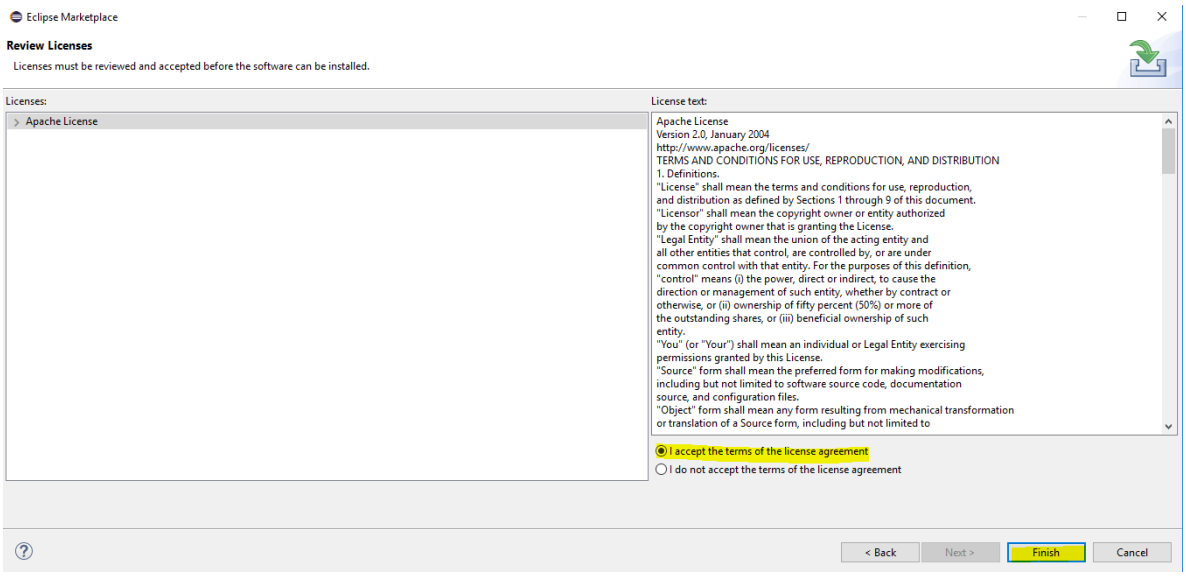
4.32. En la ventana emergente , escriba en el campo Find la palabra TestNg luego realice click en la lupa, cuando se encuentre el plugin TestNg for Eclipse, seleccione la opción Install.



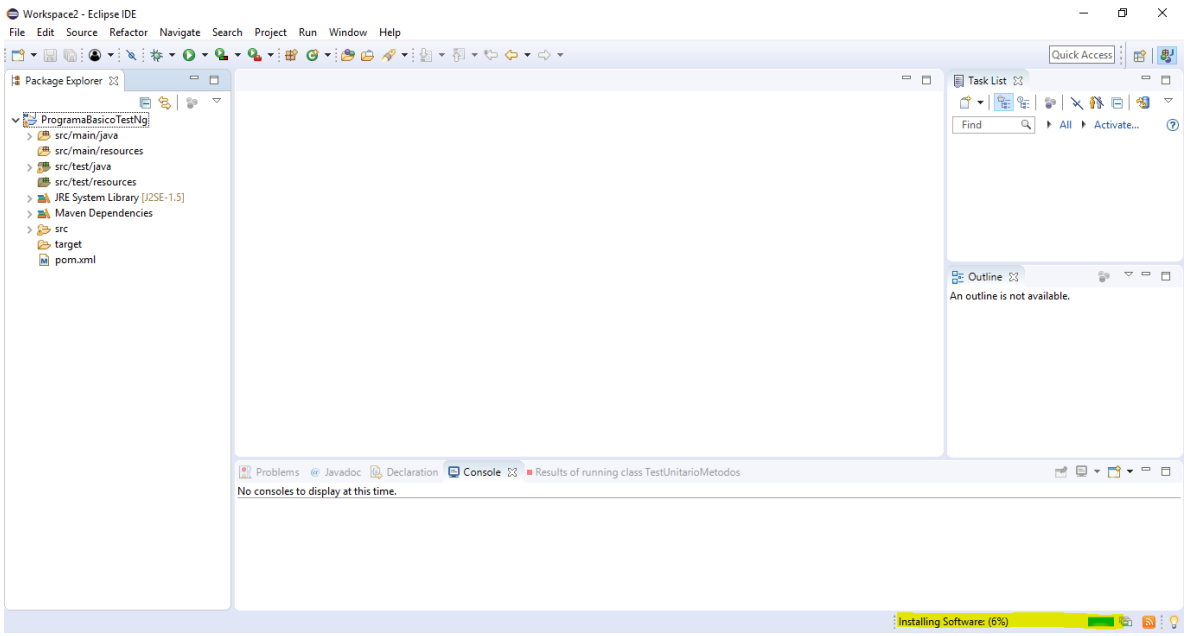
4.33. Seleccione los checks de las opciones TestNg(required) y TestNG M2E (Maven) Integration (Optional), después de esto realice click sobre el botón Confirm >



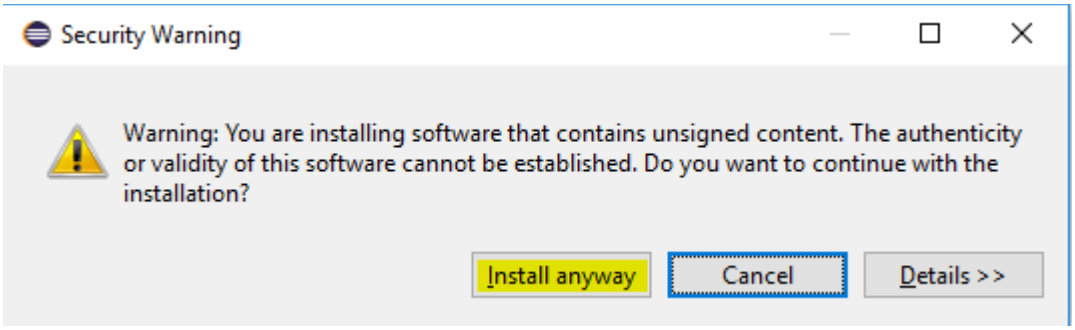
4.34. Acepte los términos y condiciones, finalmente realice click sobre el botón Finish.



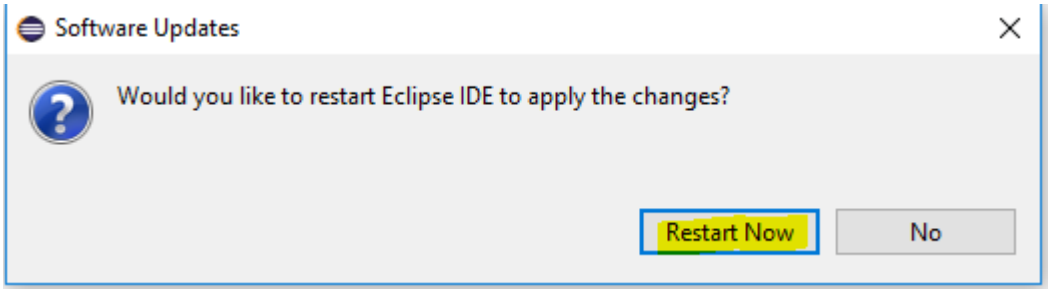
4.35. Espere que el nuevo plugin se instale completamente.



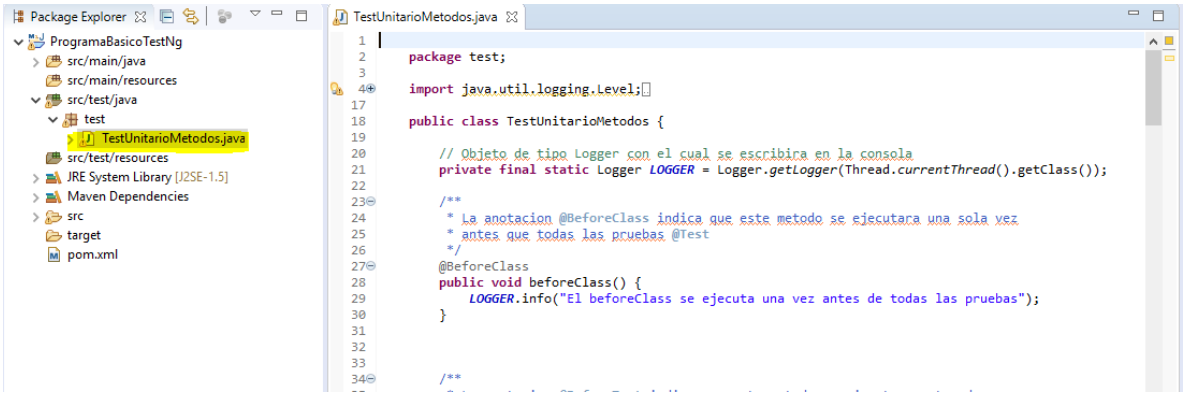
4.36. Si se muestra una ventana emergente con una alerta de contenido no firmado digitalmente por fuentes confiables, seleccione la opción Install anyway.



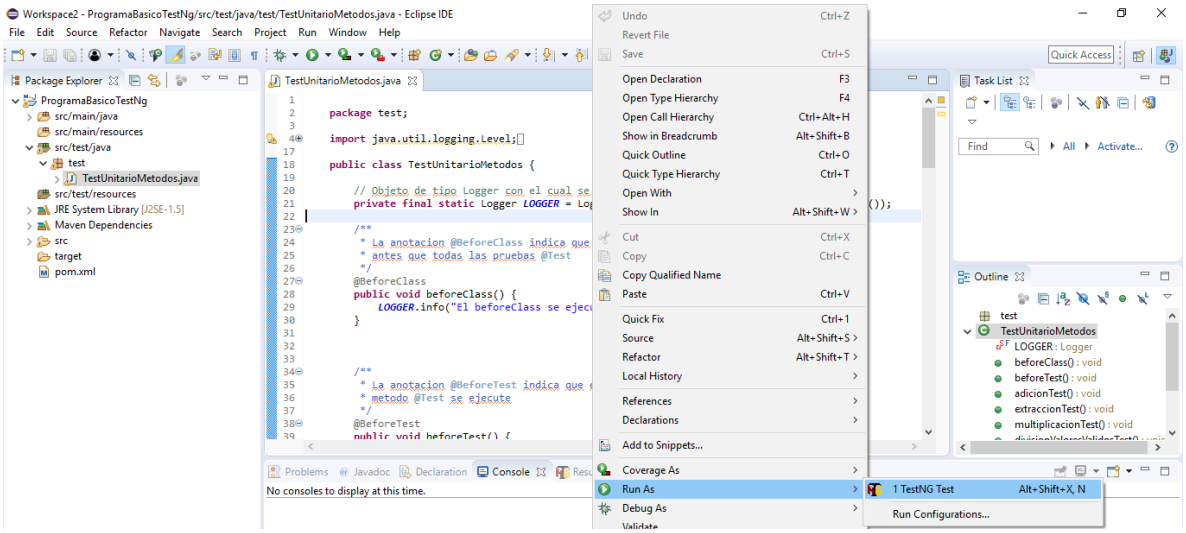
4.37. Al finalizar la instalación del plugin, se mostrará un mensaje que solicita el reinicio de Eclipse, por lo cual guarde todos los cambios pendientes y seleccione la opción Restart Now.



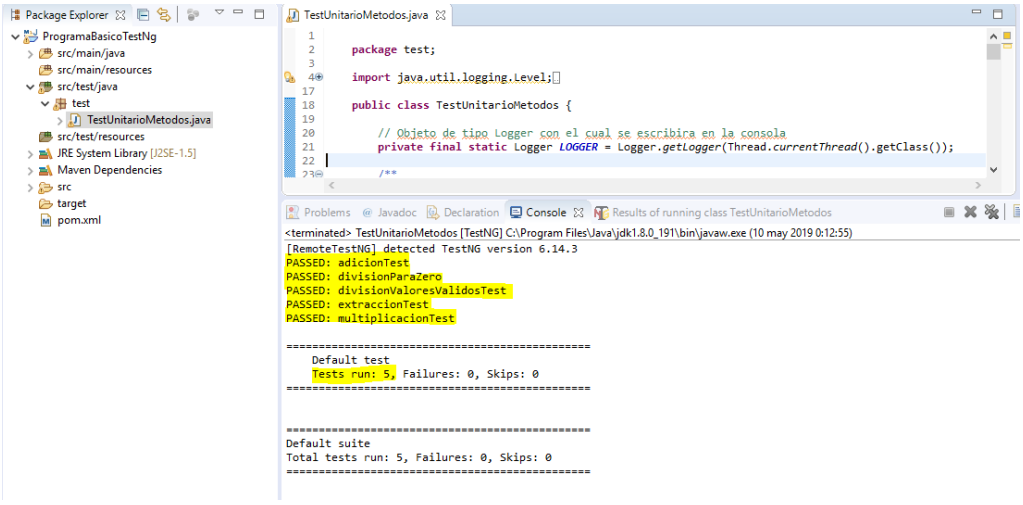
4.38. Después del reinicio de Eclipse, realice doble click sobre la clase TestUnitariosMetodos.java para abrirla.



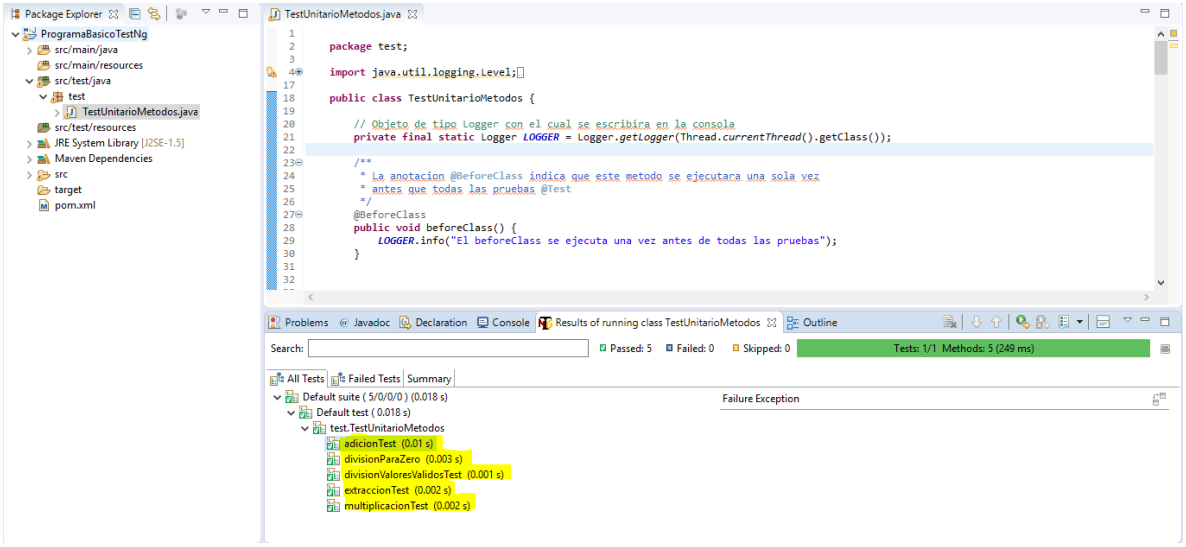
4.39. Sobre el código abierto de la clase TestUnitariosMetodos.java realice click derecho, en el menú emergente seleccione la opción Run As → TestNg Test, esto ejecutará todas las pruebas unitarias de esta clase.



4.40. En la ventana Console, podrá observar que las 5 pruebas escritas anteriormente fueron ejecutadas.



4.41. En la ventana Results of running class TestUnitarioMetodos, pobra observar la ejecución de manera exitosa de las 5 pruebas escritas anteriormente.

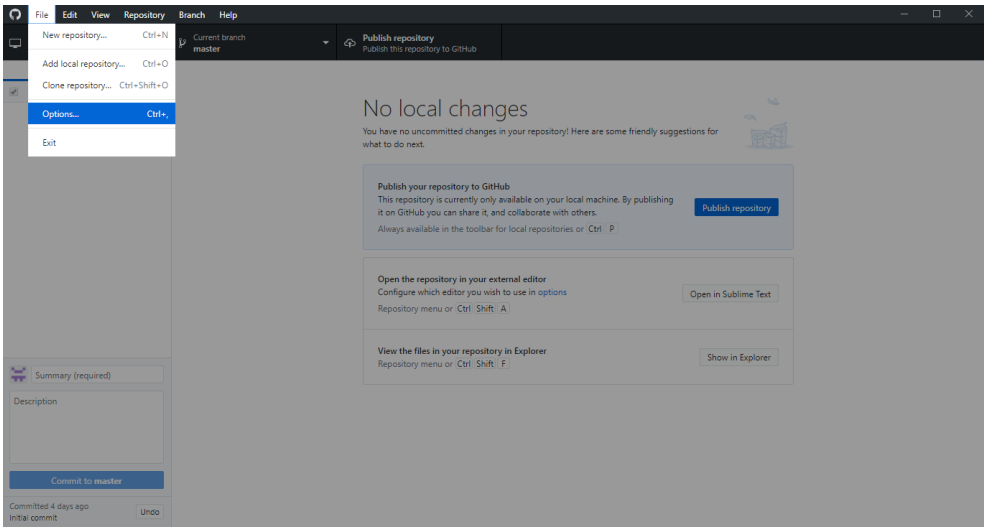


4.42. Finalmente tendrá un programa básico en Java y Maven, con pruebas unitarias sobre sus métodos usando el framework TestNg.

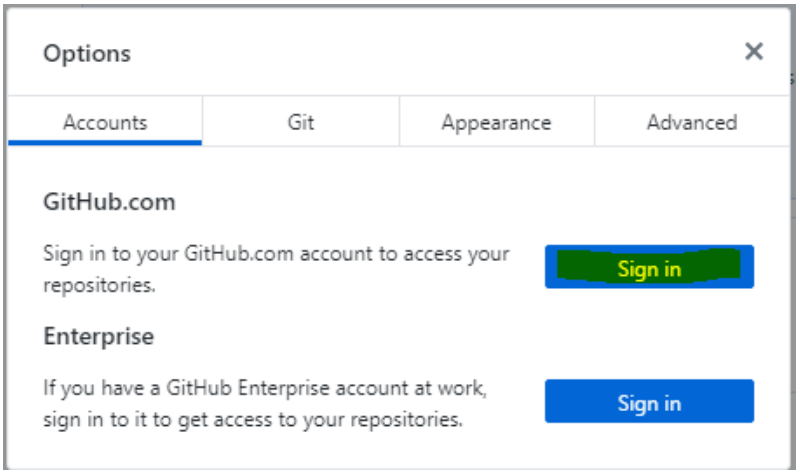
5. VERSIONAMIENTO DE PROYECTO JAVA EN GITHUB

Para realizar el versionamiento del proyecto Java se utilizará la cuenta de GitHub creada anteriormente y el software de GitHub Desktop instalado

5.1. Ejecute el programa GitHub Desktop → realice click sobre la opción File → Options..



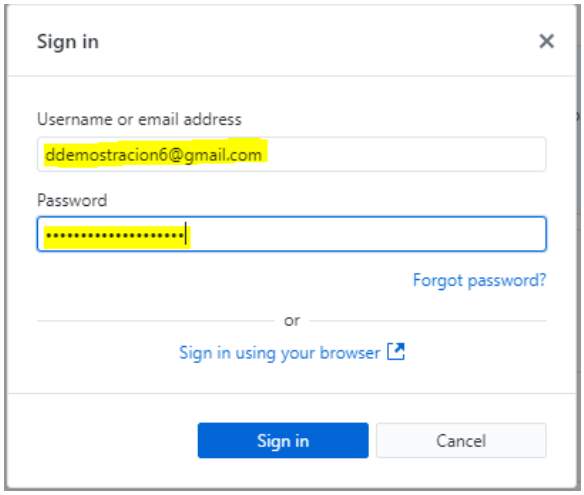
5.2. En la ventana emergente seleccione la pestaña Accounts y luego realice click en el botón Sign in perteneciente a GitHub.com



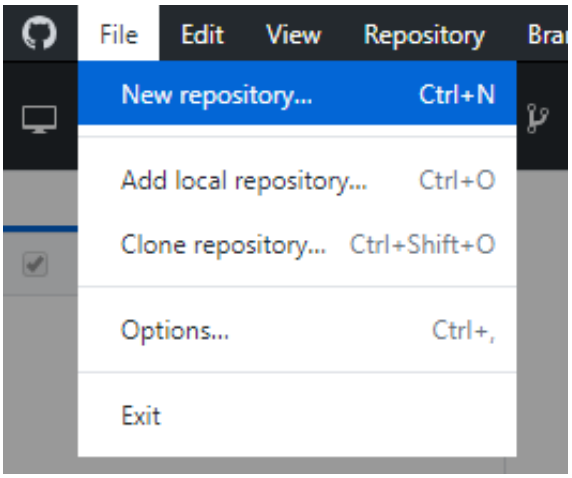
5.3. En la ventana siguiente agregue la información de la cuenta de GitHub creada anteriormente, en este caso práctico se utilizaron los datos:

Username or email address: ddemostracion6@gmail.com
Password: Passwordparamuestra1

Después de ingresar los datos solicitados, presione el botón Sign in.



5.4. El código será versionado en un repositorio, por lo cual es necesario la creación de un repositorio, realice click en File → New repository…



5.5. En la ventana emergente ingrese la información según sea solicitada:

Name: el nombre del repositorio.

Description :la descripción deseada del proyecto.

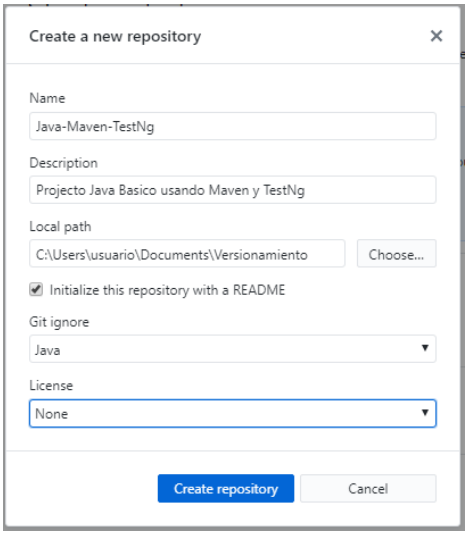
Local Path: seleccione el directorio o carpeta donde se creará el repositorio nuevo en su computador.

Check Initialize this repository with a README: al activar este check significa que se creará un archivo dentro del repositorio que puede contener información importante o relevante del proyecto.

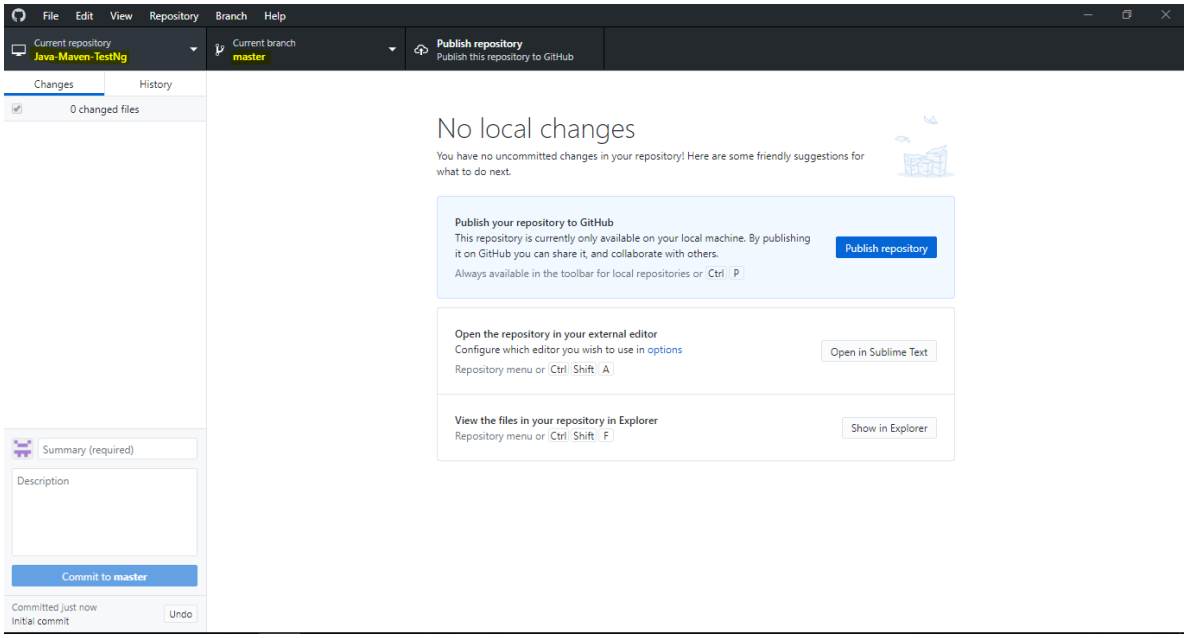
Git ignore: Este crea un archivo en el cual se puede configurar que archivos, directorios , documentos o más, que serán ignorados al versionar el proyecto.

Licence: Selección del tipo de licencia de su proyecto.

Realice click en el botón Create repository, para crear este repositorio en su equipo local, este repositorio aún no se encuentra en github.com.

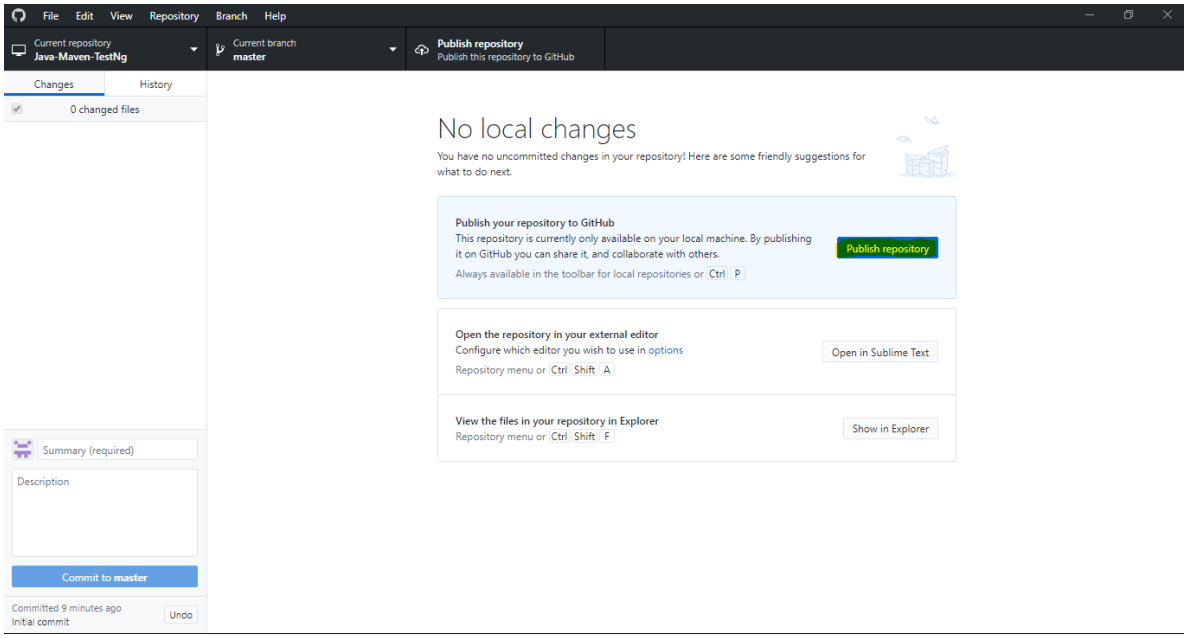


5.6. Después de la creación del repositorio, podrá observar una ventana similar a la siguiente.



- **Current repository**: El repositorio creado en su máquina local.
- **Current Branch**: La rama creada automáticamente de su repositorio.

5.7. Para publicar su repositorio en github.com, realice click en el botón publish repository.



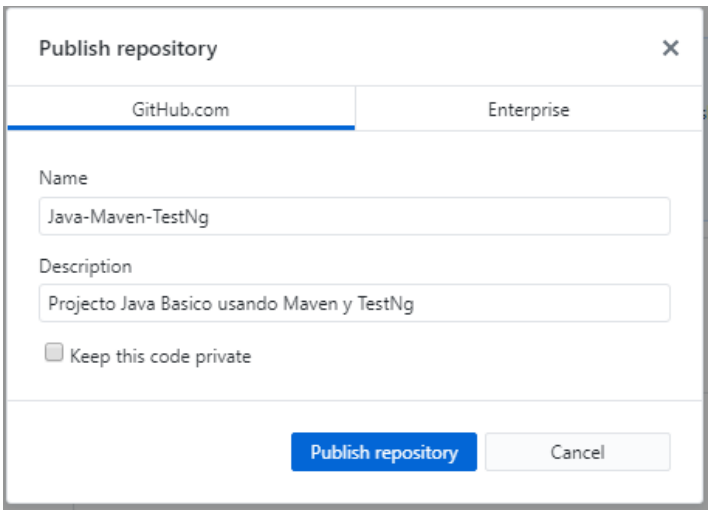
5.8. En la ventana emergente seleccione la pestaña GitHub.com y llene los campos solicitados.

Name: Ingrese con que nombre será publicado el repositorio de su máquina local en github.com, se recomienda utilizar el mismo nombre que en su máquina local.

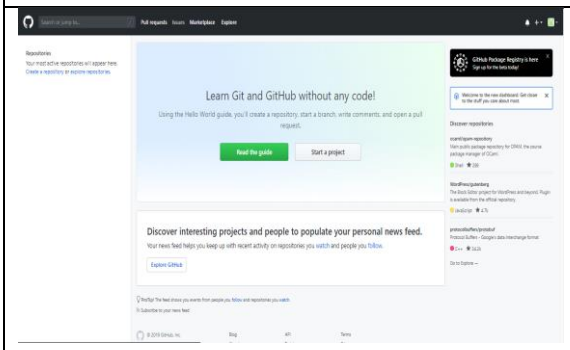
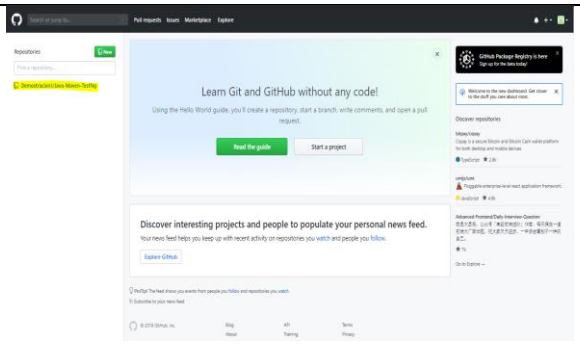
Description: Ingrese la descripción de su proyecto.

Check Keep this code private: Si se activa este check, el repositorio será privado, en este ejemplo práctico se creará el repositorio público.

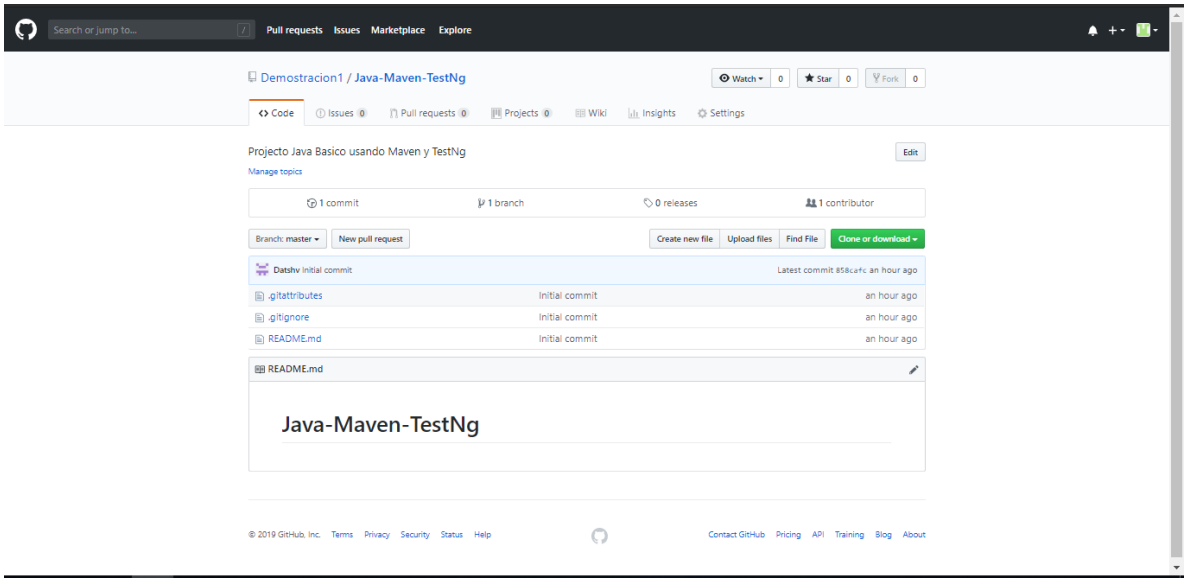
Finalmente realice click sobre el botón Publish repository.



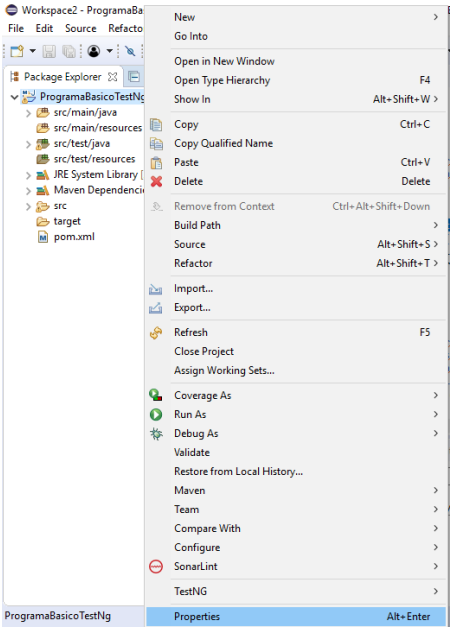
5.9. Al finalizar el proceso de publicación, puede observar el nuevo repositorio creado en su cuenta de github.com.

ANTES	DESPUÉS
	

5.10. Si abre el repositorio creado en GitHub.com observará una ventana similar a la siguiente.



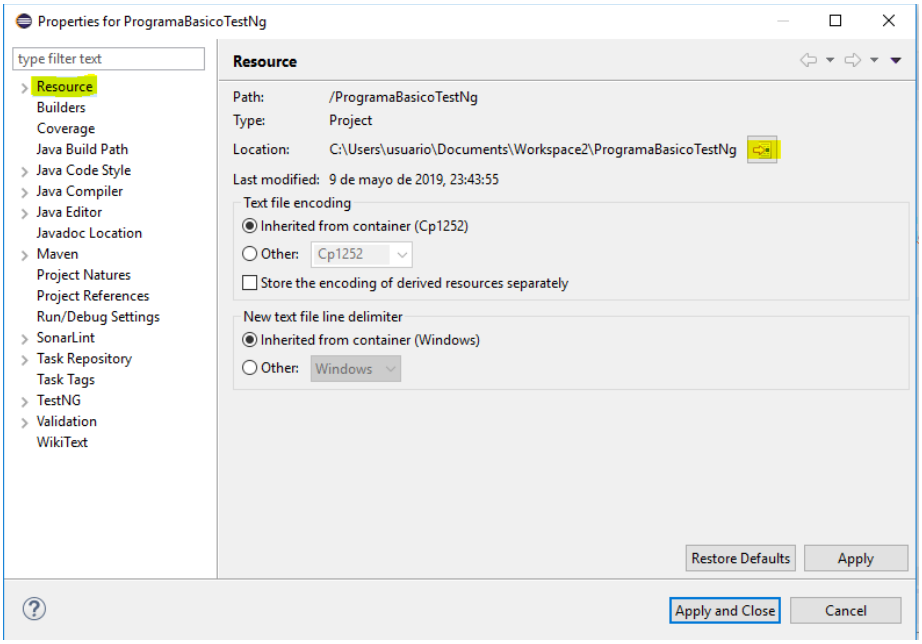
5.11. Actualmente tiene el repositorio creado en GitHub.com y conectado a su máquina local, pero aún no ha versionado su proyecto en el repositorio y para esto es necesario conocer en el equipo local donde se encuentra su proyecto Java, por lo que ejecutará Eclipse y se ubicará sobre el proyecto creado anteriormente → click derecho sobre el proyecto → properties.



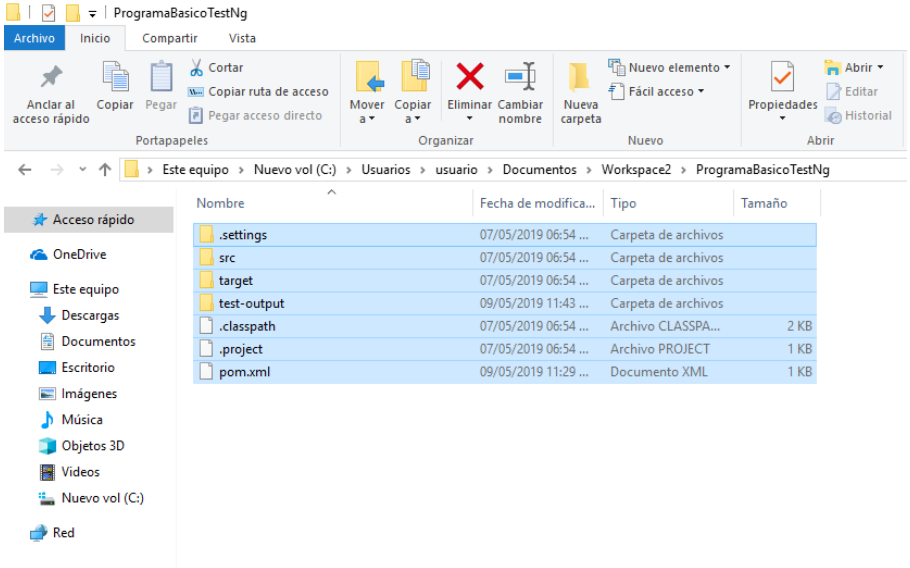
5.12. En la ventana emergente, seleccione la pestaña > Resource y realice click en el botón que esta frente al directorio del proyecto.

En caso de que el botón no se encuentre habilitado copie la ruta que se encuentra en Location, en este caso sería: C:\Users\usuario\Documents\Workspace2\ProgramaBasicoTestNg

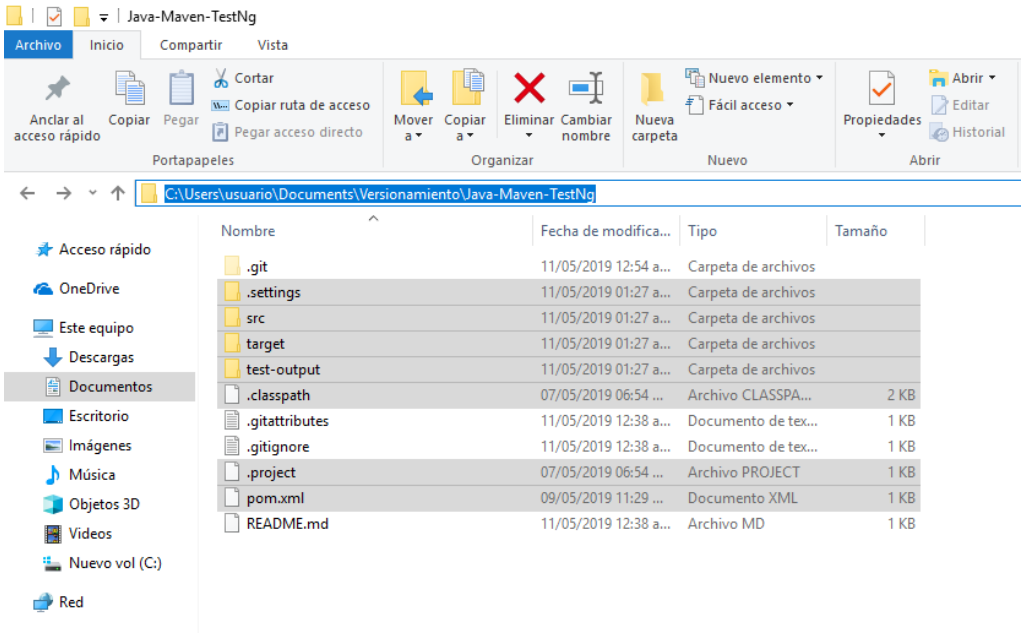
Finalmente busque esta ruta en su máquina local.



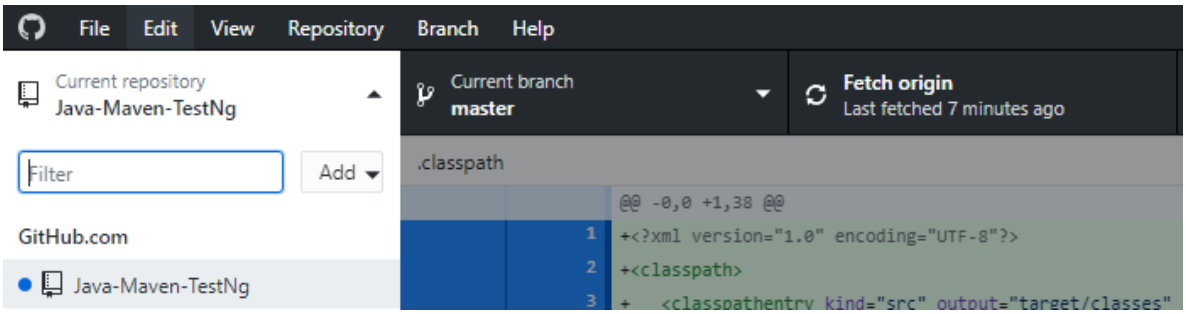
5.13. Ubicado en la carpeta que contiene su proyecto, copie toda la información.



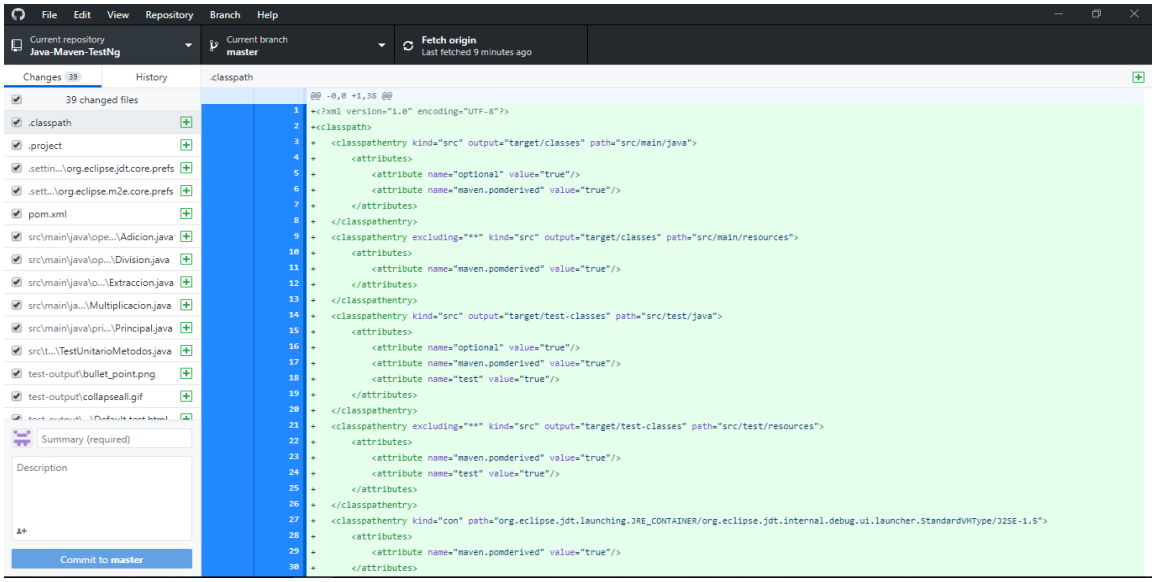
5.14. Los archivos copiados, péguelos en la carpeta donde creo el repositorio local, los archivos contenidos en la carpeta del repositorio lucirán similar a la siguiente imagen.



5.15. Ejecute el programa GitHub Desktop, realice click sobre la opción Current repository y seleccione el repositorio creado anteriormente.



5.16. Se visualizarán los archivos de su proyecto Java para ser versionados en GitHub.com

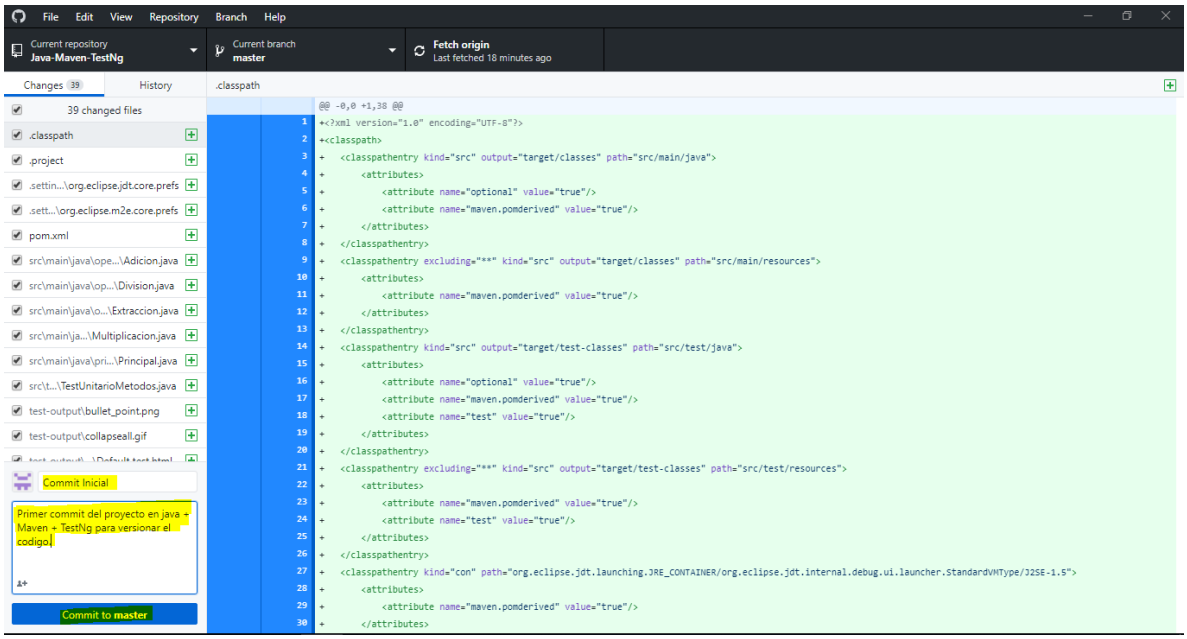


5.17. Para versionar estos archivos llene en la ventana principal de GitHub Desktop los campos:

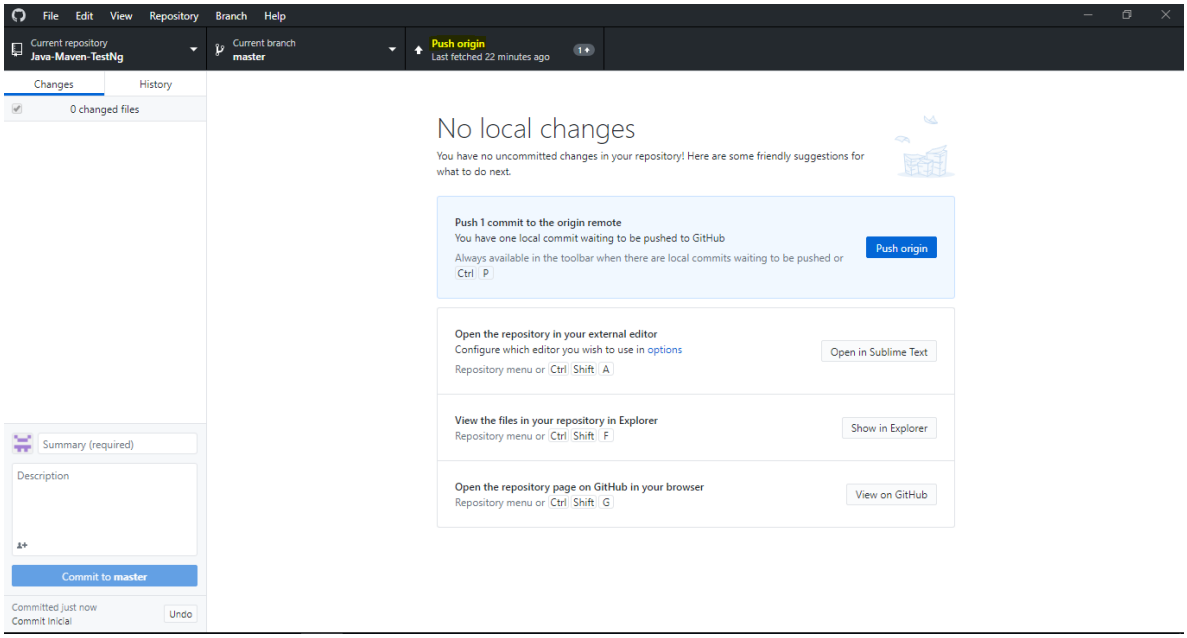
Summary (required): Resumen de pocas palabras indicando los cambios realizados.

Description: Descripción de los cambios realizados.

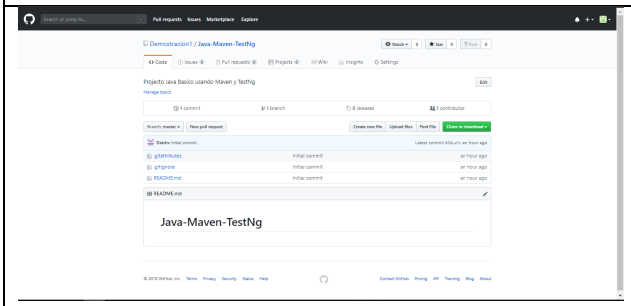
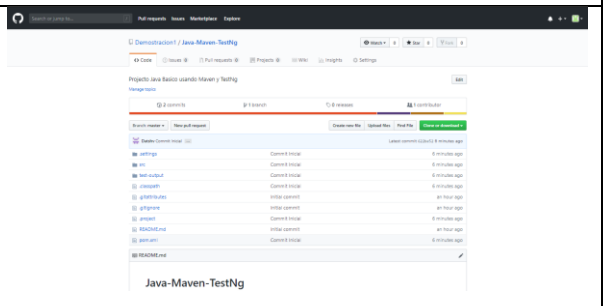
Después de llenar los campos → realice click sobre el botón Commit to master



5.18. Después de realizar click en commit to master, los cambios aún se encuentran en su máquina local, por lo cual es necesario realizar click en la opción push origin para enviar los cambios a GitHub.com



5.19. Al finalizar el push origin, ingrese a la cuenta de GitHub.com donde se versiono el proyecto y seleccione el repositorio creado, podrá observar el proyecto versionado

Antes del commit y push origin	Después del commit y push origin
	

5.20. El proyecto en Eclipse se encuentra duplicado en dos carpetas.

La carpeta original donde se creó el proyecto Java de Eclipse:

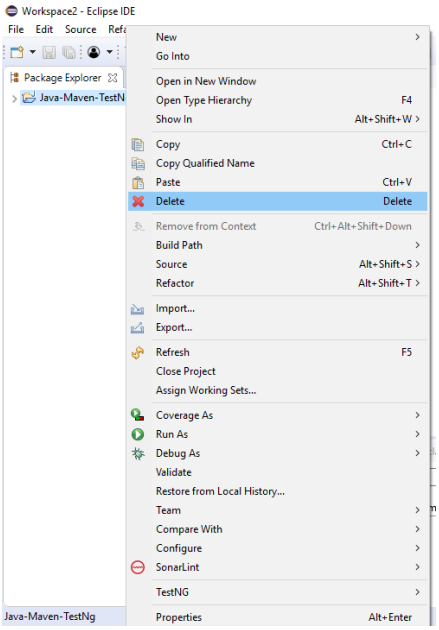
- C:\Users\usuario\Documents\Workspace2\Java-Maven-TestNg

La carpeta donde se encuentra versionado el código:

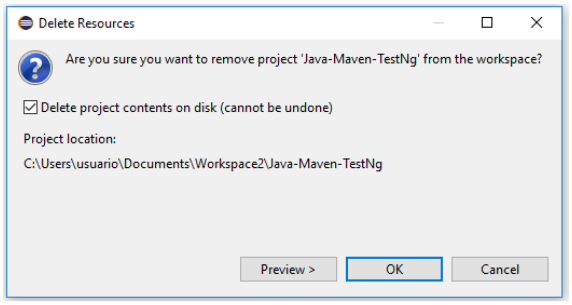
- C:\Users\usuario\Documents\Versionamiento\Java-Maven-TestNg

Se tiene que conectar la carpeta con el proyecto versionado a Eclipse.

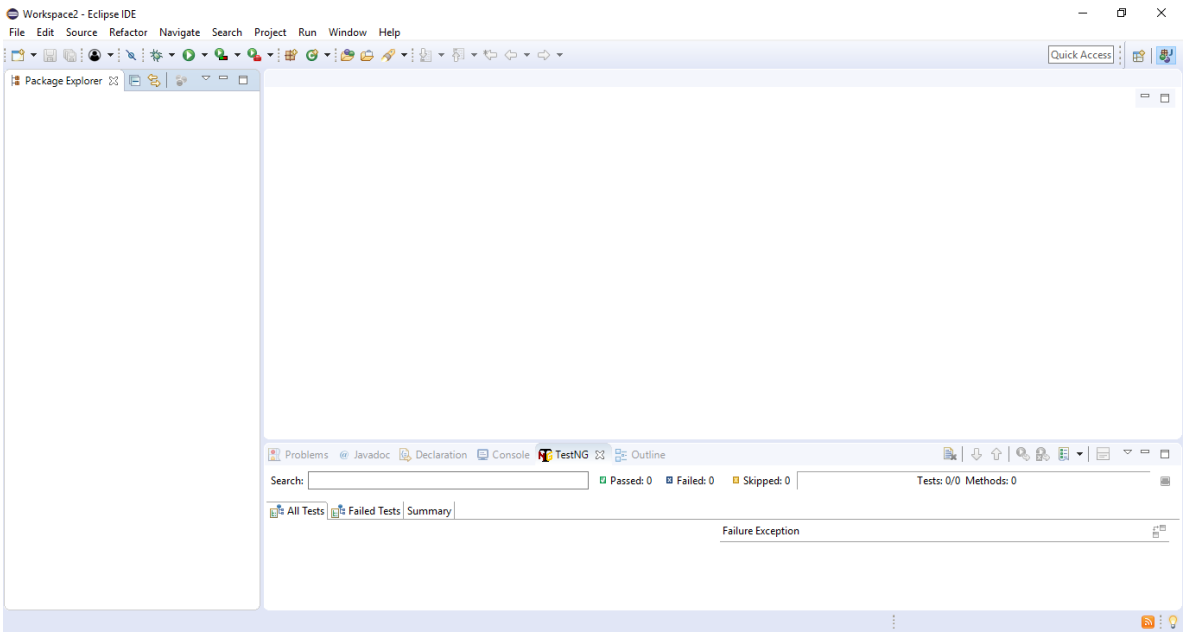
5.21. Ejecute Eclipse, busque el proyecto originalmente creado y elimínelo, realizando click derecho sobre el nombre del proyecto → click sobre la opción Delete.



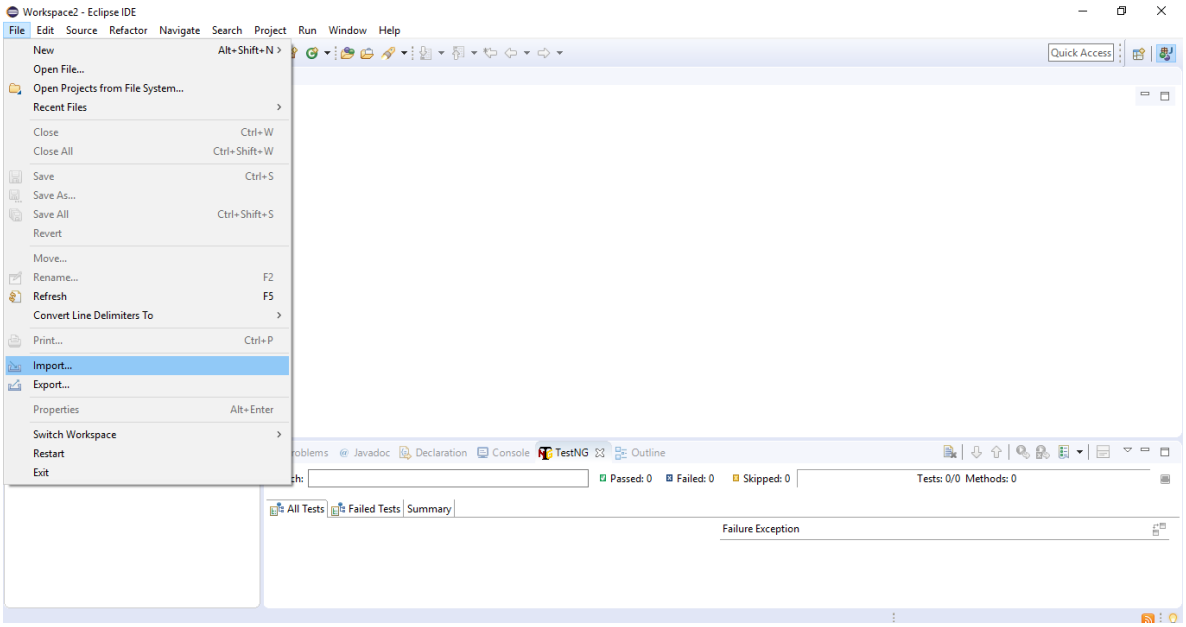
5.22. En la ventana de confirmación borrado, marque como activo el check “Delete Project contents on disk (cannot be undone)” y realice click sobre el botón OK



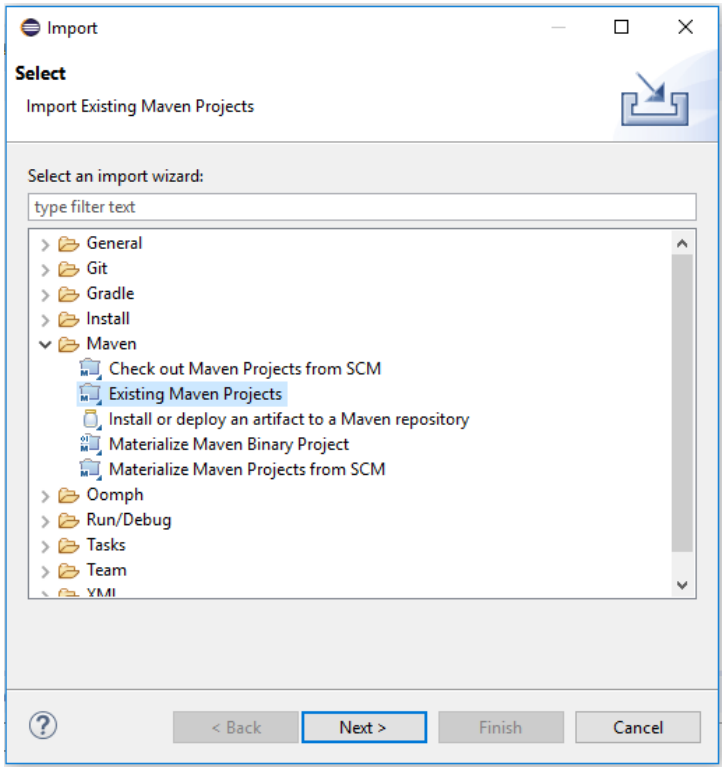
5.23. Después de borrar el proyecto, este desaparece de su área de trabajo en Eclipse.



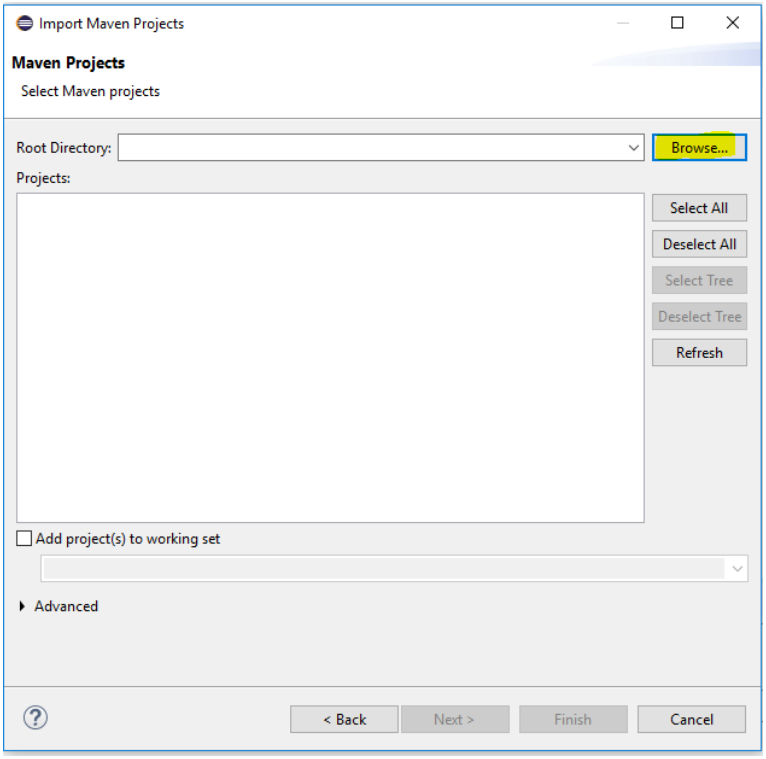
5.24. Importe el proyecto versionado a su área de trabajo en eclipse, para esto realice click sobre el menú File → Import…



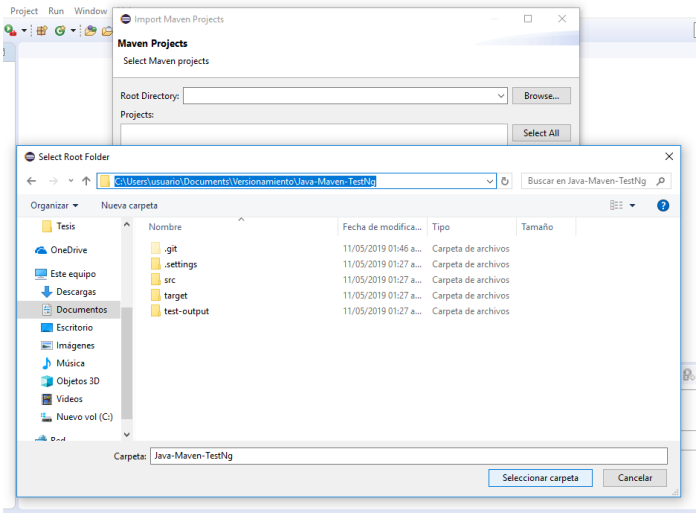
5.25. En la ventana emergente seleccione Maven → Existing Maven Projects → Next



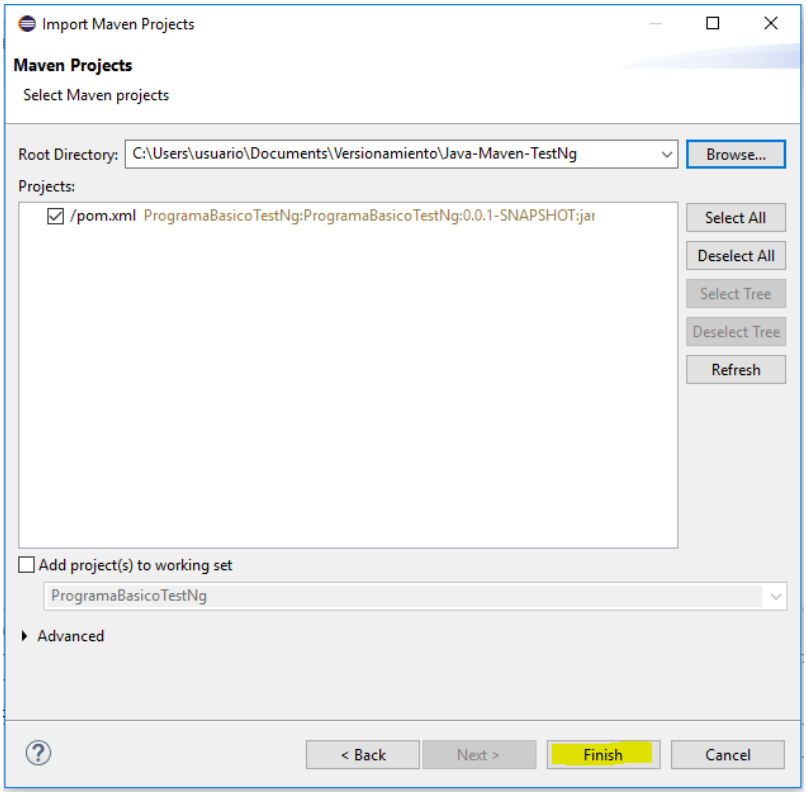
5.26. En la ventana siguiente oprima el botón Browse para buscar el proyecto Maven en su computadora



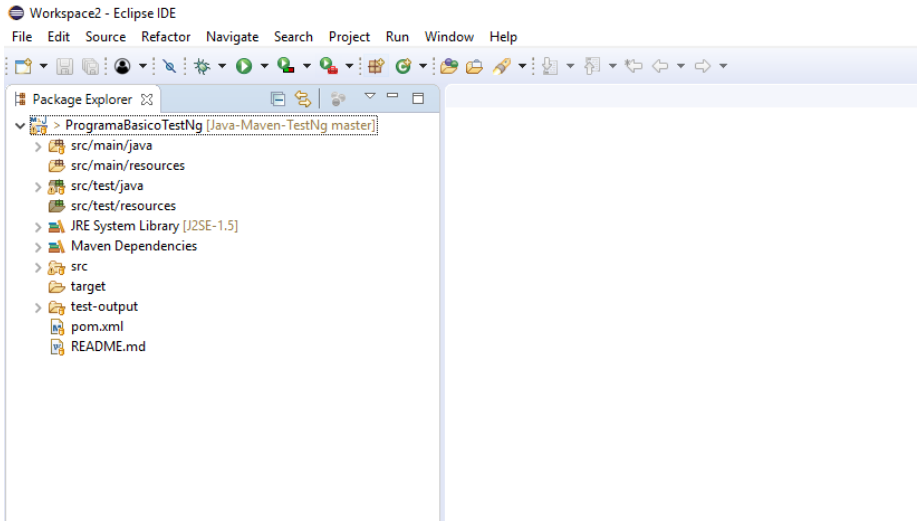
5.27. Seleccione el directorio en el cual se encuentra su repositorio y el proyecto versionado, en este caso práctico era:
C:\Users\usuario\Documents\Versionamiento\Java-Maven-TestNg
Luego de encontrar el directorio, oprima el botón seleccionar carpeta



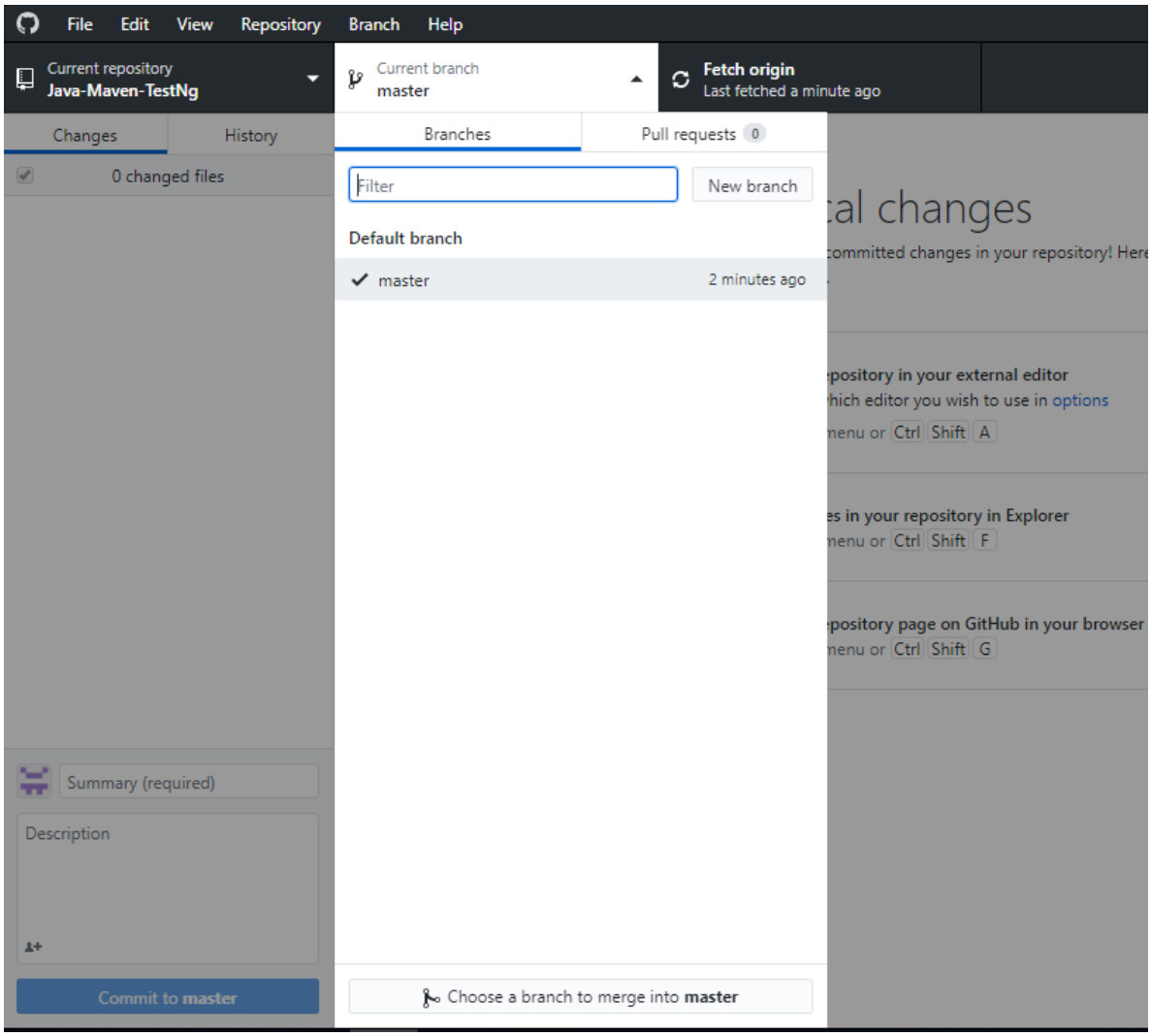
5.28. Automáticamente el sistema encontrará el archivo pom.xml, el cual contiene la configuración de su proyecto, después de esto realice click en el botón finish



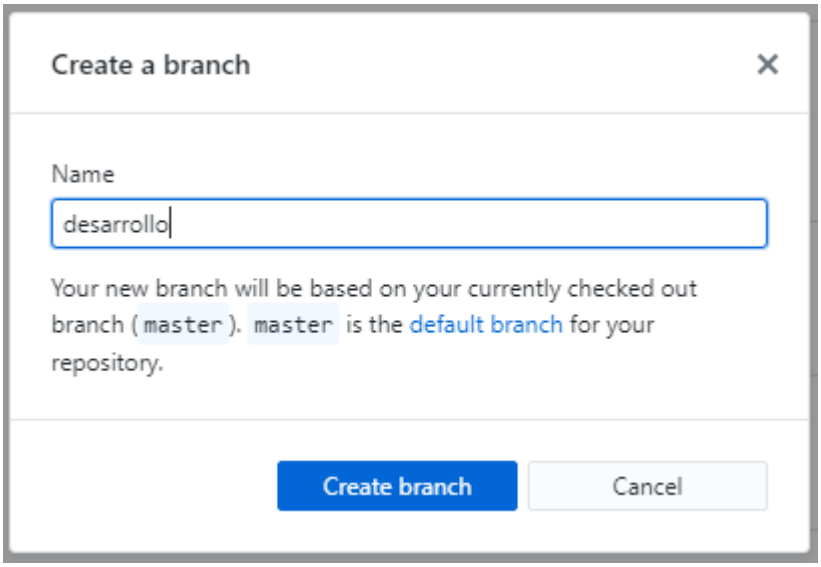
5.29. Ahora el proyecto versionado se encuentra en su área de trabajo de Eclipse y se visualiza del siguiente modo.



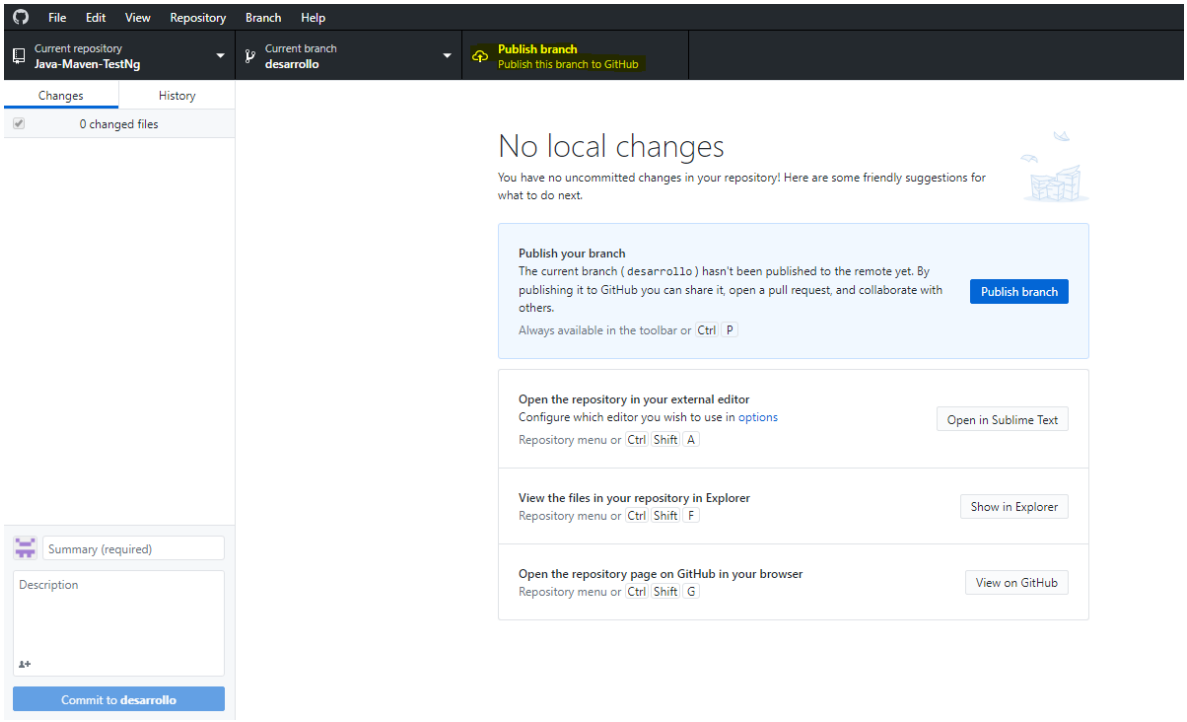
5.30. Ahora es necesario la creación de una nueva rama en github para su proyecto, en la cual se harán los cambios necesarios y finalmente cuando estos cambios sean definitivos, se realizará un pull request para unirlos con la rama master. Para la creación de una nueva rama a partir de la rama master, ejecute el software GitHub Desktop, seleccione la opción Current branch→ Branches → realice click en el botón New branch



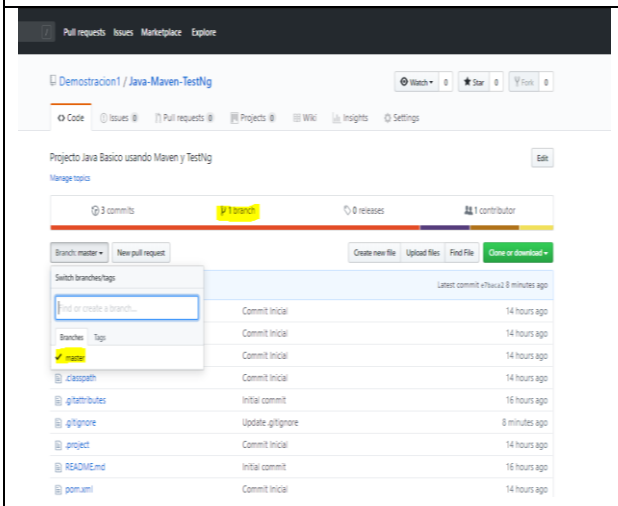
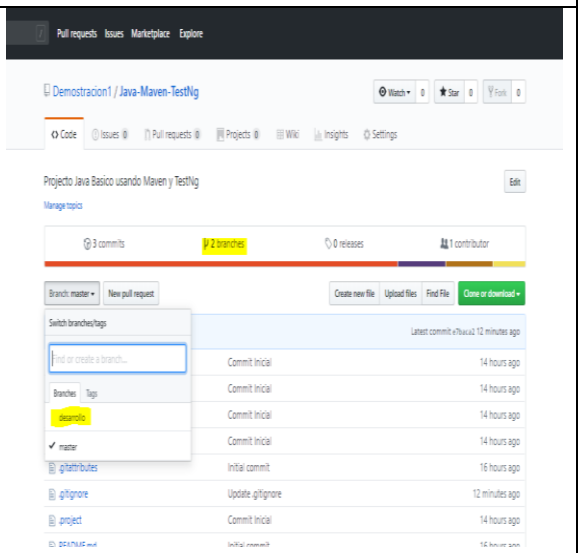
5.31. En la ventana siguiente agregue el nombre de esta nueva rama, en este caso práctico se creará la rama desarrollo, la cual será una copia de la rama master, luego de agregar el nombre, realice click en el botón Create branch



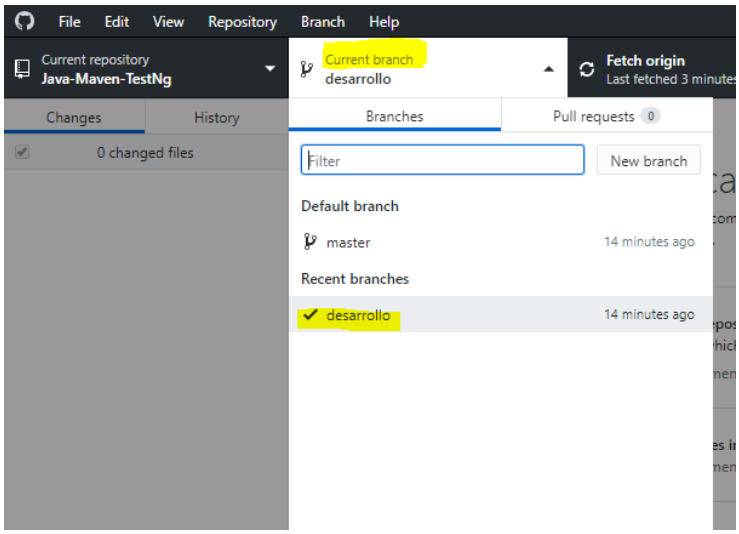
5.32. Realice click sobre el botón Publish branch, para publicar esta nueva rama en el repositorio de GitHub.com



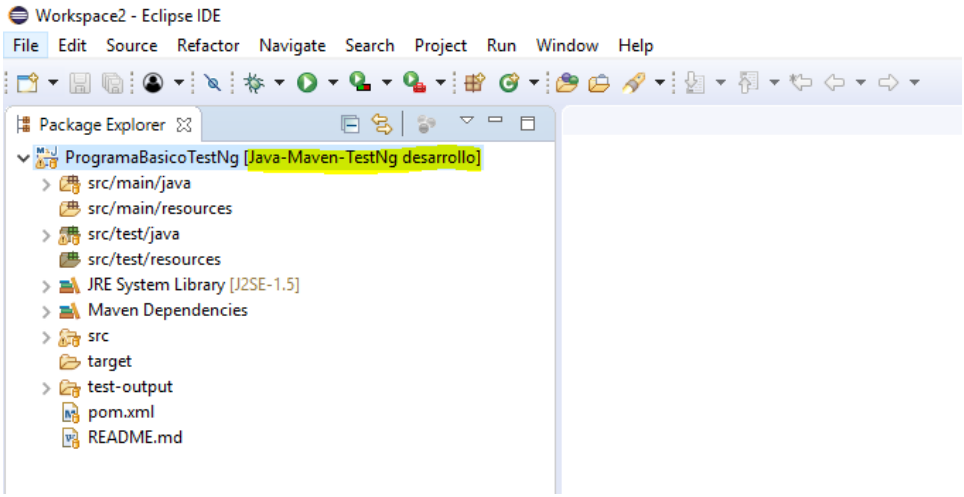
5.33. Después de crear la rama desarrollo, ingrese a GitHub.com con la cuenta creada anteriormente y ubíquese en el repositorio de proyecto java, podrá observar como la nueva rama fue creada.

ANTES DE LA CREACIÓN DE LA RAMA	DESPUÉS DE LA CREACIÓN DE LA RAMA
	

5.34. En la herramienta GitHub Desktop seleccione la rama sobre la cual va a trabajar, en este caso se seleccionará la rama desarrollo, para esto realice click sobre la opción Current branch → desarrollo



5.35. Ejecute Eclipse y ubíquese sobre el proyecto Java, podrá observar que frente al nombre del proyecto se encuentra el nombre del repositorio y la rama, en GitHub Desktop esta seleccionada la rama desarrollo, significa que los cambios que se realizan en el proyecto serán versionados en la rama desarrollo, si se cambia la Current branch en GitHub desktop, esta también cambiará en el proyecto en Eclipse.

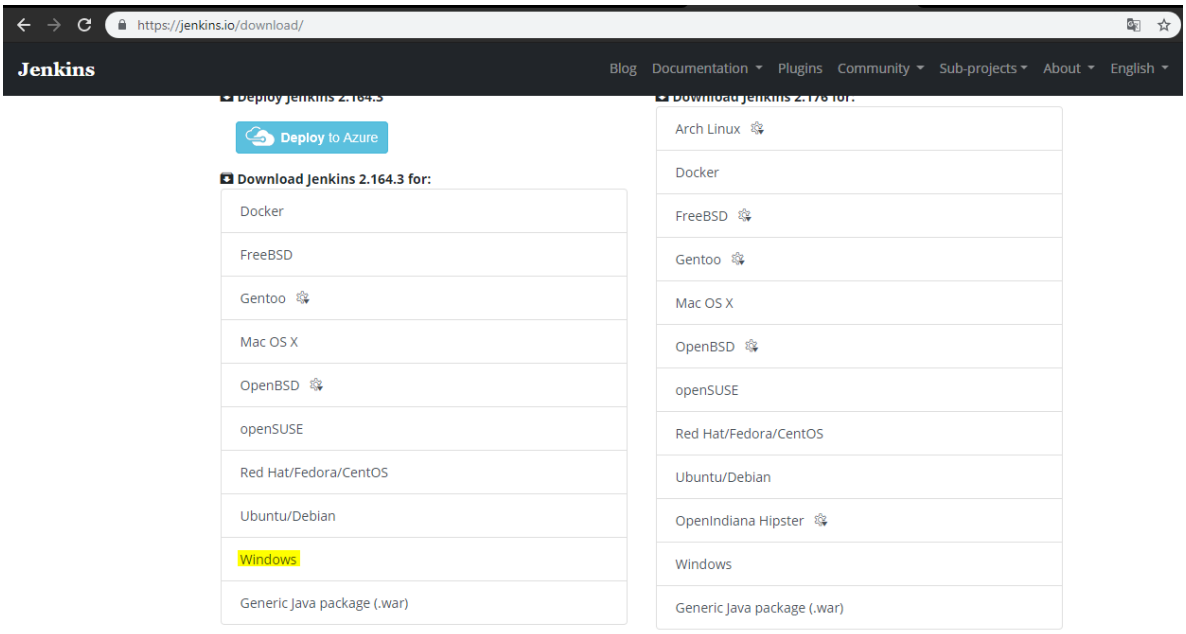


6. INSTALACIÓN DE JENKINS

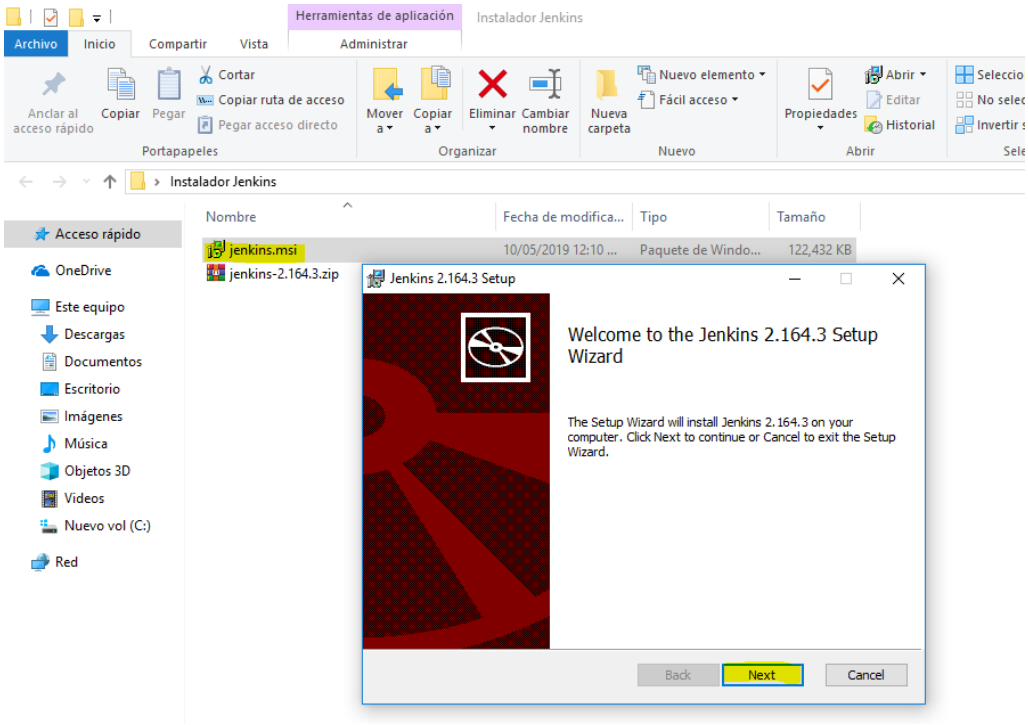
Para la ejecución de manera automática de las pruebas unitarias del código versionado en GitHub es necesario el uso de Jenkins como un servidor de integración continua, en esta sección se muestra su instalación.

6.1. Ingrese a la página https://jenkins.io/download/ y realice click sobre el botón Download.

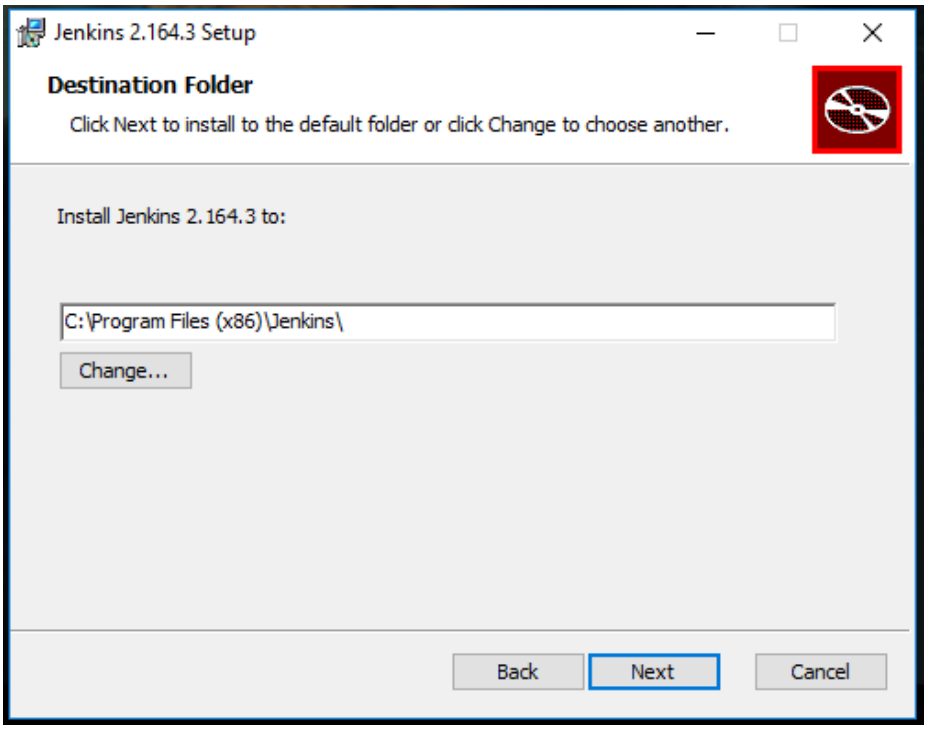
6.2. En la sección inferior de la página, seleccione el sistema operativo en el cual se instalará GitHub, en este caso Windows



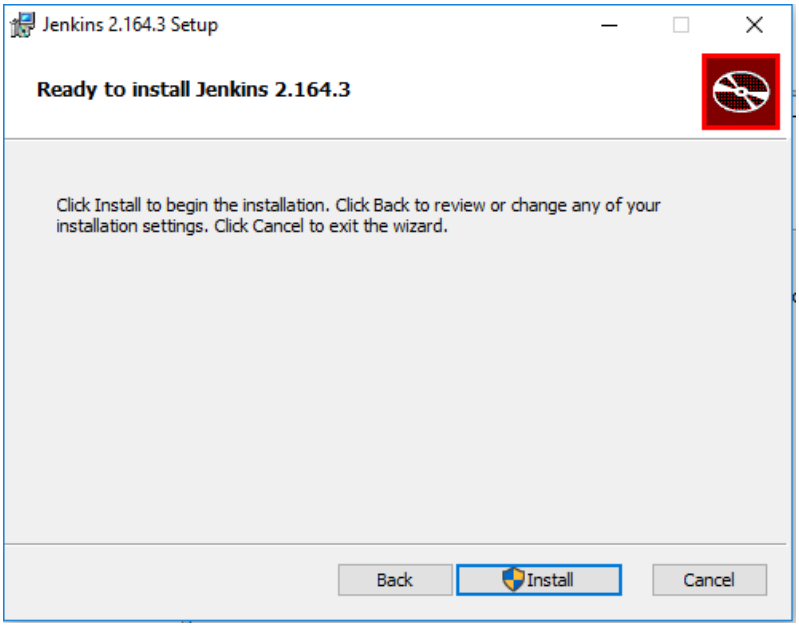
6.3. Después de que la descarga se complete, descomprima el archivo.zip y realice doble click sobre el instalador → oprima el botón Next.



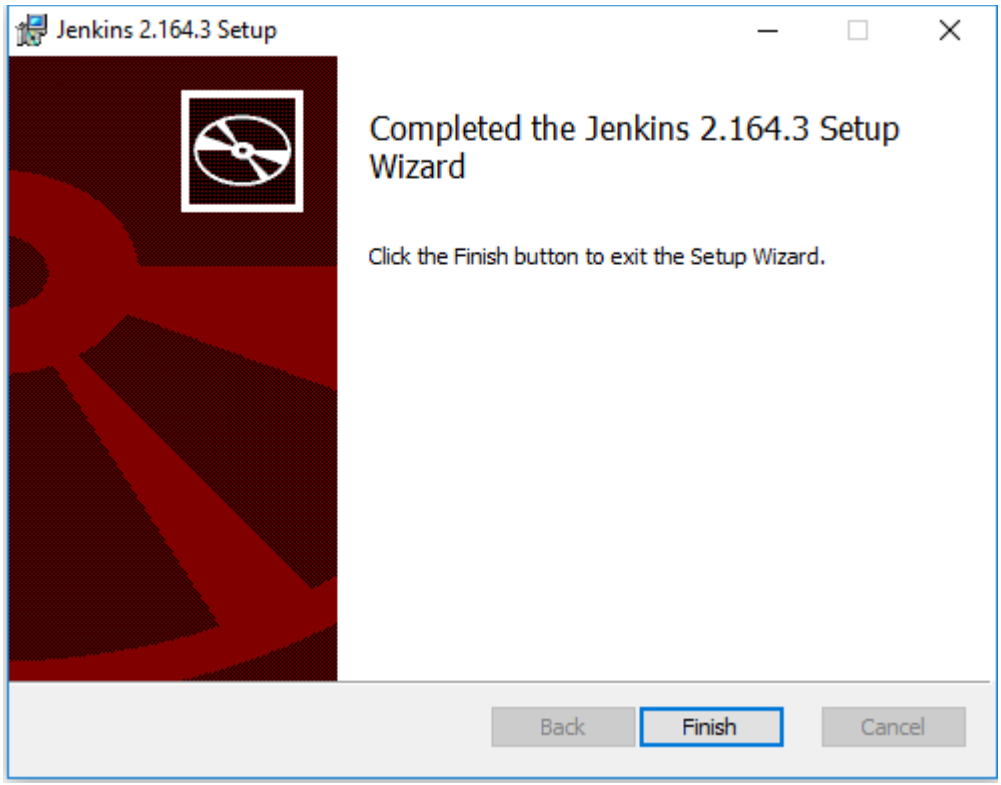
6.4. Seleccione la carpeta de instalación de Jenkins, en este caso se mantendrá la carpeta por default y oprima el botón Next.



6.5. Realice click sobre el botón Install y espere hasta la finalización de la instalación.

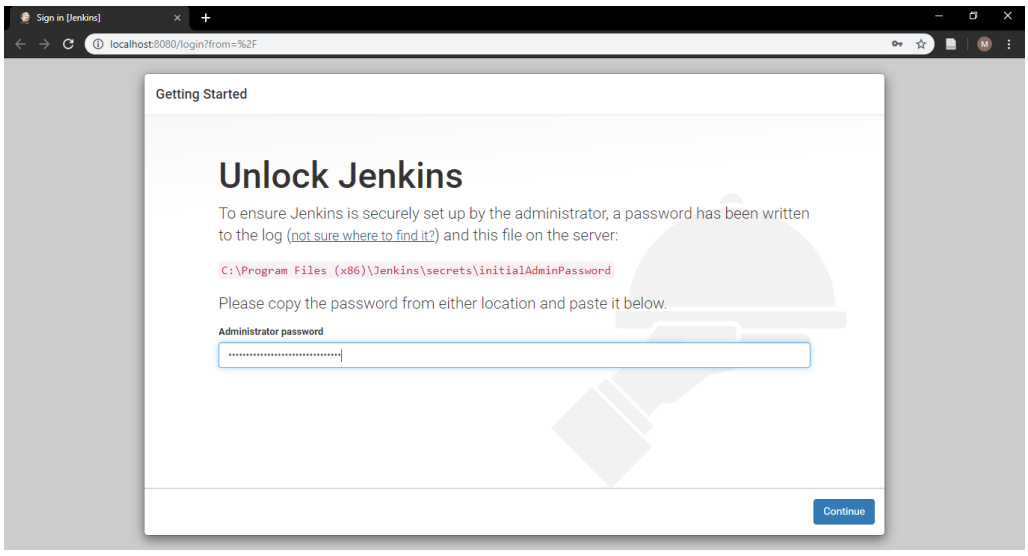


6.6. Después de que la instalación a finalizado de manera exitosa, realice click en el botón Finish.

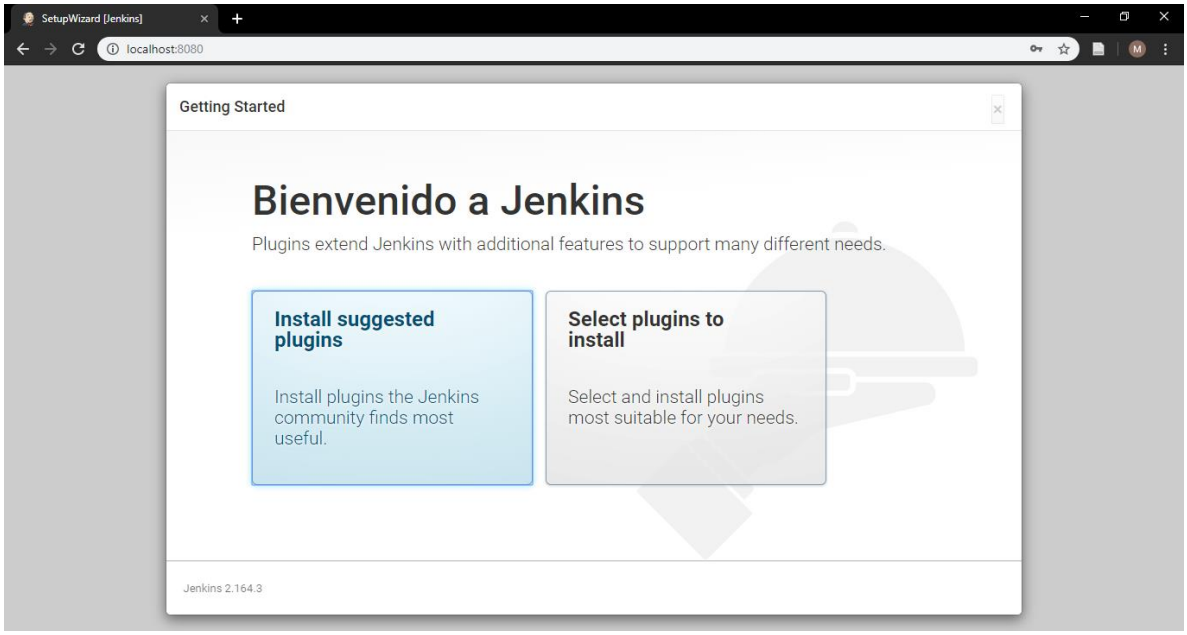


6.7. Automáticamente se abrirá el explorador predeterminado de su sistema, en este se mostrará una ventana de espera, similar a la siguiente.

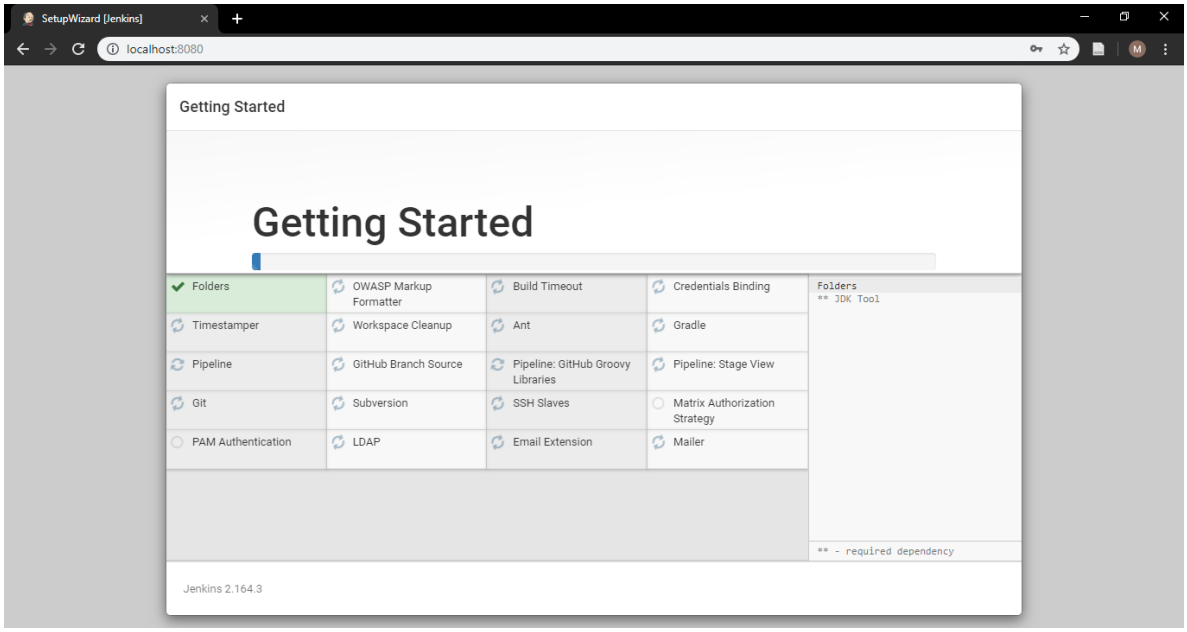
6.8. Se le solicitará una clave inicial, la cual se encuentra en la carpeta de instalación de Jenkins, en este caso práctico el directorio fue:
C:\Program Files (x86)\Jenkins\secrets\initialAdminPassword
Abra el archivo initialAdminPassword con un editor de texto, copie la contraseña y péguela en el campo Administrator password, después realice click sobre el botón Continue.



6.9. Seleccione la opción Install suggested plugins.



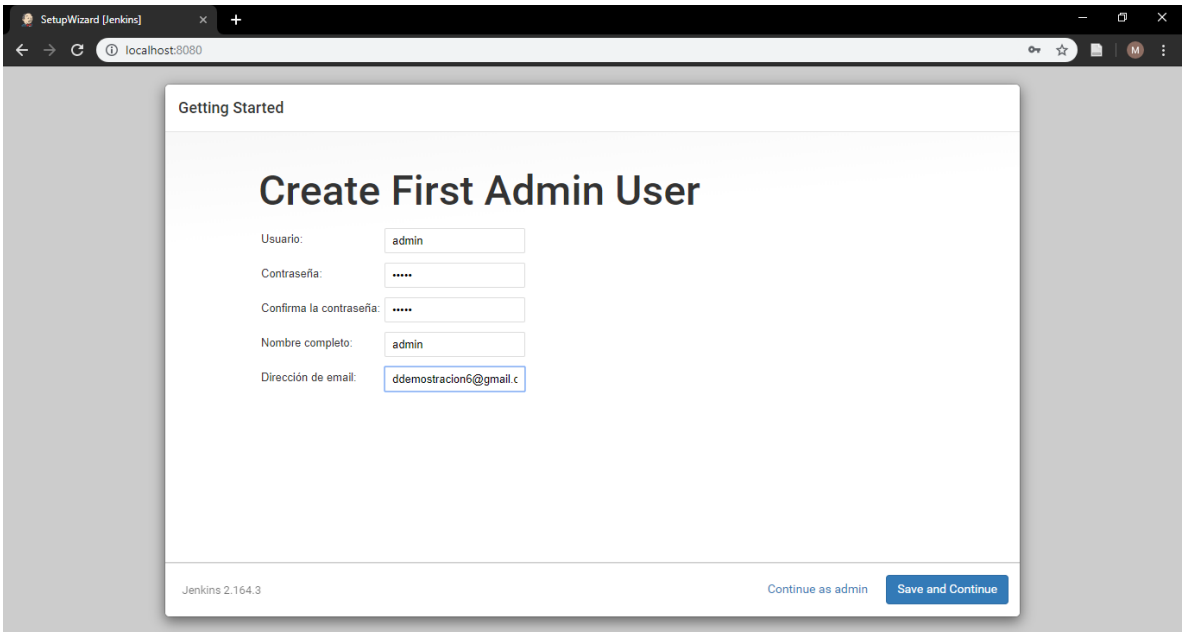
6.10. Espere hasta que la instalación finalice.



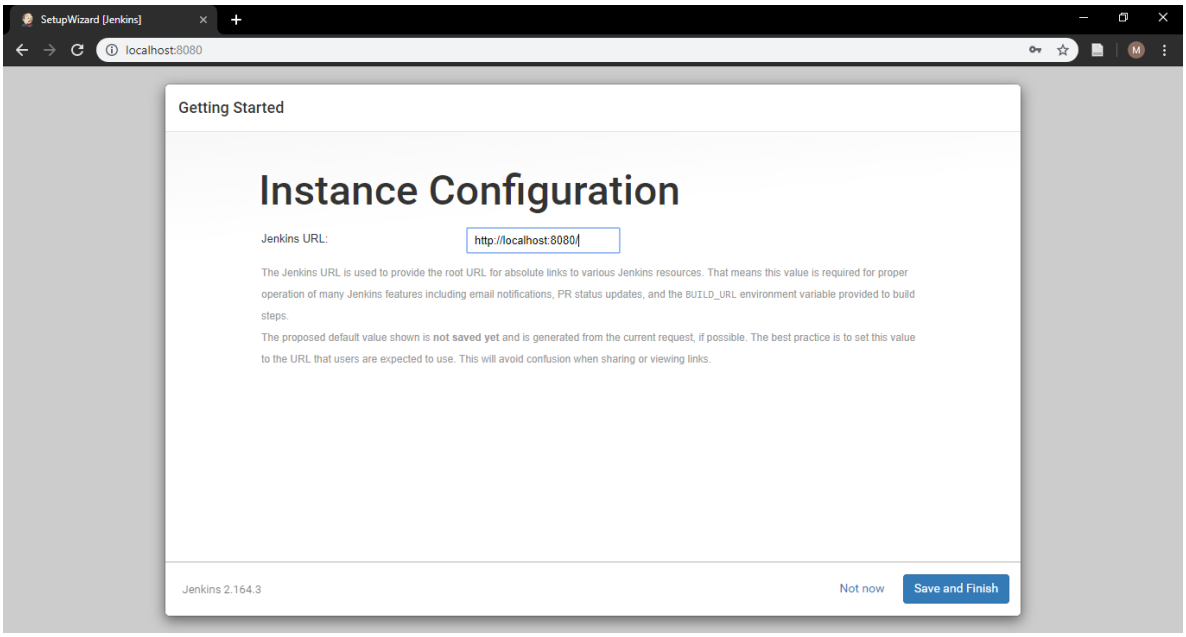
6.11. Ingrese los datos solicitados para crear el primer usuario Administrador de Jenkins, en este caso se utilizaron los siguientes datos:

- **Usuario**: admin
- **Contraseña**: admin
- **Confirma la contraseña:** admin
- **Nombre completo:** admin
- **Dirección de email:** ddemostracion6@gmail.com

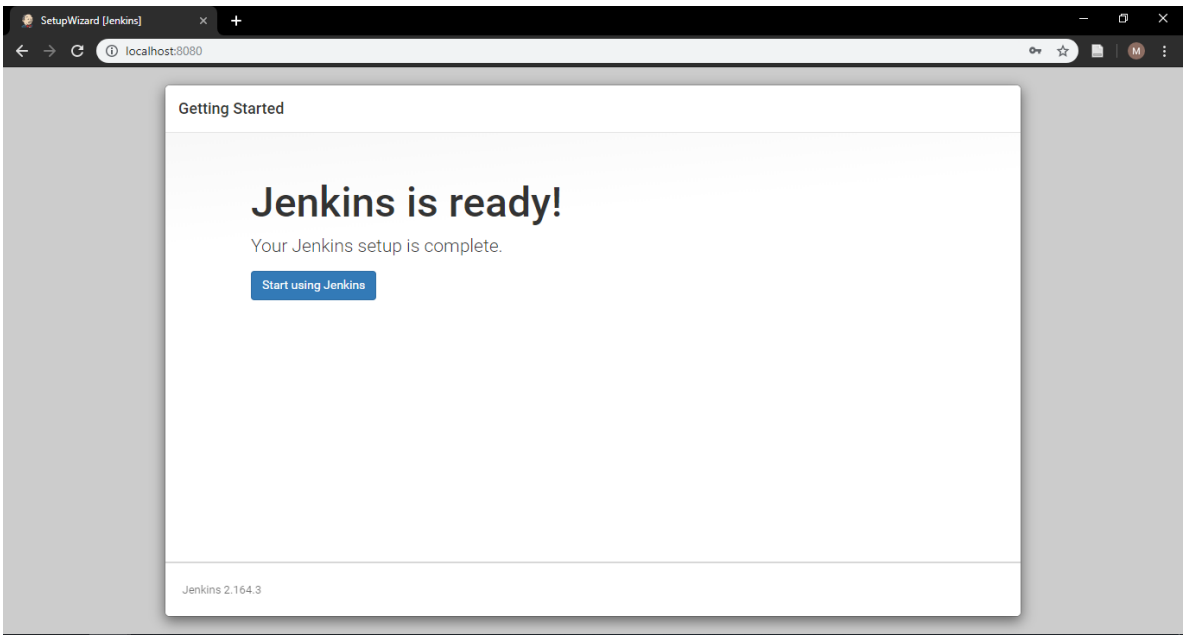
Finalmente presione el botón Save and Continue.



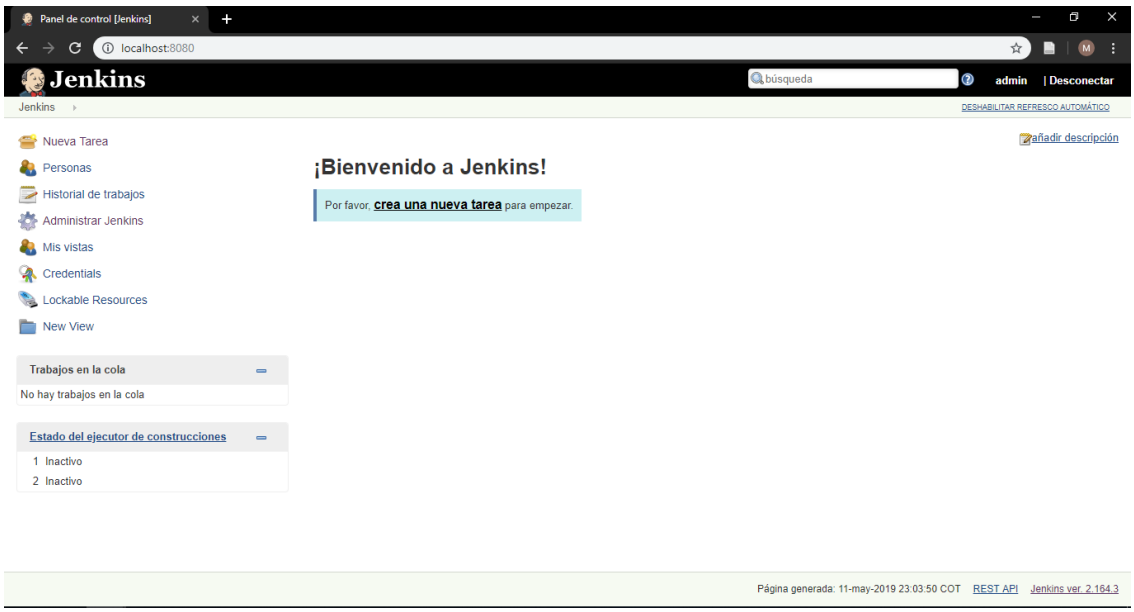
6.12. Ingrese la Url que será utilizada por Jenkins, en este caso se recomienda utilizar la que tiene por default, http://localhost:8080/ , luego oprima Save and Finish



6.13. Realice click sobre el botón Start using Jenkins



6.14. Después de la instalación Jenkins dispondrá el panel principal en el explorador.



7. INSTALACIÓN DE PLUGINS GITHUB PULL REQUEST, EMAIL EXTENSION TEMPLATE, SONARQUBE SCANNER Y SONAR QUALITY GATE EN JENKINS

En esta sección se mostrará como instalar los plugins necesarios, para la configuración entre estos está el plugin **GITHUB PULL REQUEST**, con el cual se realizarán las configuraciones y conexiones necesarias con github para que Jenkins ejecute las tareas programadas cada vez que un pull request sea creado en GitHub.

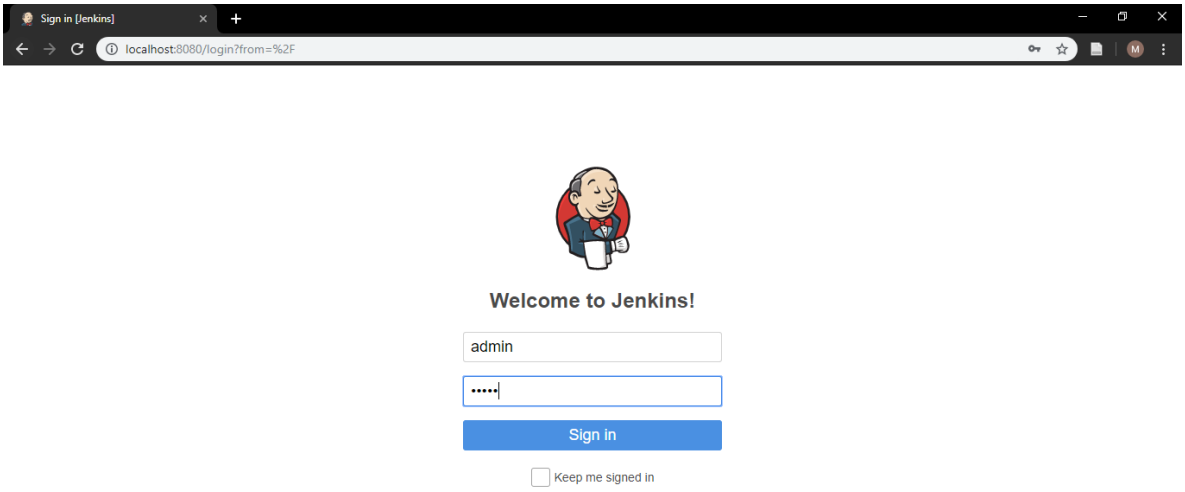
El plugin **EMAIL EXTENSION TEMPLATE** permitirá a Jenkins enviar correos con la información necesaria de la ejecución.

SONARQUBE SCANNER es el plugin encargado la configuración y conexión del servidor SonarQube con Jenkins, para que por medio de SonarQube se logre la revisión de código estático

Adicionalmente a los plugins anteriormente mencionados, se instalará el plugin **SONAR QUALITY GATE** que permite captar el estado de la Quality Gate en Sonarqube después de la revisión iniciada por la ejecución de Jenkins, y según su estado reportar la ejecución como inestable o fallida.

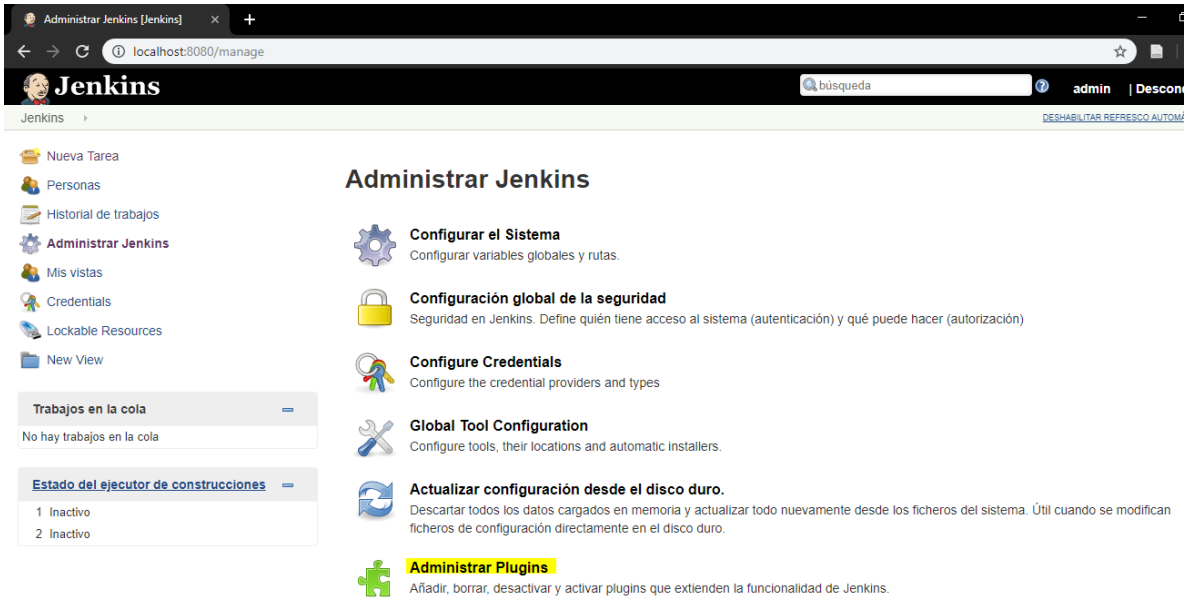
7.1. Ejecute Jenkins y abra en su navegador preferido la dirección anteriormente configurada para ingresar a Jenkins, después ingrese el usuario y contraseña creados para el usuario administrador y realice click sobre el botón Sign in, en el caso práctico se utilizaron los siguientes datos:

- **Dirección**: https://localhost:8080/
- **Usuario**: admin
- **Contraseña**: admin

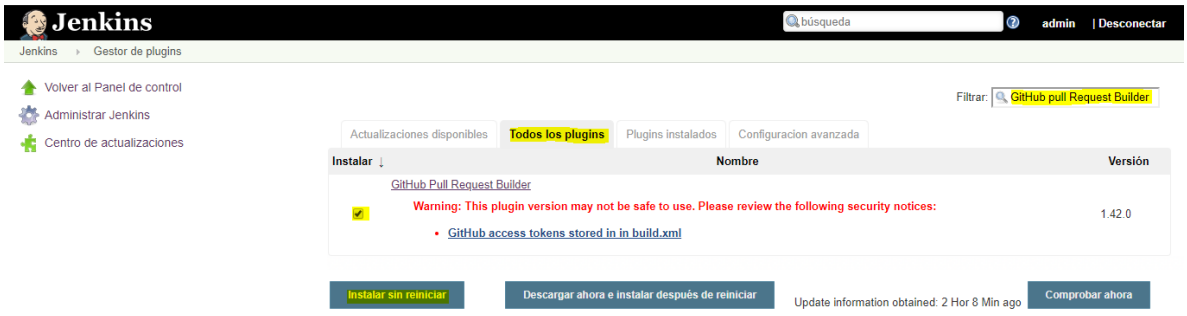


7.2. En el panel principal seleccione la opción Administrar Jenkins

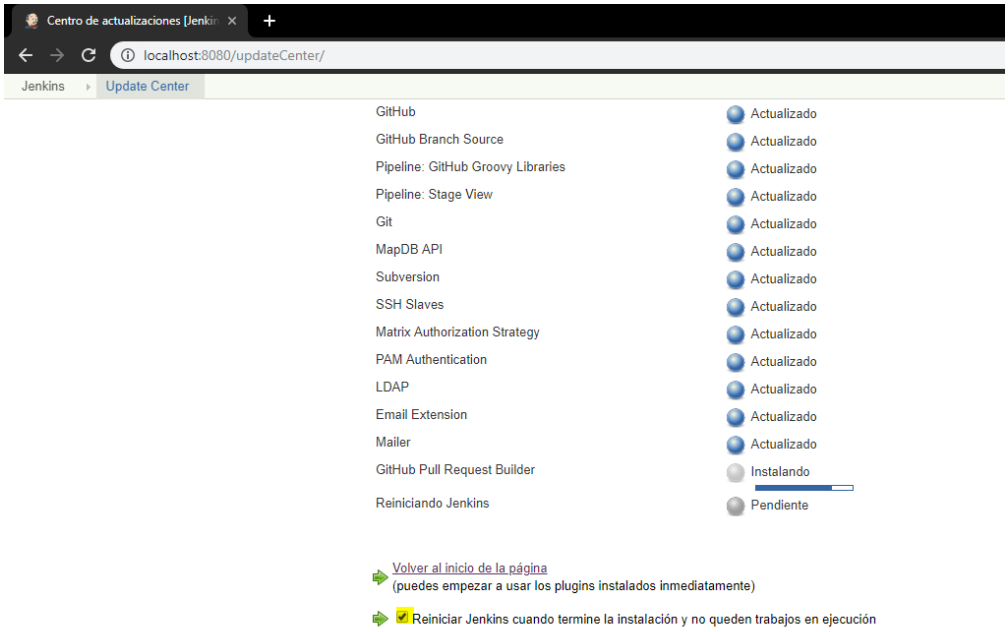
7.3. Seleccione la opción Administrar Plugins



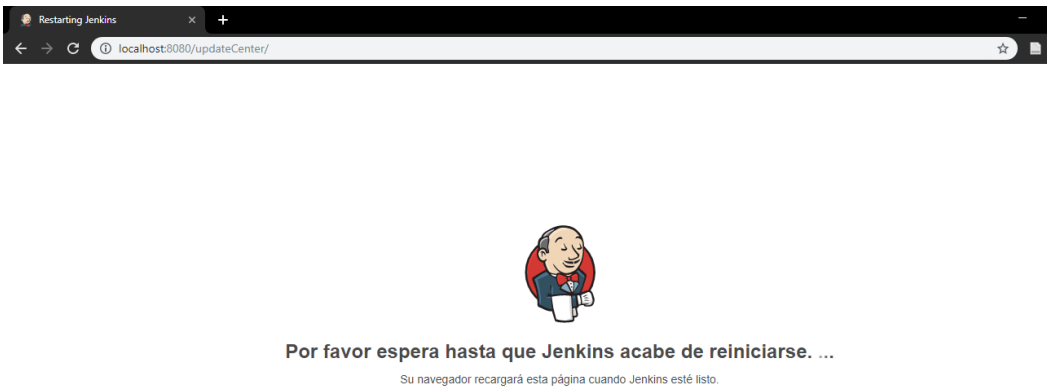
7.4. Seleccione la pestaña Todos los plugins, en el campo Filtrar escriba GitHub pull Request Builder→presione enter →seleccione el check de instalar del plugin encontrado→click en el botón instalar sin reiniciar.



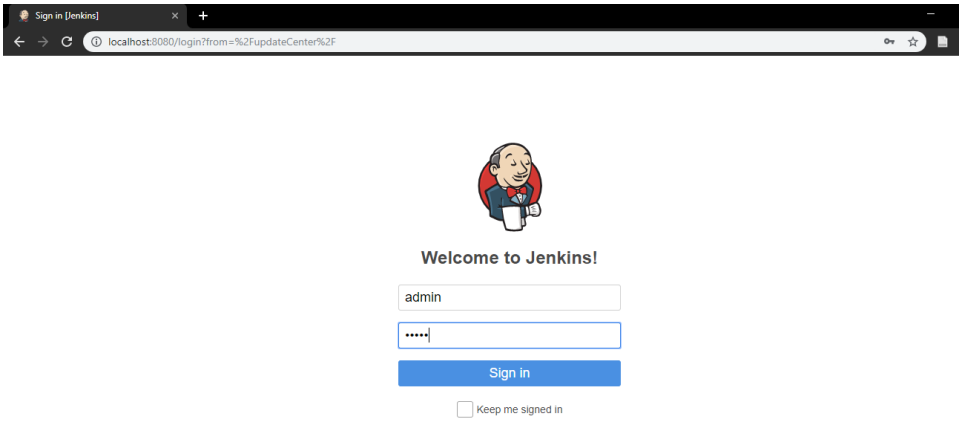
7.5. En la ventana siguiente active el check reiniciar Jenkins cuando termine la instalación y no queden trabajos en ejecución.



7.6. Espere a que Jenkins termine la instalación y se reinicie.

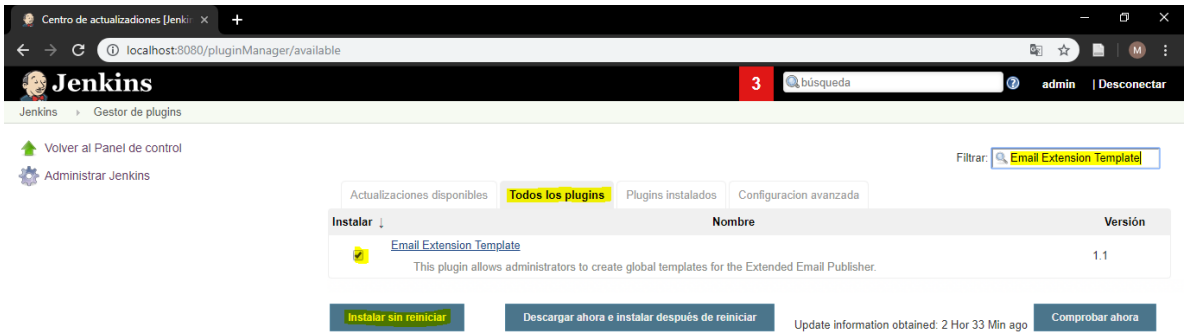


7.7. Ingrese nuevamente a Jenkins.



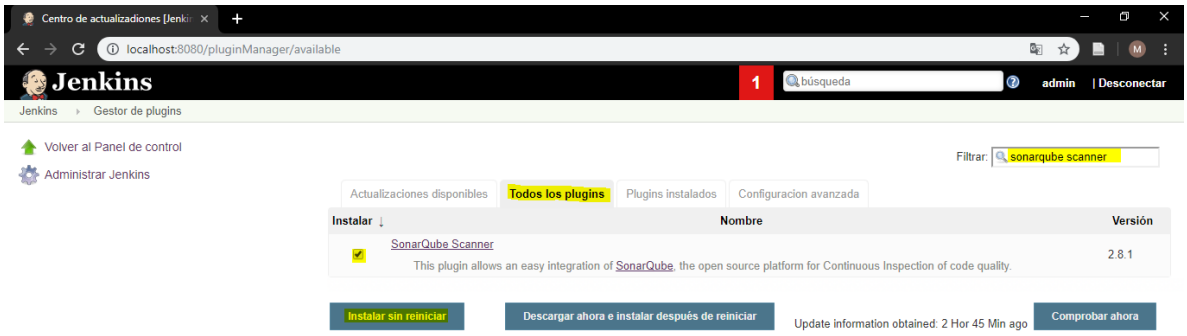
7.8. Seleccione la opción Administrar ‘plugins’.

7.9. Ahora instalará el plugin Email Extension Template el cual es necesario para el uso del plugin Email Extension, para esto seleccione la pestaña Todos los plugins, en el campo Filtrar escriba Email extension Template→presione Enter en su teclado →seleccione el check de instalar del plugin encontrado→click en el botón instalar sin reiniciar.



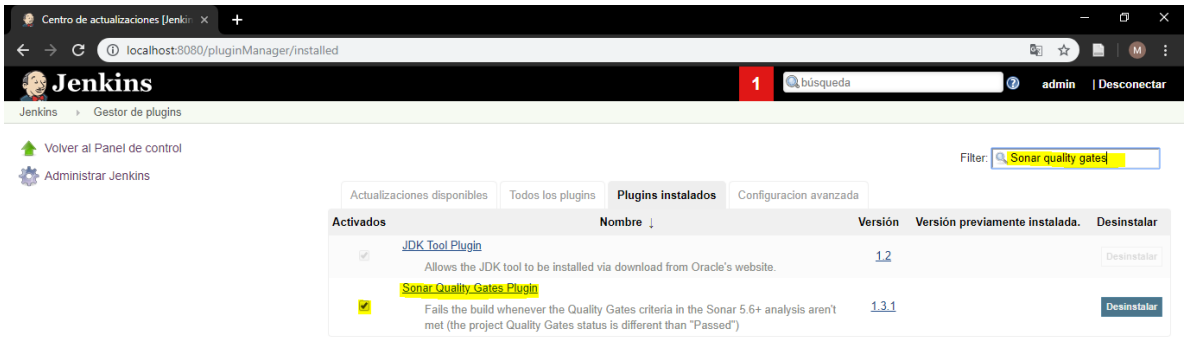
7.10. Repita los pasos 7.5, 7.6, 7.7, 7.8.

7.11. Para instalar el plugin SonarQube Scanner, seleccione la pestaña Todos los plugins, en el campo Filtrar escriba SonarQube Scanner →presione enter →seleccione el check de instalar del plugin encontrado→click en el botón instalar sin reiniciar.



7.12. Repita los pasos 7.5, 7.6, 7.7, 7.8.

7.13. Seleccione la pestaña Todos los plugins, en el campo Filtrar escriba Sonar quality gate→presione enter →seleccione el check de instalar del plugin encontrado→click en el botón instalar sin reiniciar.

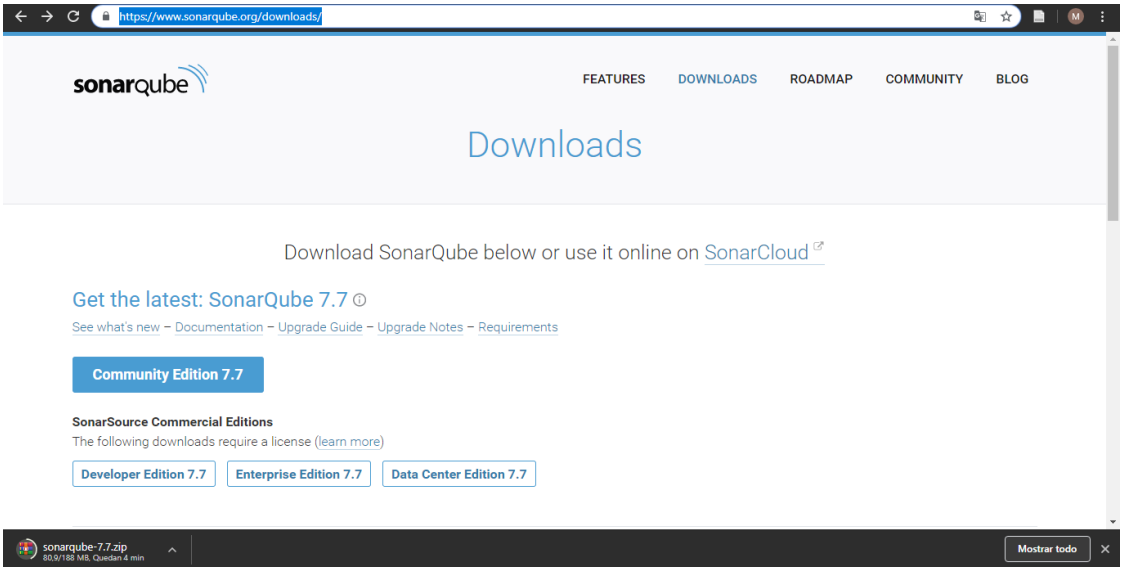


7.14. Finalmente se encuentran instalados los plugins necesario para la configuración del Job en Jenkins.

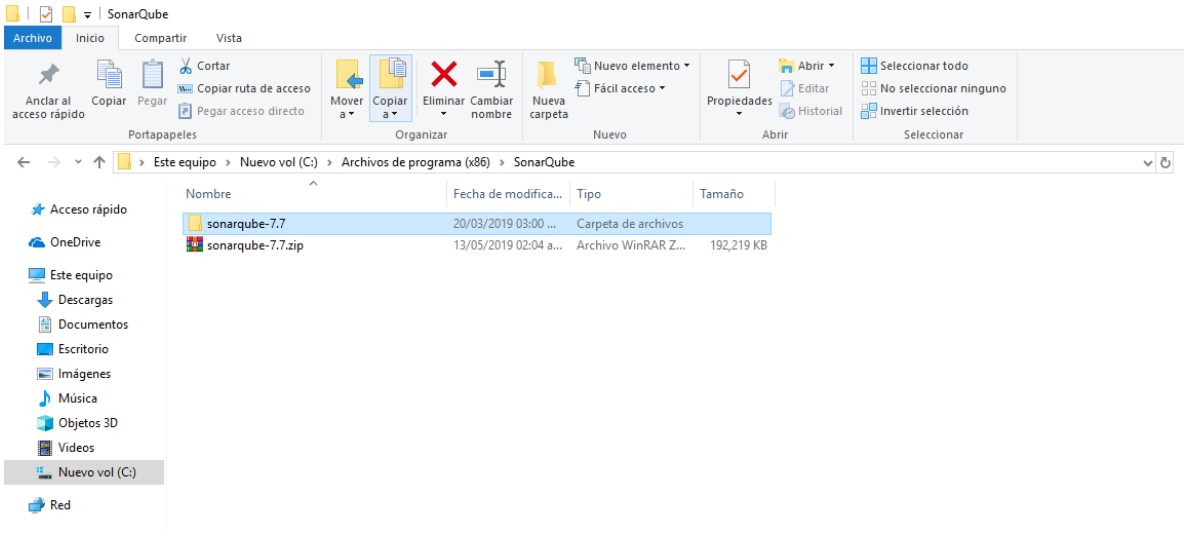
8. INSTALACIÓN DE SONARQUBE

SonarQube es la herramienta para la revisión de código estático del proyecto, reportando malas prácticas de escritura de código y falta de cubrimiento de código con pruebas unitarias, además de otras funcionalidades.

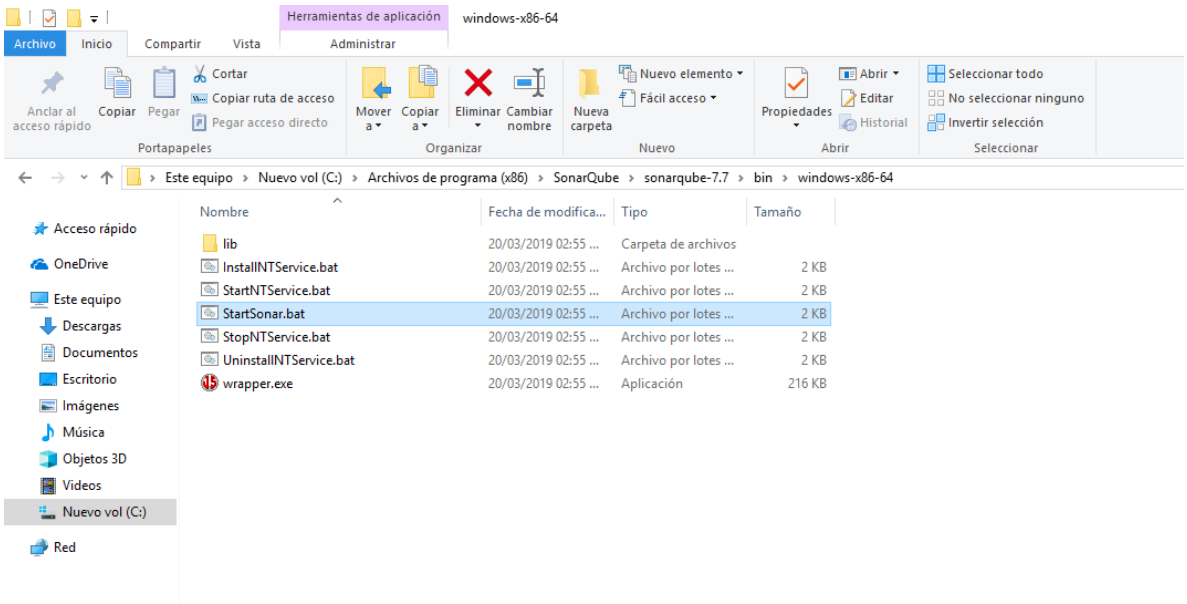
8.1. Ingrese a https://www.sonarqube.org/downloads/ → realice click sobre el botón Community Edition 7.7→ Espere hasta la finalización de la descarga.



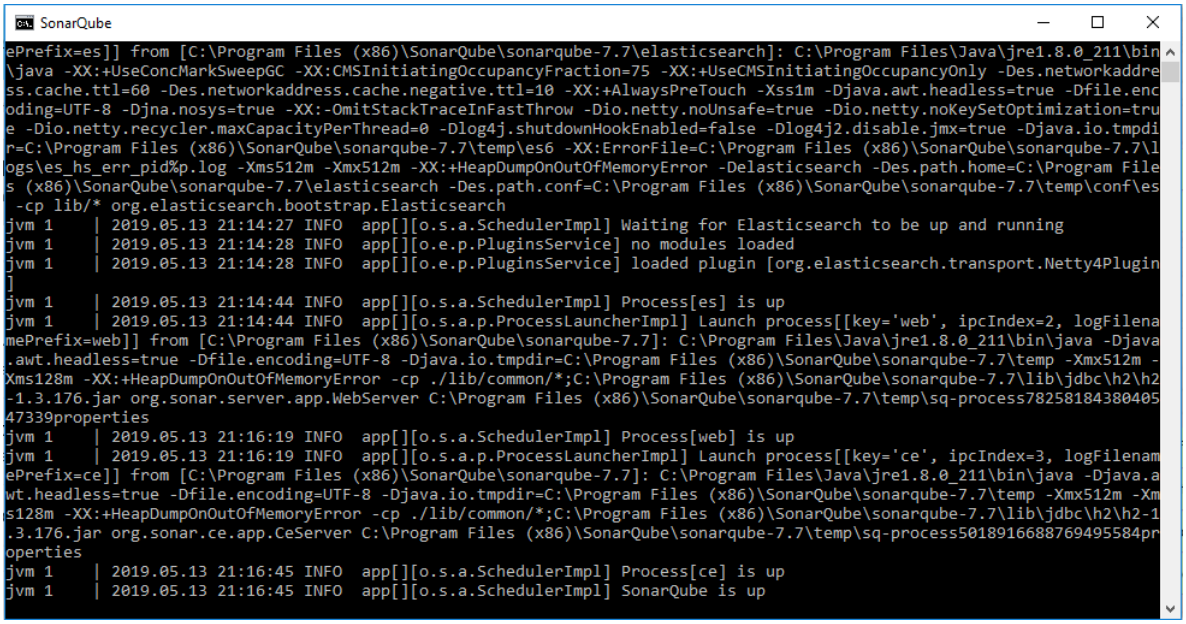
8.2. Descomprima el archivo descargado en una carpeta, de preferencia en la unidad C:, en este caso se descomprimió en el directorio: C:\Program Files (x86)\SonarQube



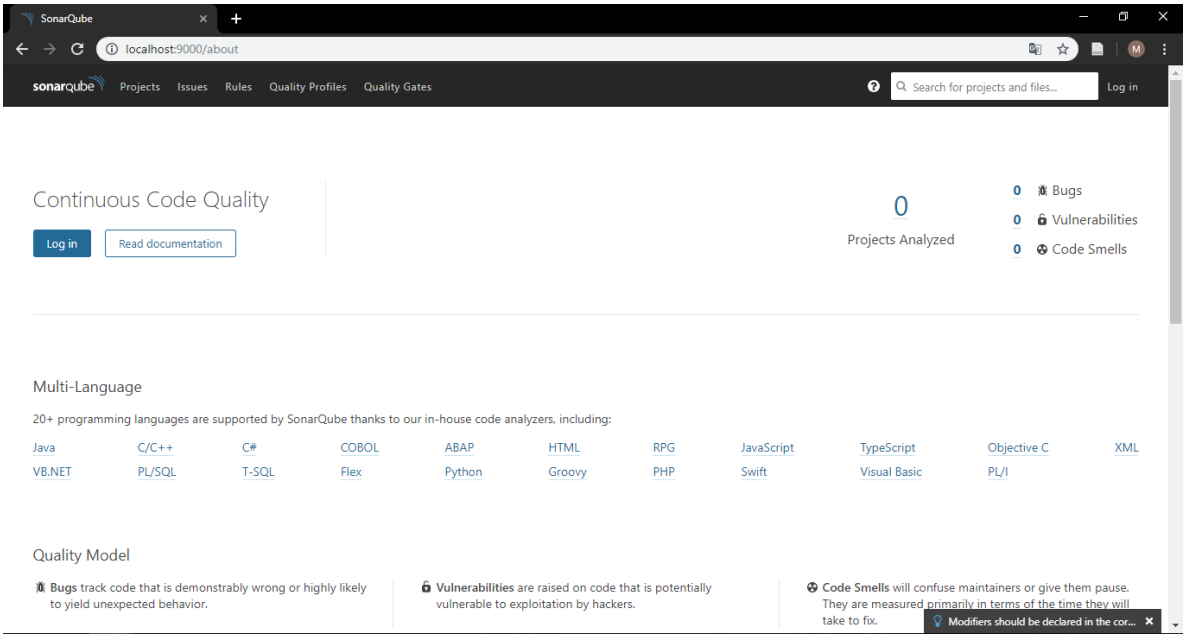
8.3. Ingrese a la carpeta descomprimida→ sonarqube-7.7 → bin → Windows-x86-64 y ejecute como administrador el archivo StartSonar.bat



8.4. Espere hasta el completo inicio de SonarQube



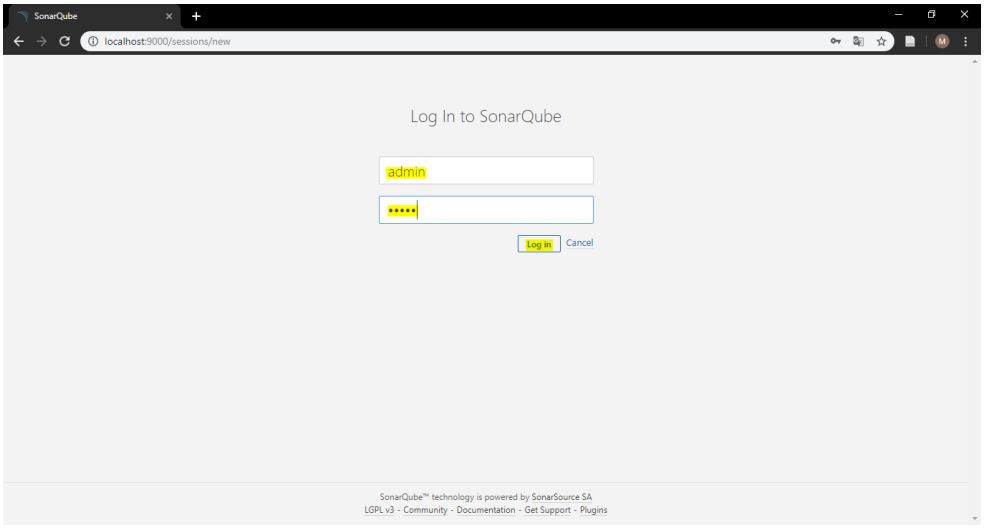
8.5. Por default la primera vez que se inicia sonarQube, la Url para acceder al sistema es http://localhost:9000 por lo cual copie y pegue esta dirección en su navegador preferido.



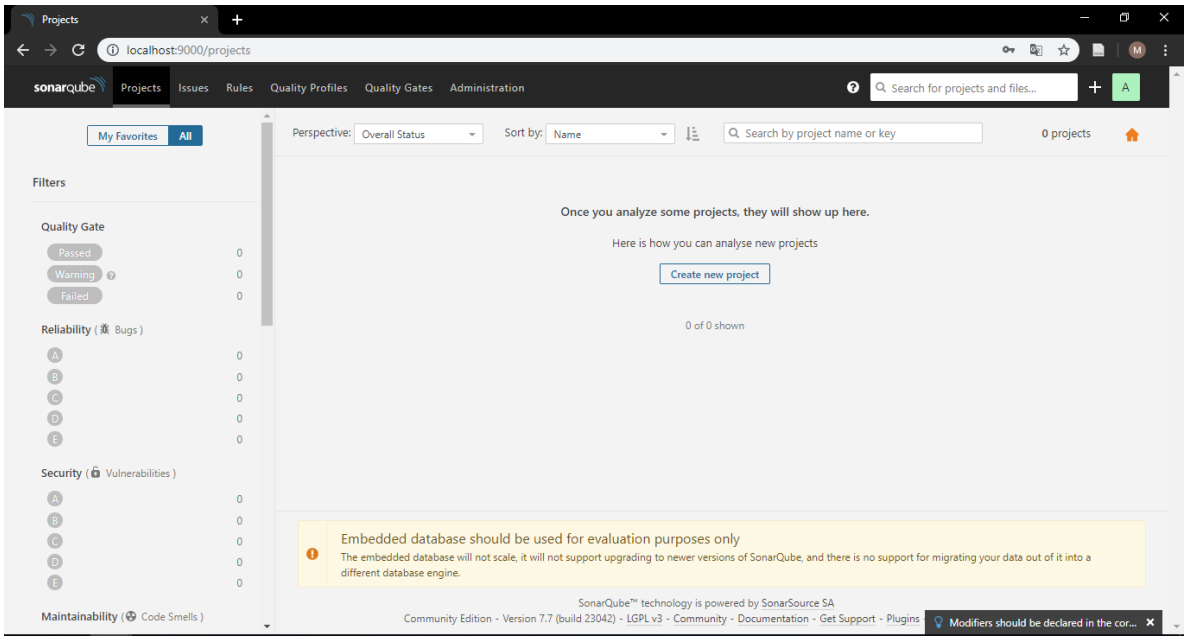
8.6. Realice click sobre el botón Log in para loguearse en el sistema, los datos por default por primera vez son:

- **Username** : admin
- **Password**: admin

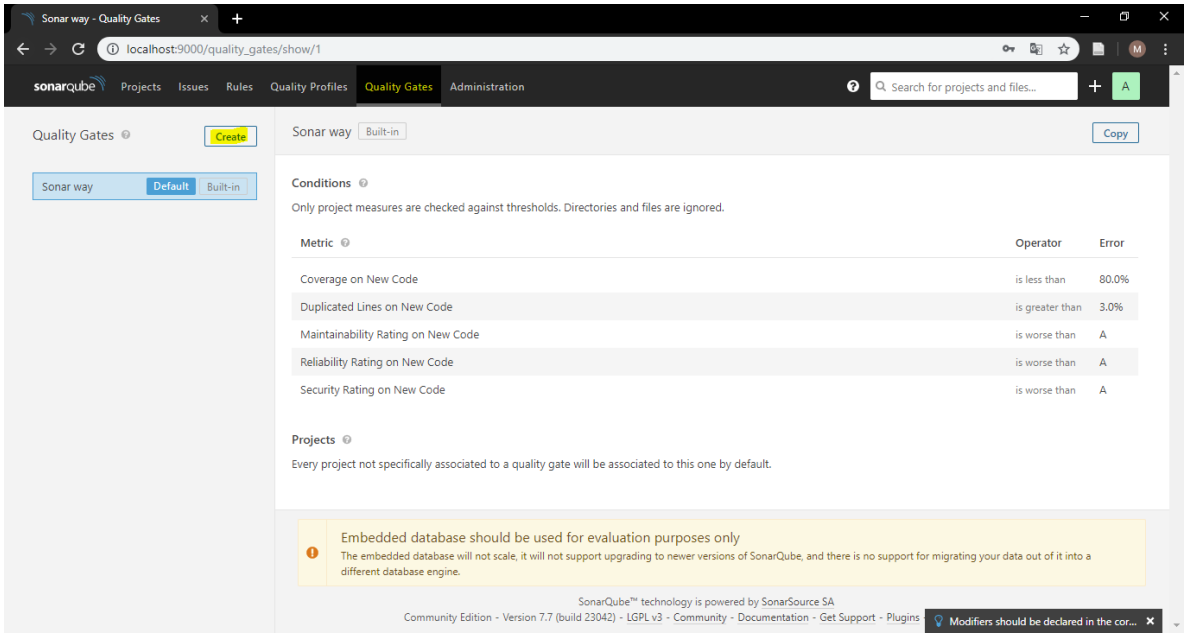
Finalmente oprima el botón Log in



8.7. SonarQube mostrará una ventana similar a la siguiente.

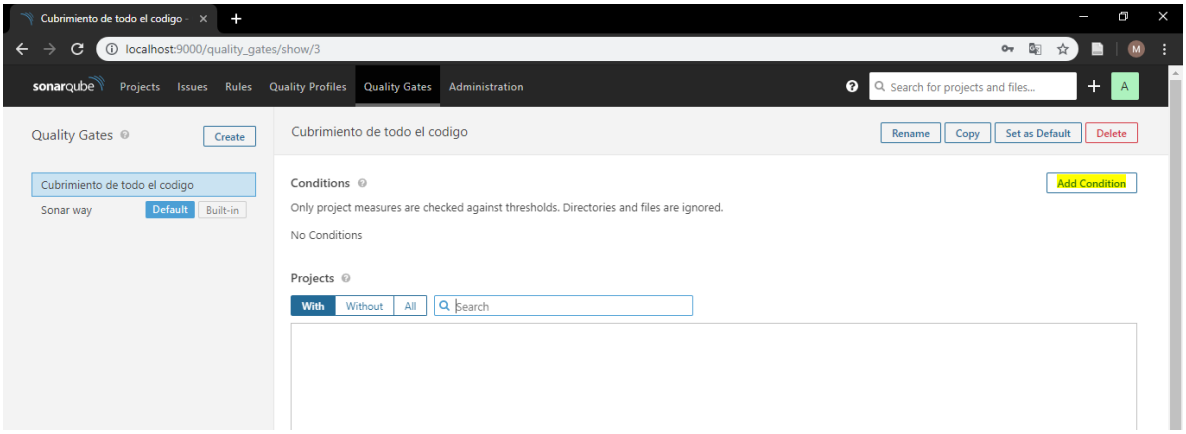


8.8. Por motivos del ejercicio práctico se modificará una Quality gate, la cual es una serie de reglas que indicarán si después de la revisión de código estático, este supero o no supero la Quality gate, para lo cual realice click sobre el botón del menú superior Quality Gates→ create



8.9. Nombre la regla a crear, en este caso se nombrará “Cubrimiento de todo el código”, después realice click en el botón Save

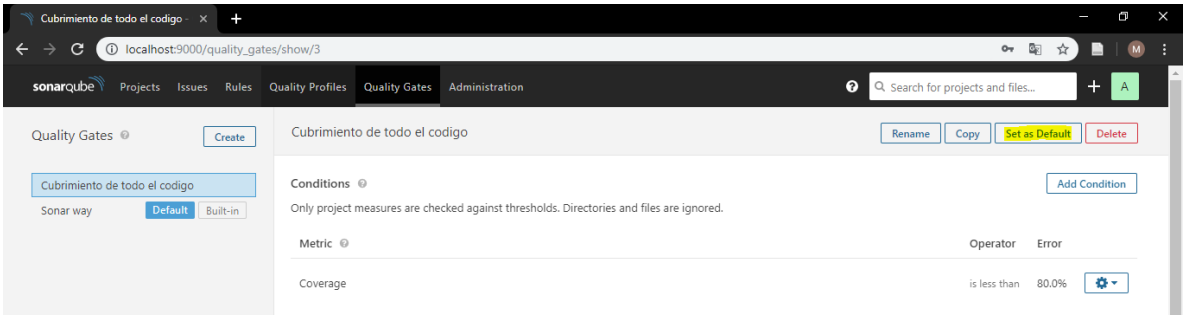
8.10. Oprima el botón Add condition, para agregar la condición o regla de esta Quality Gate



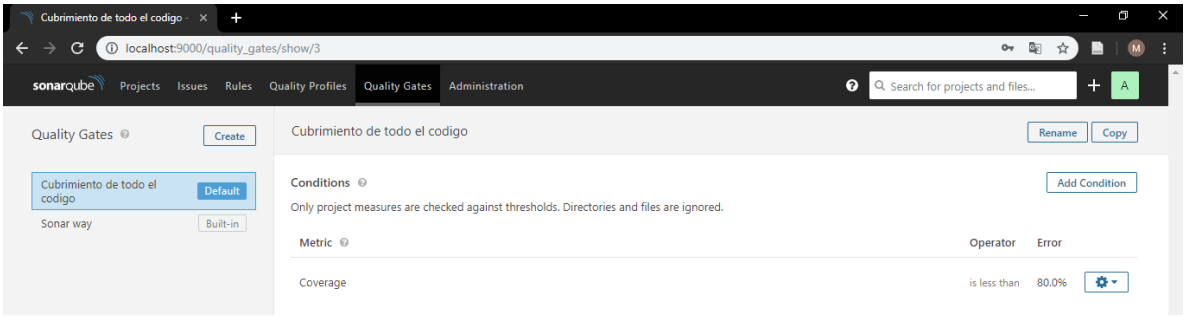
8.11. En la ventana emergente, seleccione de la lista Metric la opción Coverage, en el campo Error , agregue el valor de 80 y finalmente oprima el botón Add Condition.

Esta regla indica que, si el cubrimiento de pruebas unitarias sobre el total de código analizado es menor al 90%, no se cumple con la Quality Gate.

8.12. Seleccione el botón Set as Default , para indicar que esta Quality Gate se aplicará como default a todos los proyectos nuevos que sean escaneados por SonarQube



8.13. Finalmente se mostrará una ventana similar a la siguiente.



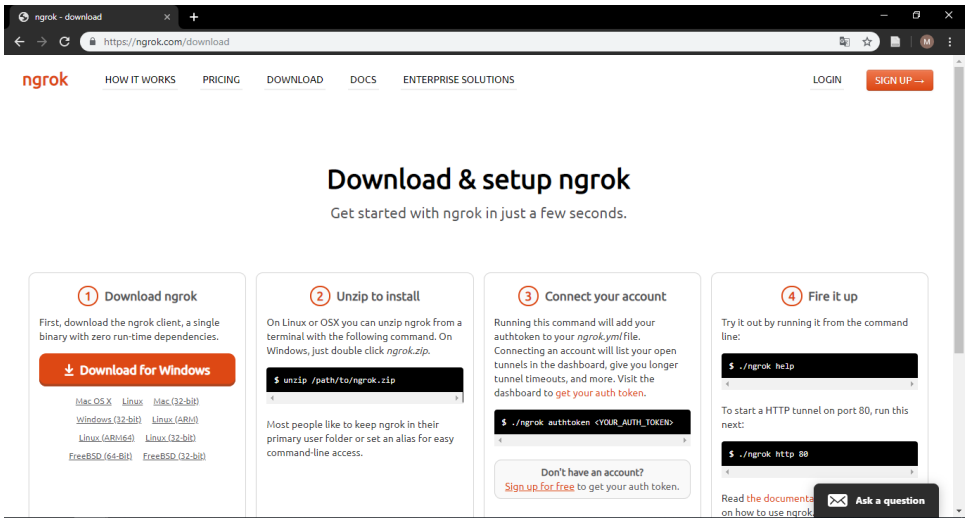
9. INSTALACIÓN DE NGROK

El software Ngrok tiene la función de exponer al internet los servidores locales Jenkins y Sonarqube, crea una Url hacia internet a las Url de servicios locales, por ejemplo:

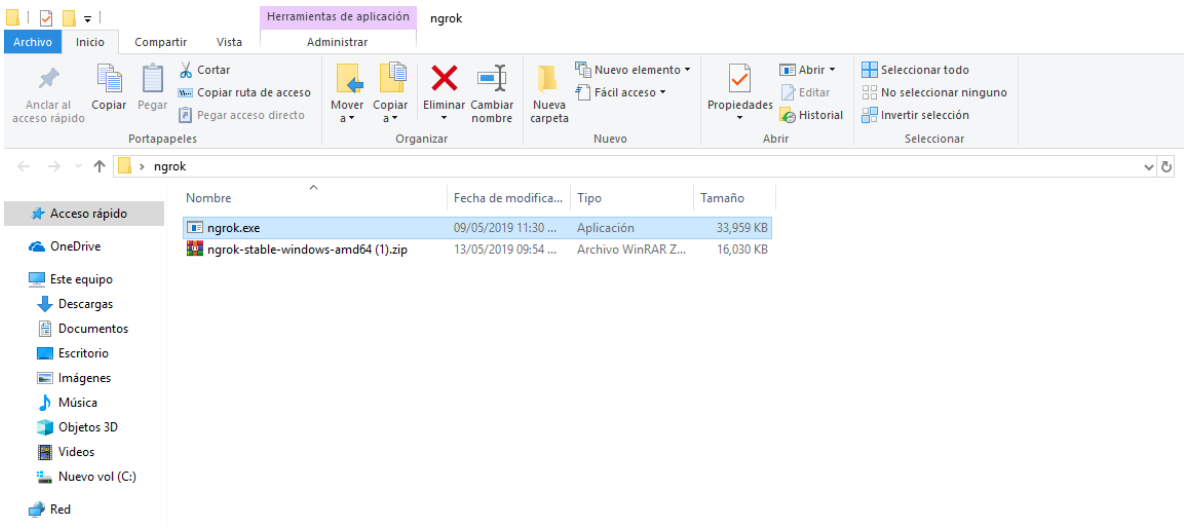
Url de Jenkins local:
http://localhost:8080 (Servidor local accesible solo desde la propia maquina)
Después de usar ngrok:
http://f15524bf.ngrok.io (Servidor local accesible desde internet)

Se utiliza esta herramienta para conectar exitosamente los webhooks de GitHub con Jenkins, GitHub no puede establecer conexión contra una Url como http://localhost:8080, porque esta es local y necesita una dirección o ip expuesta al internet para realizar la comunicación.
Es importante tener en cuenta que estas direcciones son temporales y tienen una duración de máximo 8 horas.

9.1. Ingrese a la página oficial de ngrok https://ngrok.com/download y oprima el botón Download for Windows para iniciar la descarga del software.



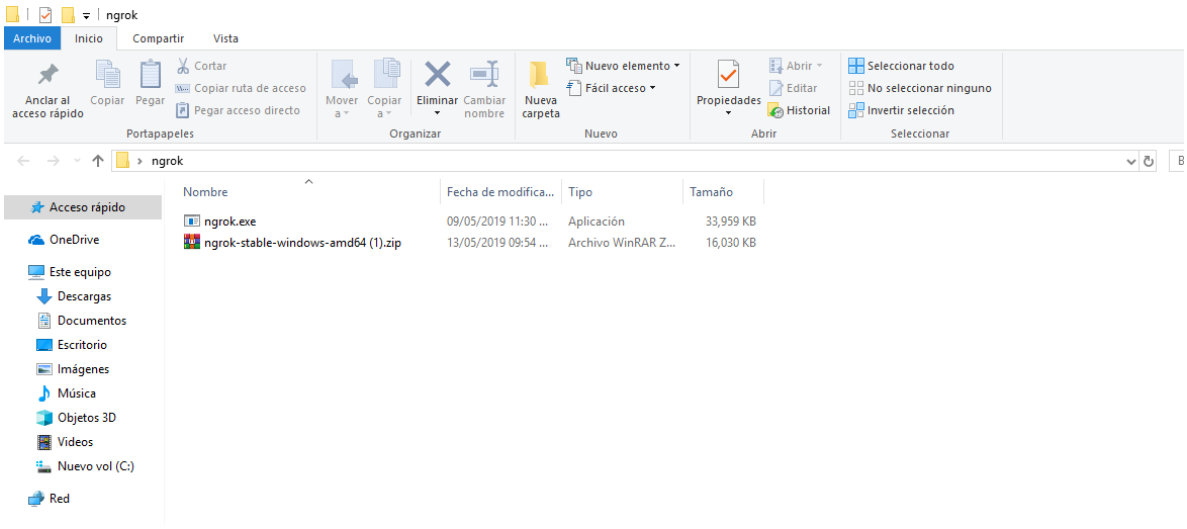
9.2. Después de finalizar la descarga, descomprima el archivo.



10. CONFIGURACIÓN DE GITHUB WEBHOOK PULL REQUEST

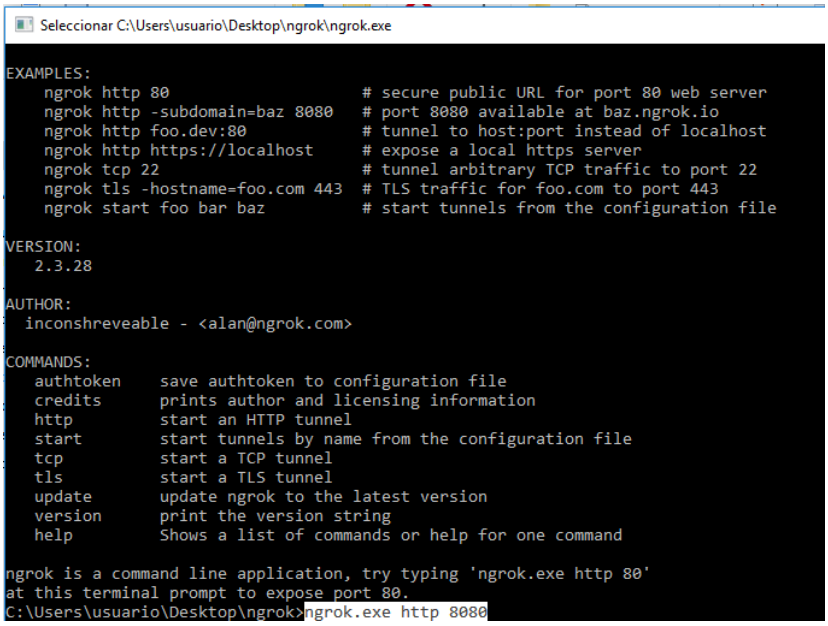
La configuración del WebHook pull request permite a github enviar una petición de ejecución a Jenkins cada vez que se realice un pull request y recibir la información resultante de la ejecución de Jenkins.

10.1. Primero se expone la Url local http://localhost:8080 de Jenkins al internet utilizando el software ngrok, para lo cual realice doble click sobre el software ngrok.exe



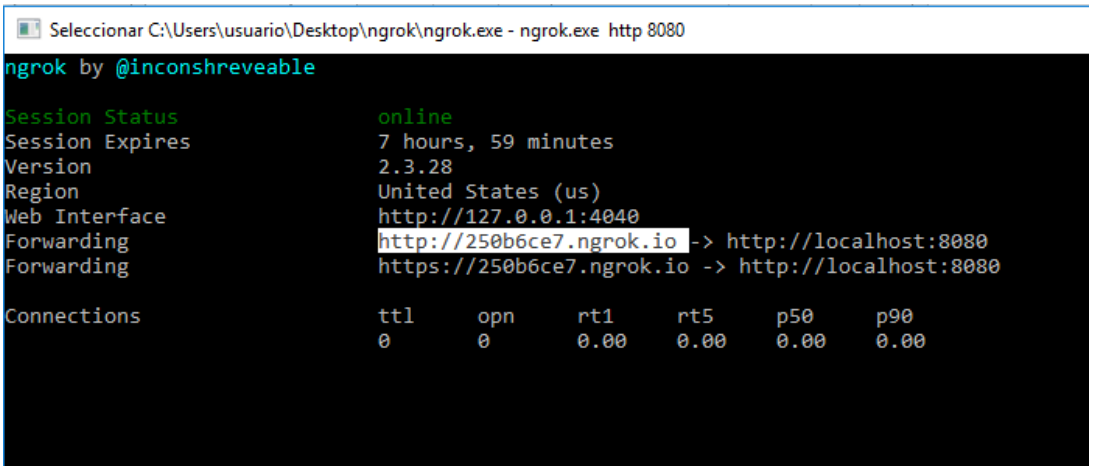
10.2. En la ventana que muestra el sofware ngrok, ingrese la siguiente instrucción: ngrok.exe http 8080

Esta instrucción solicita la creación de un túnel para exponer el servicio del puerto 8080, el cual pertenece a http://localhost:8080 de Jenkins. Luego de ingresar la instrucción oprima el botón Enter en su teclado.



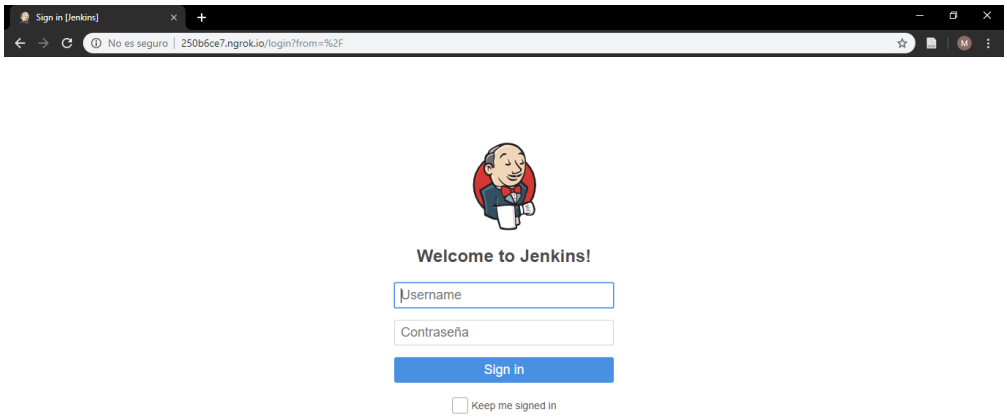
10.3. El sistema mostrará la Url con al cual puede acceder a Jenkins desde internet, siendo en este caso la dirección http://250b6ce7.ngrok.io o https://250b6ce7.ngrok.io

.



10.4. Copie la Url resultante y péguela en su navegador de preferencia para verificar si puede acceder a Jenkins.

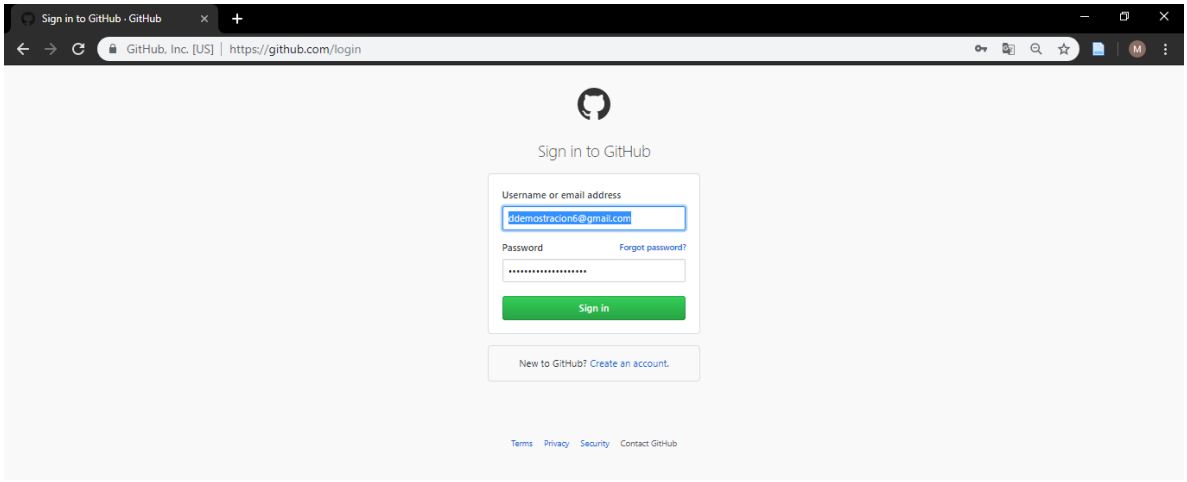
En este caso se utilizó la Url: http://250b6ce7.ngrok.io



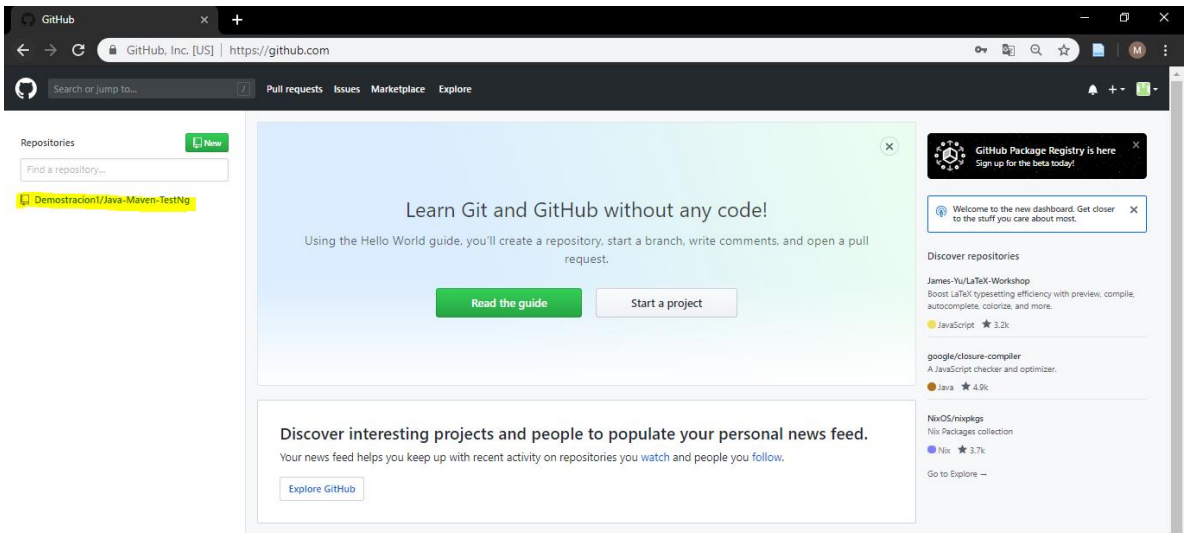
10.5. Ingrese a Github.com con la información de la cuenta de GitHub creada anteriormente en la cual se encuentra versionado el código del proyecto java, en este caso práctico se utilizaron los datos:

- **Username or email address**: ddemostracion6@gmail.com
- **Password**: Passwordparamuestra1

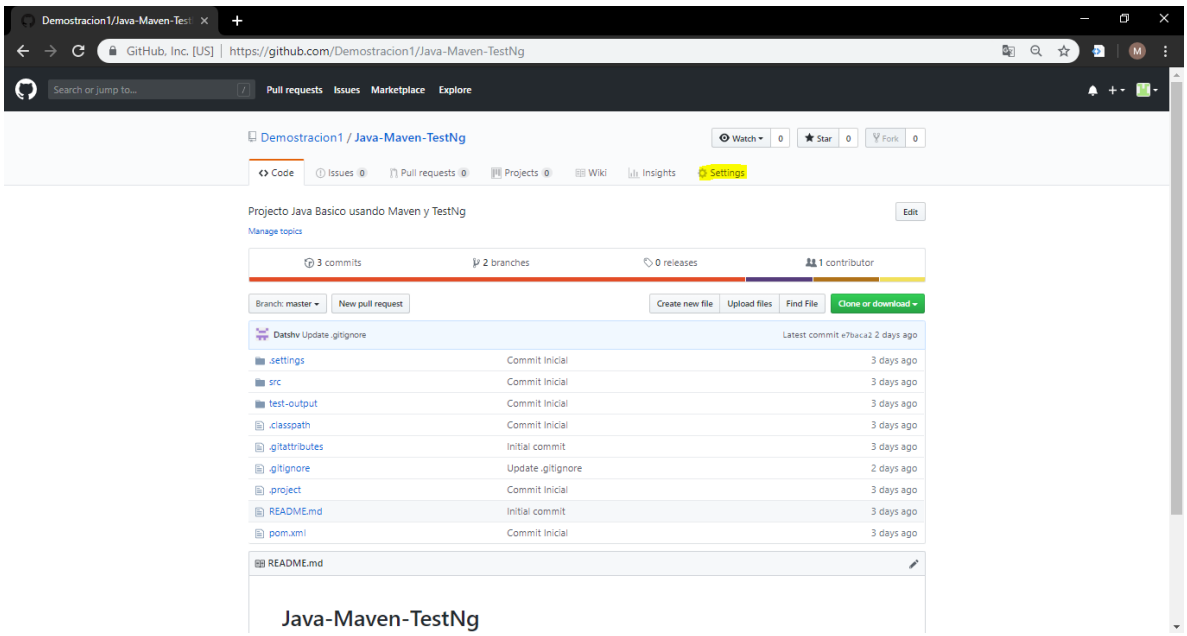
Después de ingresar los datos solicitados, presione el botón Sign in.



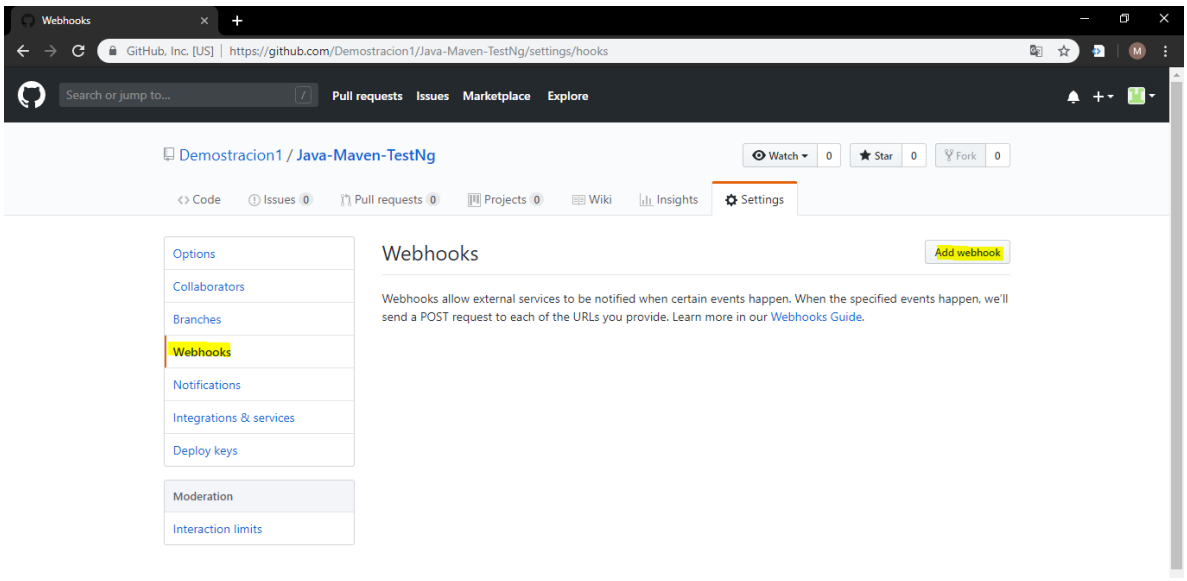
10.6. Realice click sobre el repositorio que contiene el proyecto versionado.



10.7. En el panel principal del repositorio realice click sobre la opción settings.



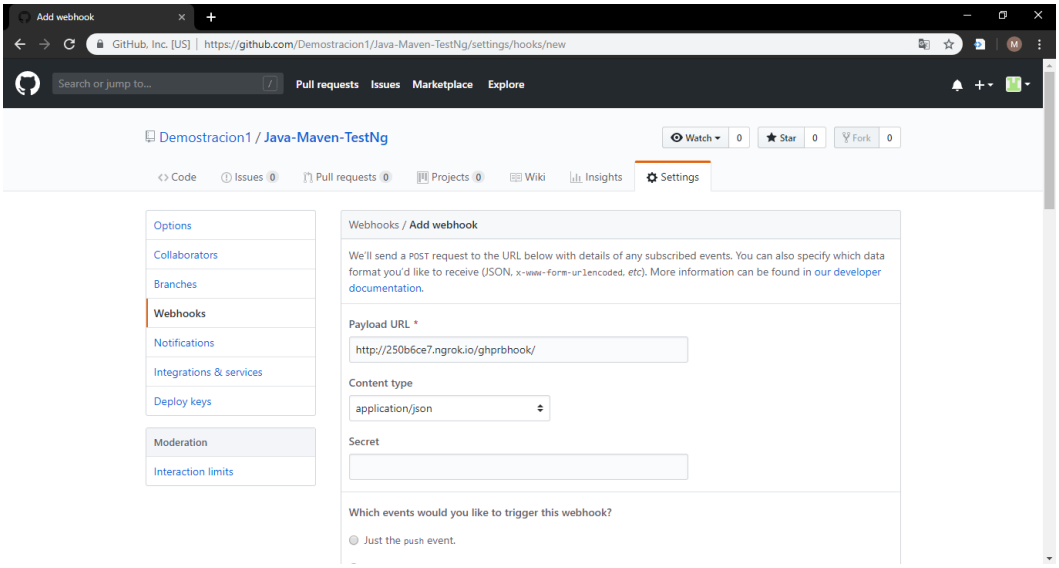
10.8. Seleccione la opción Webhooks → Add webhook



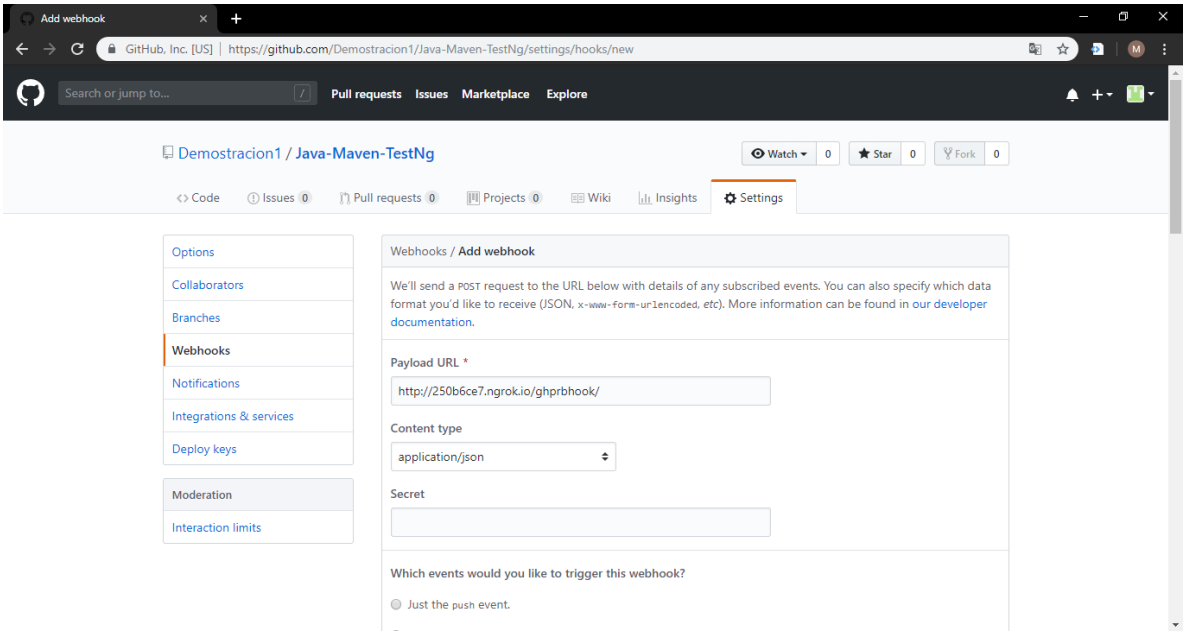
10.9. En el campo Payload Url : agregue la Url de Jenkins que proveyó ngrok.exe, en este caso práctico fue http://250b6ce7.ngrok.io , además a esta Url agregue /ghprbhook/ al final , resultando en la Url :

http://250b6ce7.ngrok.io/ghprbhook/

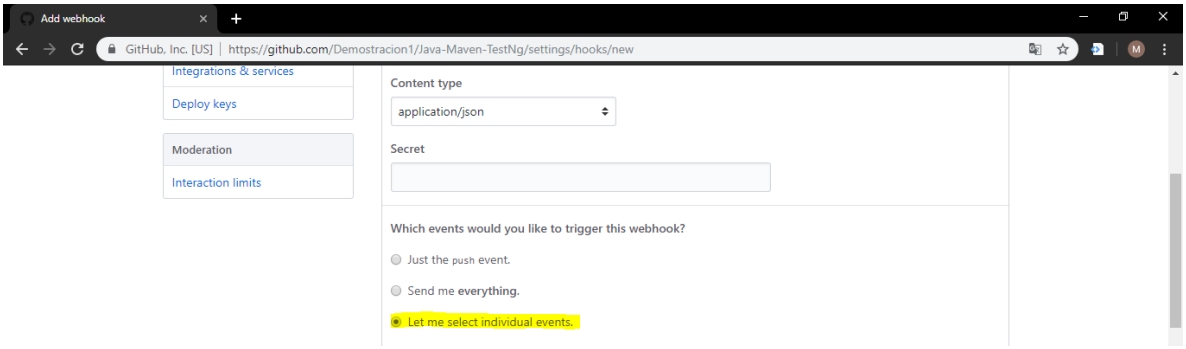
Recuerde hacer uso de la Url que ngrok.exe dio a Jenkins.



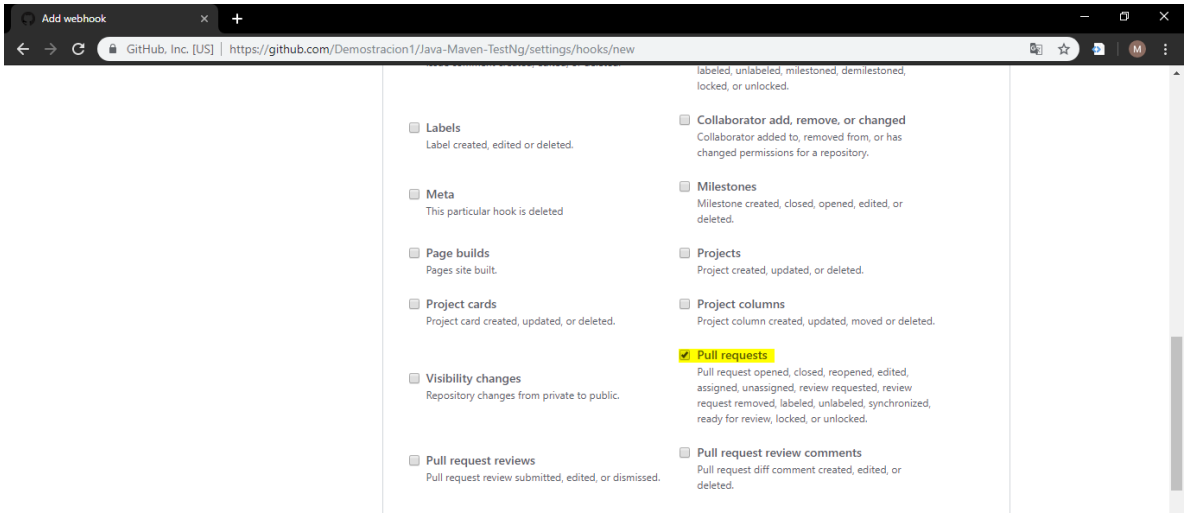
10.10. En la lista desplegable del campo Content type, seleccione la opción application/json



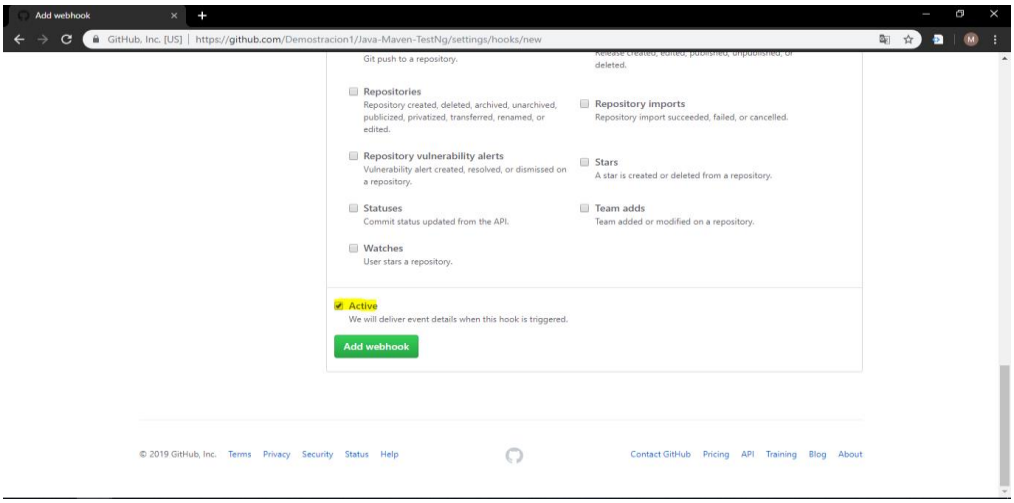
10.11. En la opción, Which events would you like to trigger this webhook?, seleccione Let me select individual events.



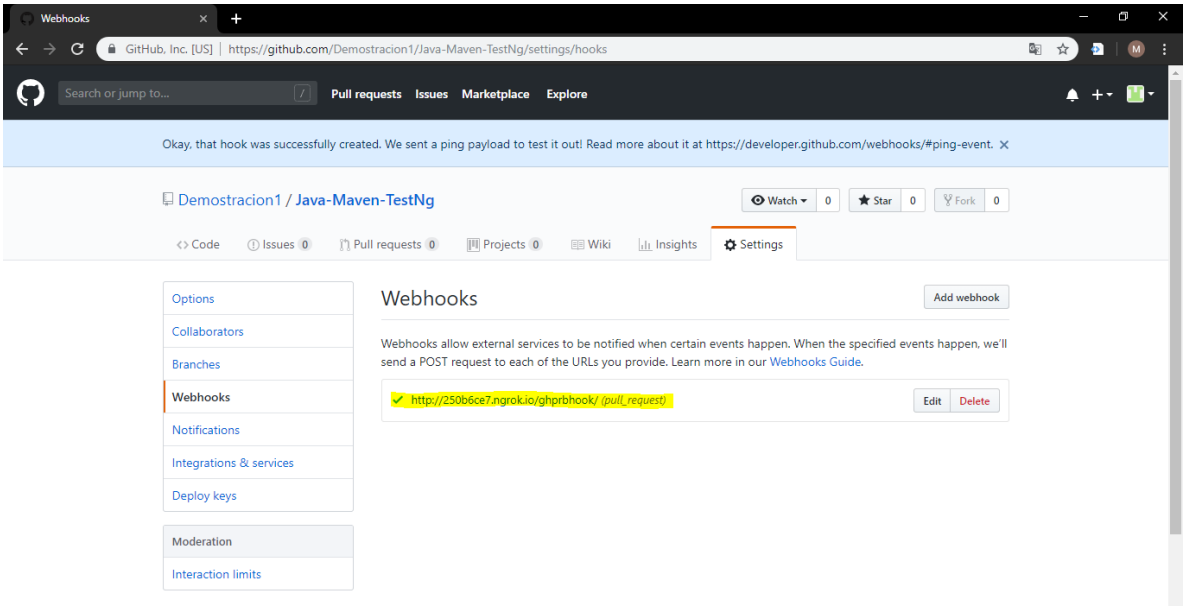
10.12. En la lista de eventos, marque la opción Pull Request y desmarque todas las demás.



10.13. Marque la casilla Active y oprima el botón Add webhook.



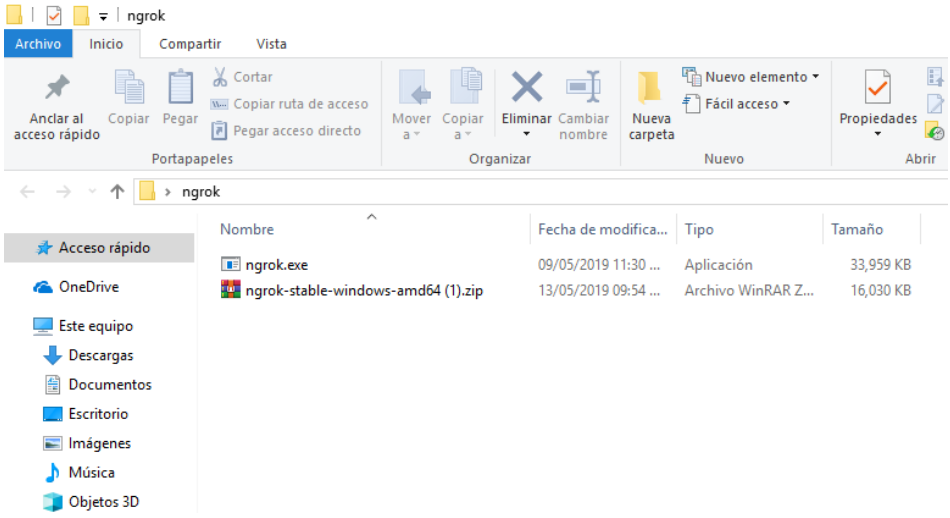
10.14. Al finalizar la configuración del Webhook, verifique que este tenga el visto bueno al lado de la Url del webhook.



11. EXPONER SONARQUBE A INTERNET

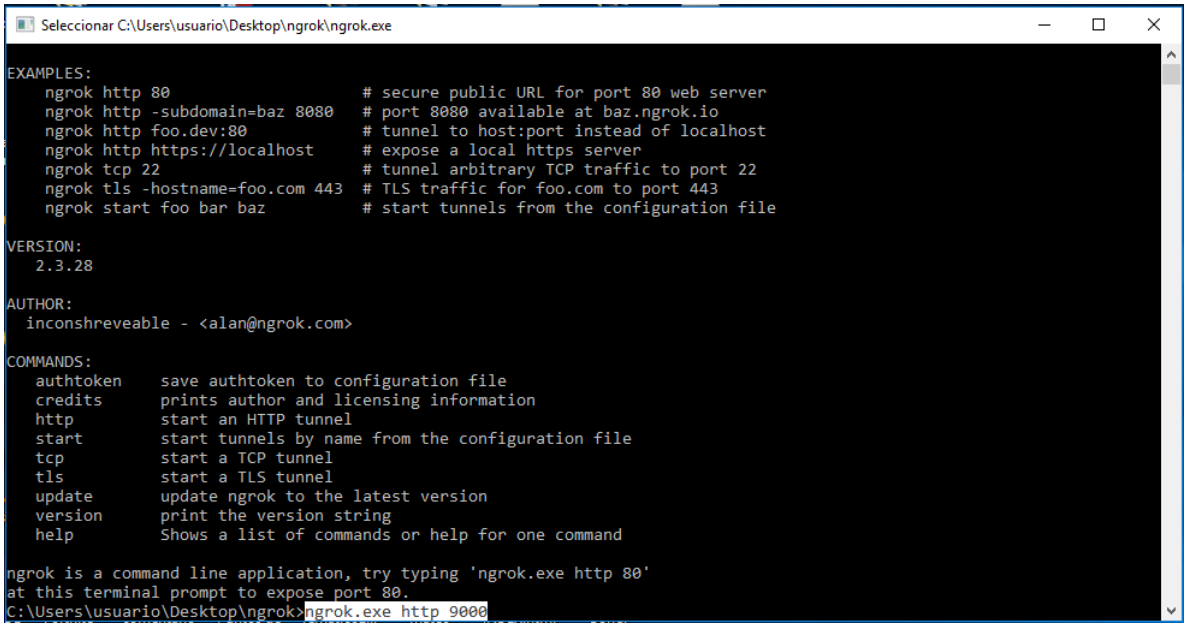
Con el uso del software ngrok.exe se crear un túnel para exponer al internet a Sonarqube.

11.1. Realice doble click sobre el software ngrok.exe

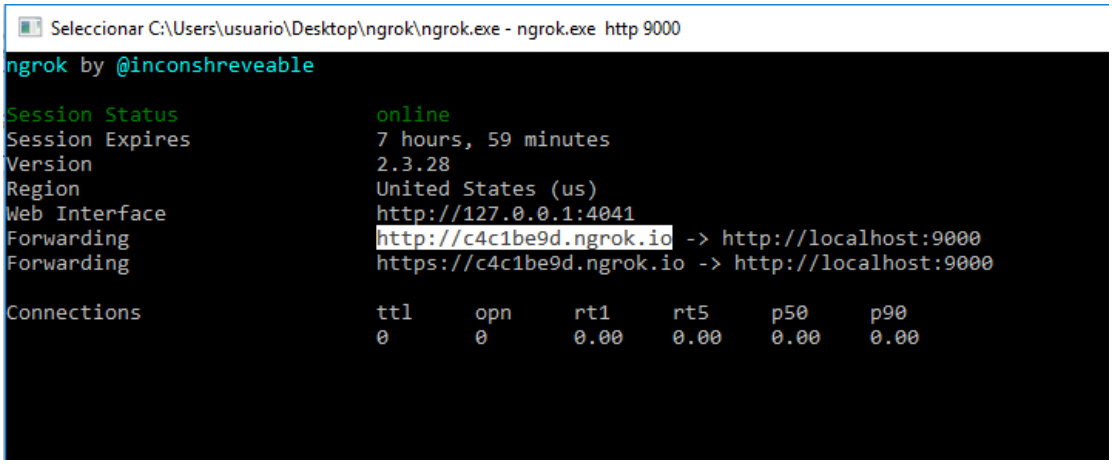


11.2. En la ventana que muestra el sofware ngrok, ingrese la siguiente instrucción: ngrok.exe http 9000

Esta instrucción solicita la creación de un túnel para exponer el servicio del puerto 9000, el cual pertenece a http://localhost:9000 de SonarQube. Luego de ingresar la instrucción oprima el botón Enter en su teclado.

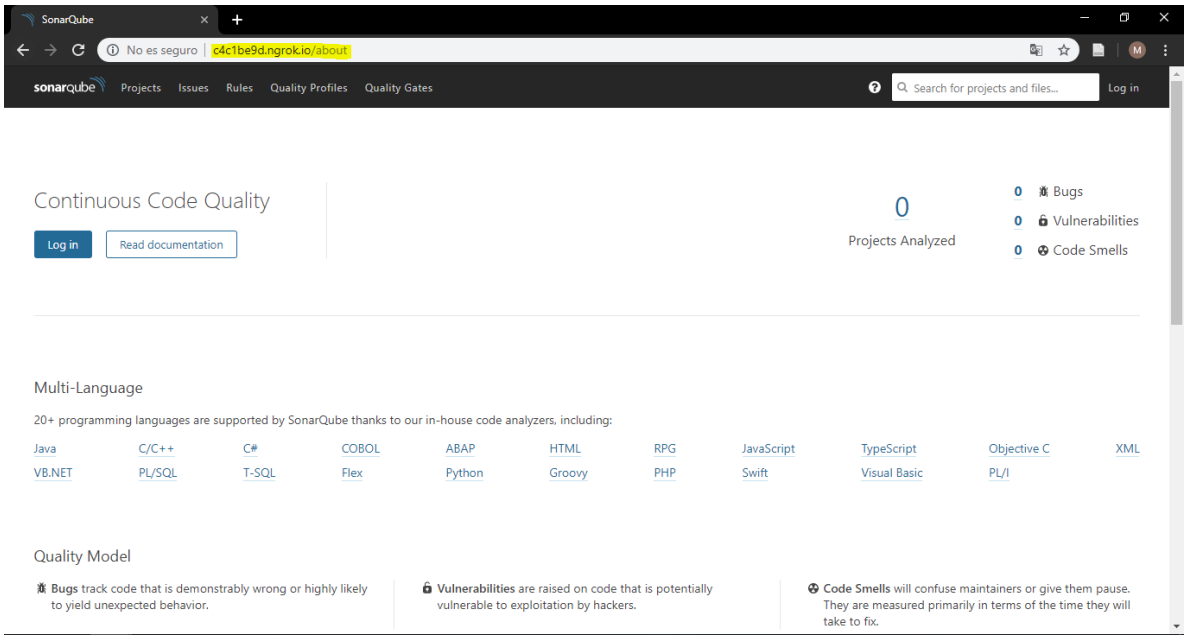


11.3. El sistema mostrará la Url con al cual puede acceder a SonarQube desde internet, siendo en este caso la dirección http://c4c1be9d.ngrok.io o https://c4c1be9d.ngrok.io



11.4. Copie la Url resultante y péguela en su navegador de preferencia para verificar si puede acceder a su SonarQube.

En este caso se utilizó la Url: http://c4c1be9d.ngrok.io

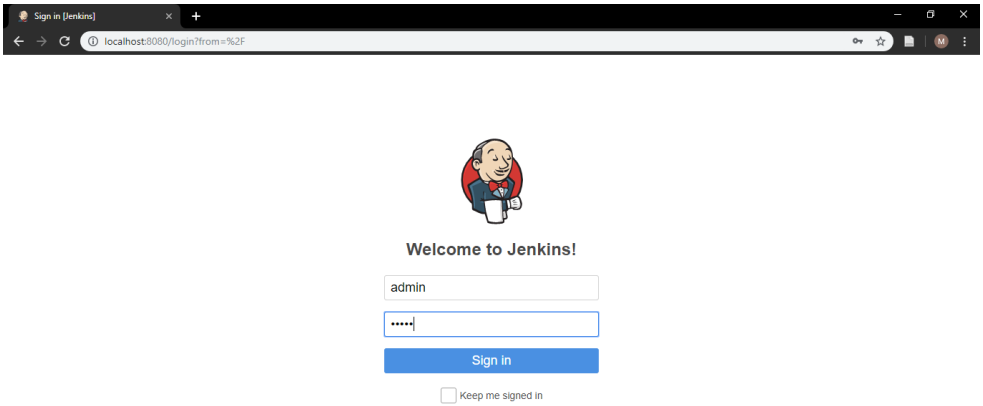


12. CONFIGURACIÓN DE PLUGINS EN JENKINS

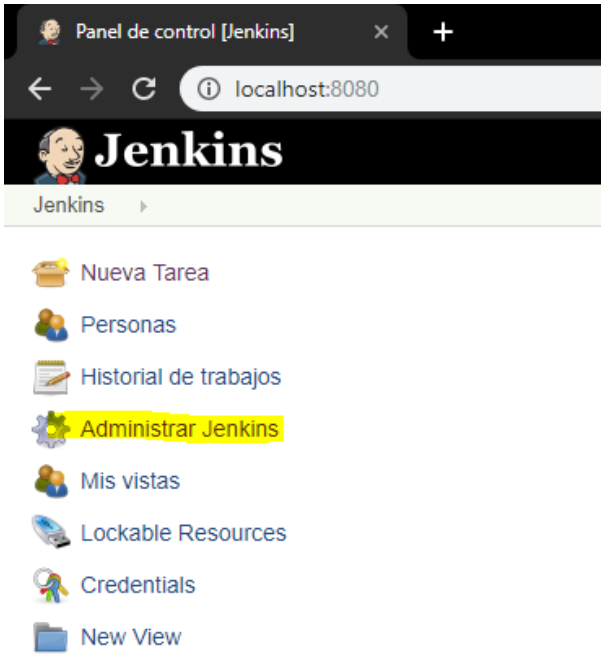
En esta sección se configurarán los plugins EMAIL EXTENSION TEMPLATE, SONARQUBE SCANNER, SONAR QUALITY GATE y GITHUB PULL REQUEST los cuales fueron instalados anteriormente en Jenkins.

12.1. Ingrese al servidor Jenkins, en este caso práctico se utilizará la Url: http://localhost:8080 , además del usuario y contraseña creados anteriormente durante la instalación de Jenkins.

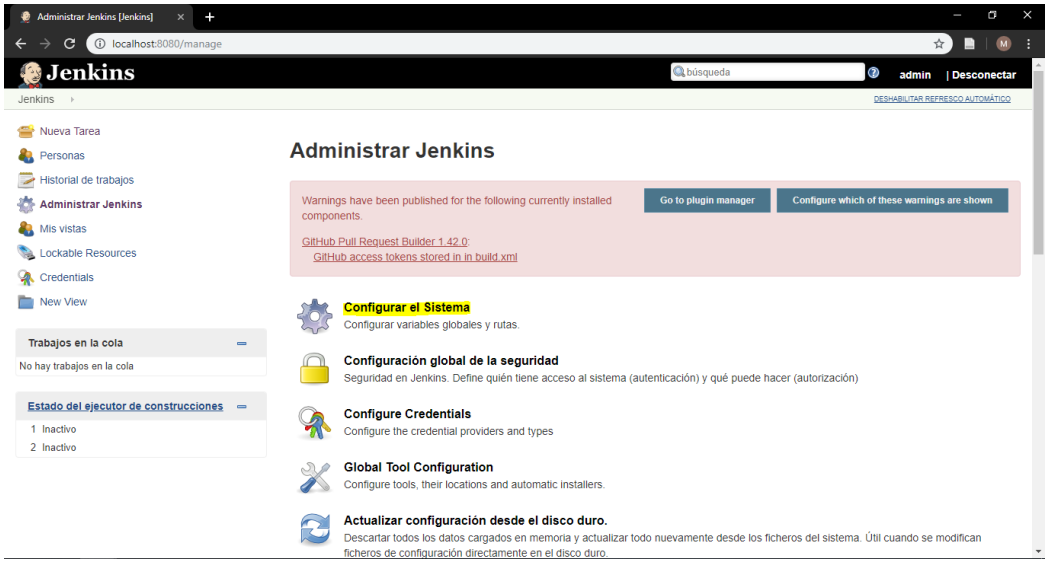
- **Username**: admin
- **Password** : admin



12.2. En el panel principal de Jenkins, realice click sobre la opción Administrar Jenkins



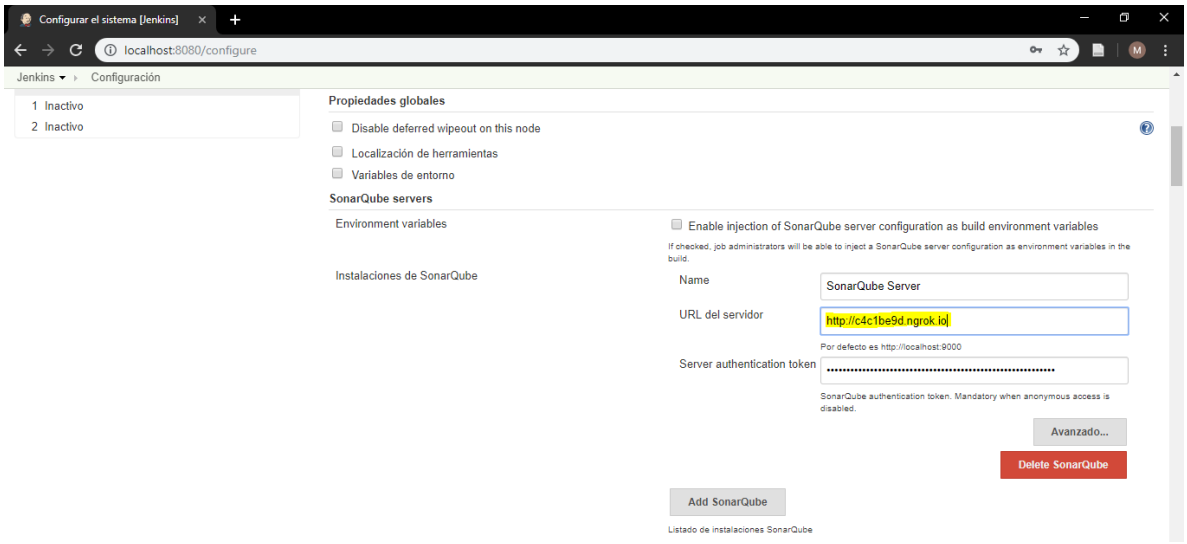
12.3. En la siguiente ventana seleccione la opción Configurar el Sistema.



12.4. Busque la sección de configuración de SonarQube servers e ingrese solo la siguiente información:

- **Name**: nombre del servidor, puede ser cualquier nombre (en este caso se nombró SonarQube Server)
- **URL del servidor:** Agregue la Url que ngrok.exe dio a SonarQube(en este caso se utilizó http://c4c1be9d.ngrok.io)

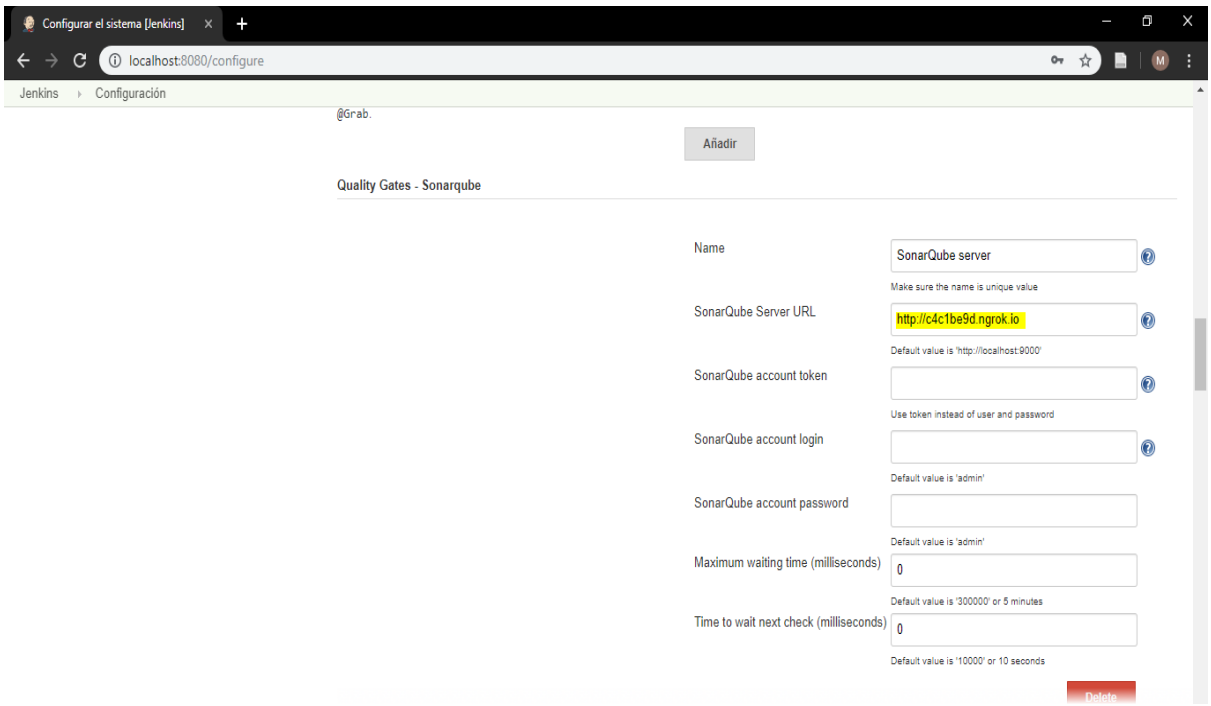
NO agregue información en el campo Server authentication token, este se llenará automáticamente.



12.5. Busque la sección de configuración de Quality Gate - SonarQube e ingrese solo la siguiente información:

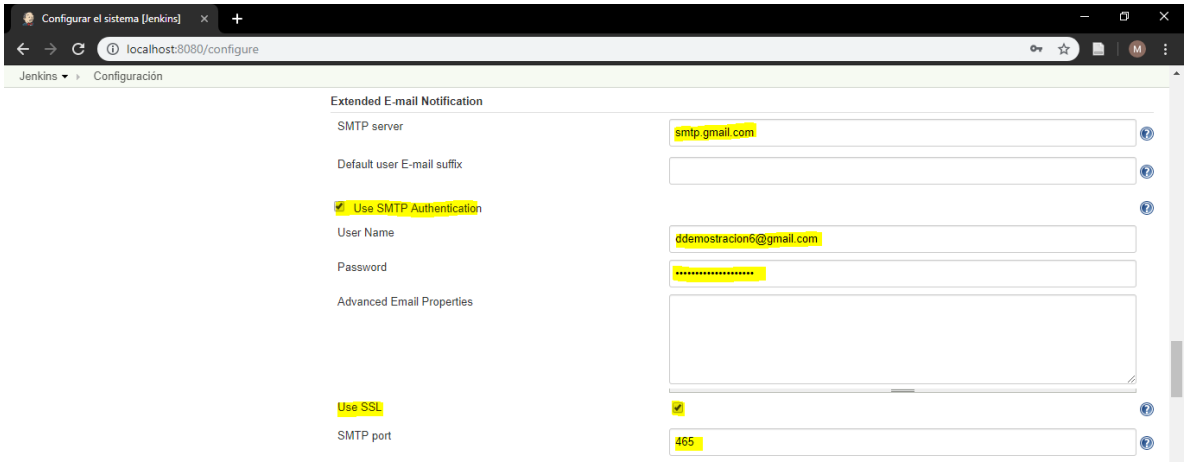
- **Name**: nombre del servidor, puede ser cualquier nombre (en este caso se nombró SonarQube Server)
- **URL del servidor:** Agregue la Url que ngrok.exe dio a SonarQube (en este caso se utilizó http://c4c1be9d.ngrok.io)

NO ingrese más información.



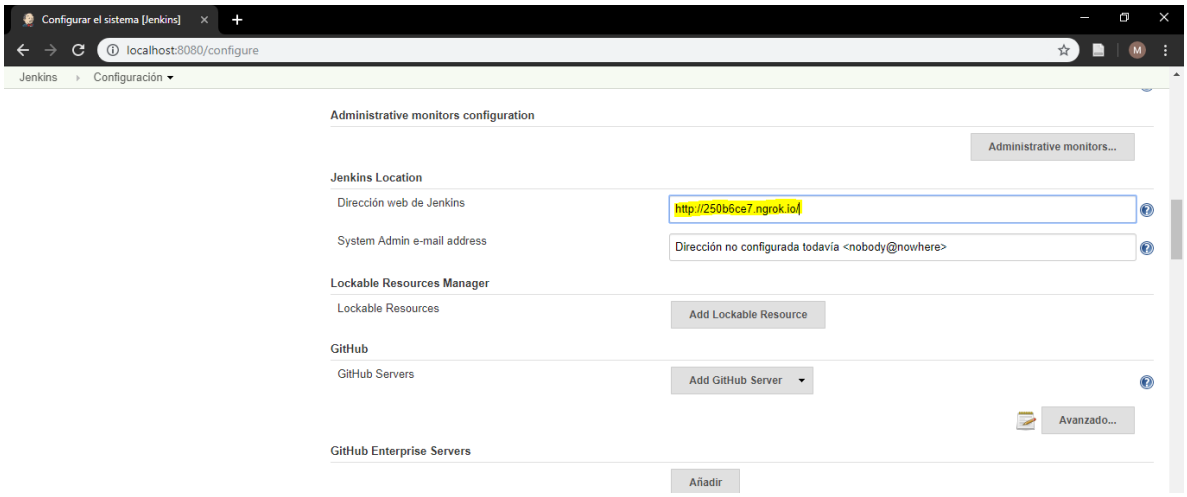
12.6. Busque la sección de configuración Extended E-mail Notification y complete los siguientes campos:

- **SMTP server:** smtp.gmail.com
- Marque como activo del check de **SMTP Authentication**
- **User Name**: puede usar cualquier correo de Gmail, en este caso se utilizará el mismo de github.com ddemostracion6@gmail.com
- **Password**: Ingrese el password de la cuenta de correo Gmail que utilizá en el campo User Name , en este caso la contraseña es Passwordparamuestra1
- Marque activo el check de **Use SSL**
- **SMTP port:** 465

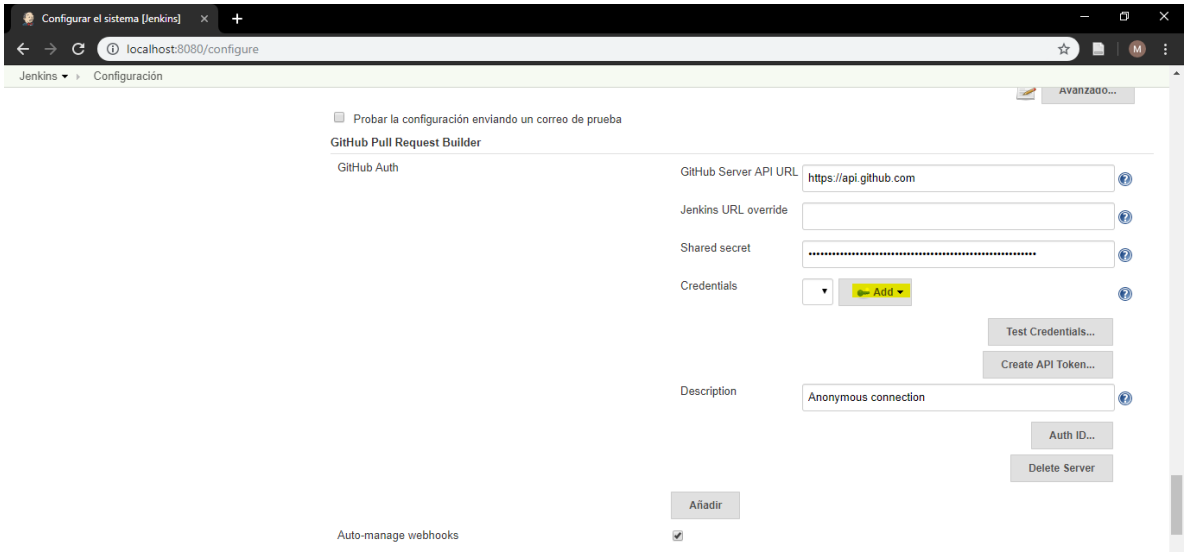


12.7. Busque la sección de Jenkins Location y agregue en el campo Dirección web de Jenkins , la Url que ngrok.exe brindo anteriormente, en este caso práctico fue: http://250b6ce7.ngrok.io

Recuerde que esta dirección solo tiene una duración de 8 horas y será cambiada después de este tiempo.



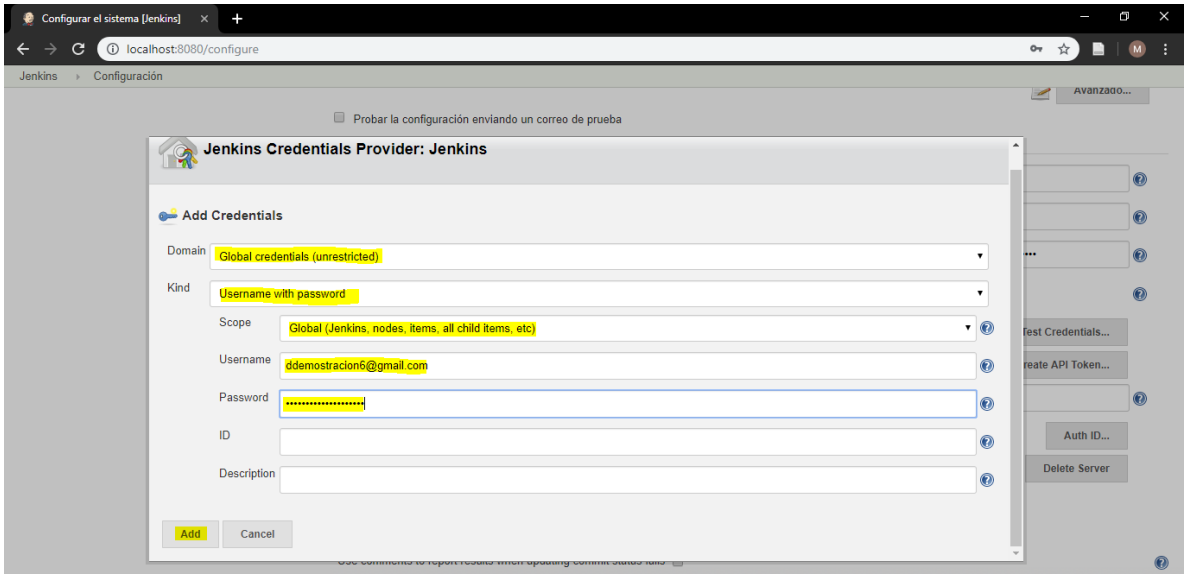
12.8. Busque la sección GitHub Pull Request Builder y oprima el botón ADD, para agregar las credenciales del usuario anteriormente creado para Github.com



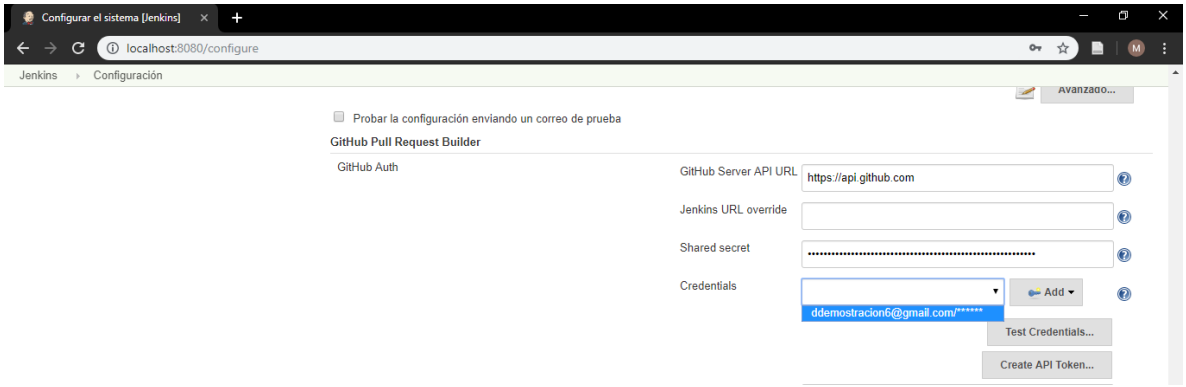
12.9. En la ventana emergente complete la siguiente información.

- **Domain**: Global credentials (unrestricted)
- **Kind** : Username with password
- **Scope**: Global (Jenkins, nodes, ítems, all child ítems, etc)
- **Username**: utilice el correo electrónico con el que ingresa a la cuenta de github.com, en este caso se utilizará, ddemostracion6@gmail.com
- **Password**: utilice el password de la cuenta de github.com anteriormente creada, en este caso se utilizará, Passwordparamuestra1

Para guardar las credenciales oprima el botón Add.

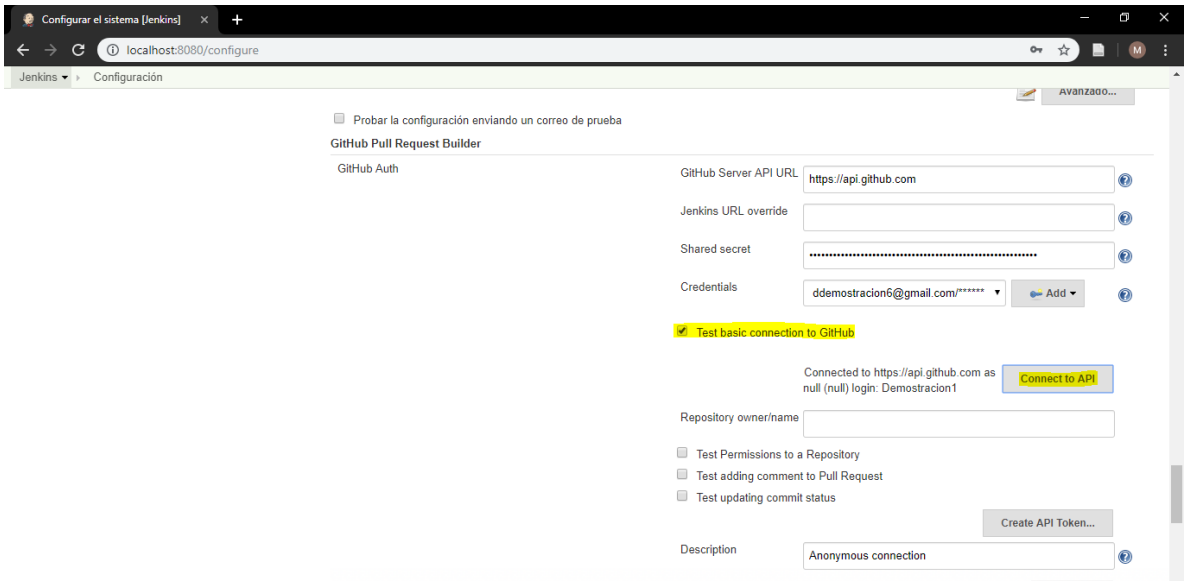


12.10. En el campo credenciales, seleccione aquellas que acaba de guardar.

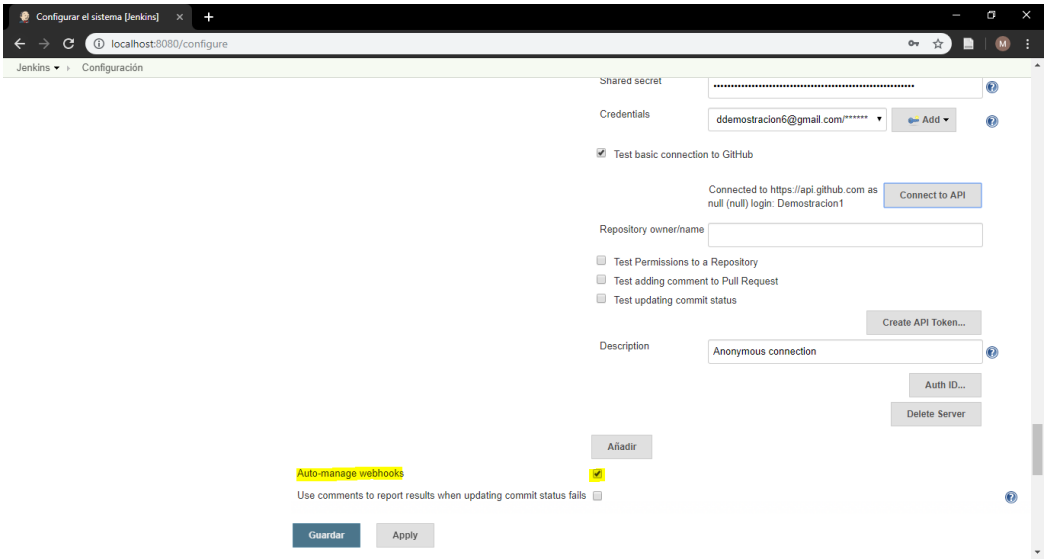


12.11. Para probar la configuración, oprima el botón Test Credentials → marque la casilla de Test basic connection to Github → Oprima el botón Connect to API

Después de realizar estos pasos observará un mensaje indicando una conexión exitosa.



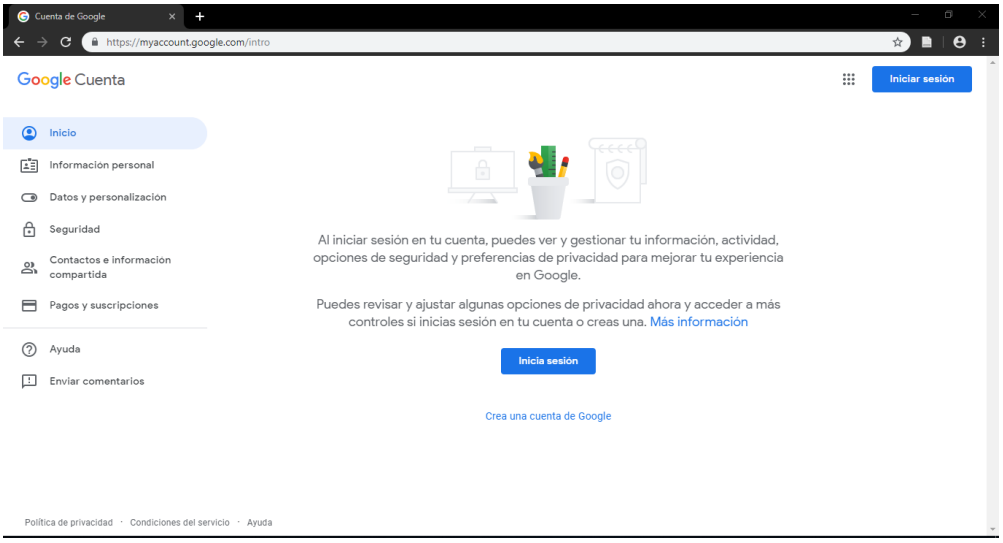
12.12. Finalmente marque como activa la casilla Auto manage webhooks y oprima el botón Guardar



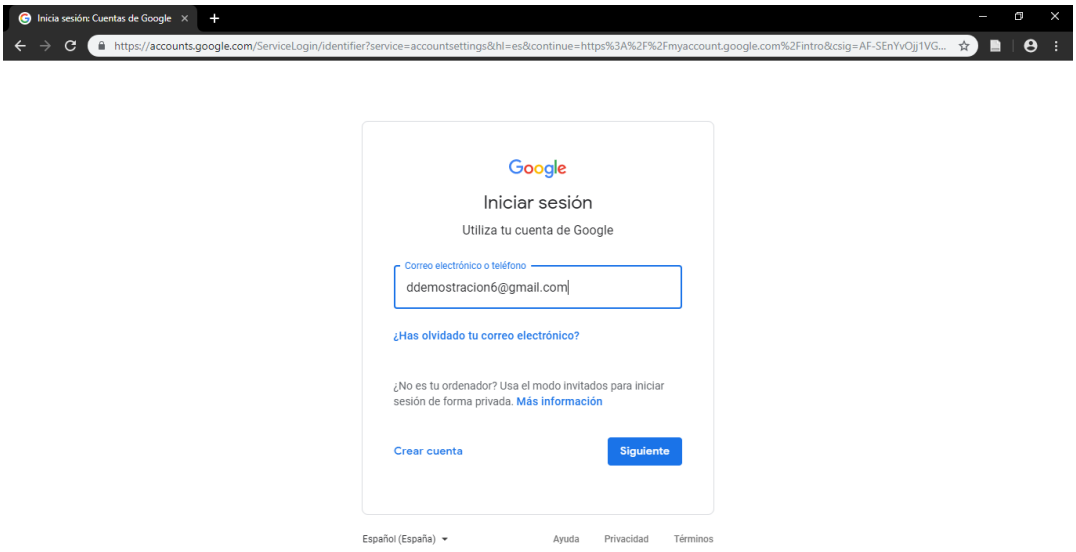
13. DISMINUIR LA SEGURIDAD EN LA CUENTA DE GMAIL.

Es necesario disminuir la seguridad de la cuenta utilizada en la configuración SMTP del plugin EMAIL EXTENSION TEMPLATE, así Google permitirá la conexión, consumo de los servicios expuestos por Gmail y él envió de correos.

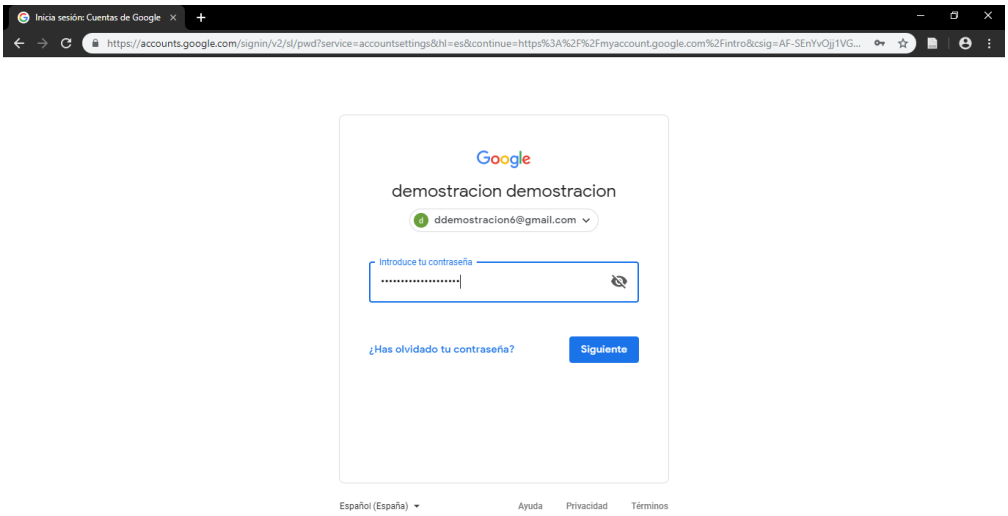
13.1. Ingrese a la siguiente Url, perteneciente a las cuentas de Google https://myaccount.google.com/intro → realice click sobre el botón iniciar sesión.



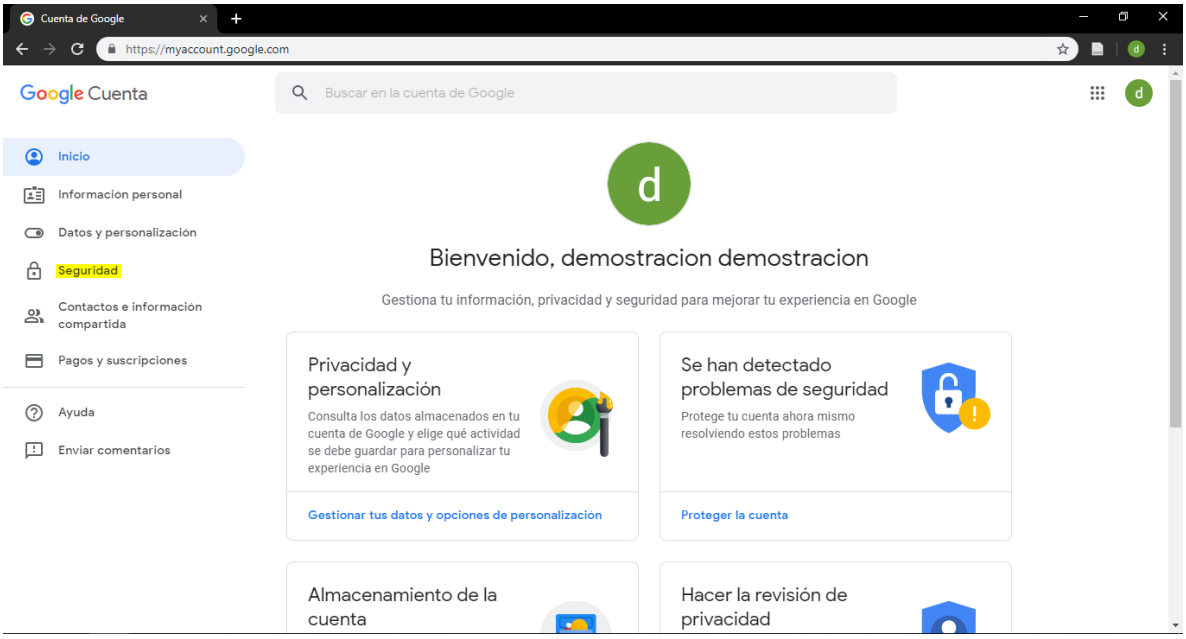
13.2. Ingrese el correo electrónico que uso para la configuración del SMTP del plugin EMAIL EXTENSION TEMPLATE, en este caso práctico se utilizó el correo ddemostracion6@gmail.com , luego oprima el botón Siguiente.



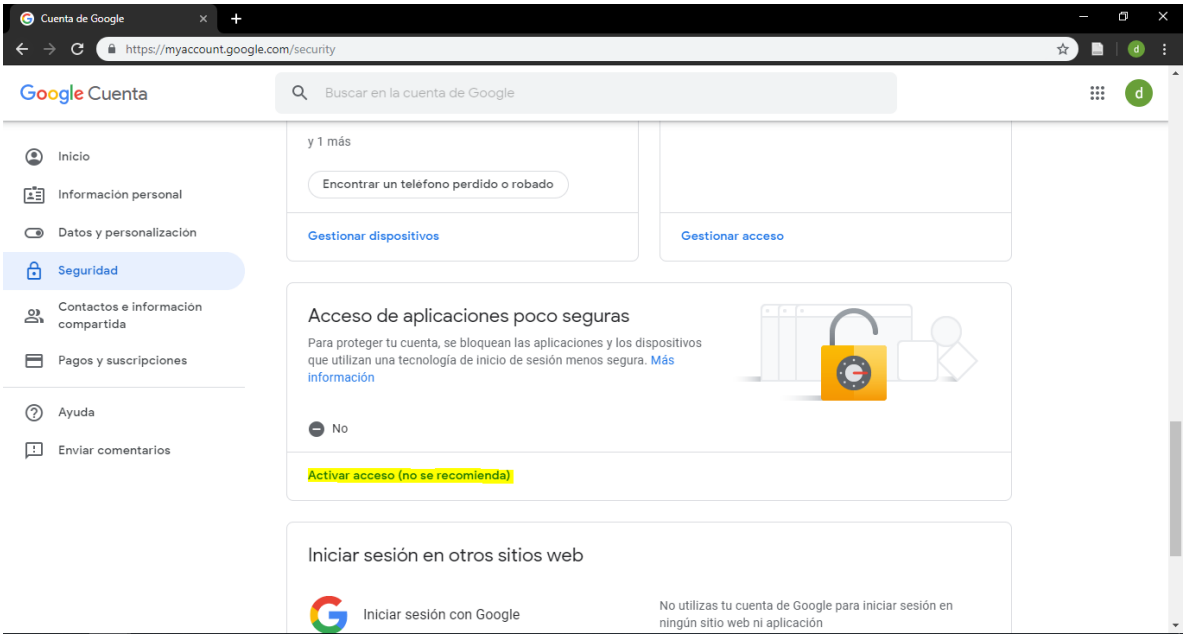
13.3. Introduzca la contraseña de la cuenta de correo Gmail utilizada en la configuración SMTP del plugin EMAIL EXTENSION TEMPLATE, en este caso práctico se utilizó el correo ddemostracion6@gmail.com , y su contraseña es Passwordparamuestra1 , después de ingresar la contraseña oprima el botón Siguiente.



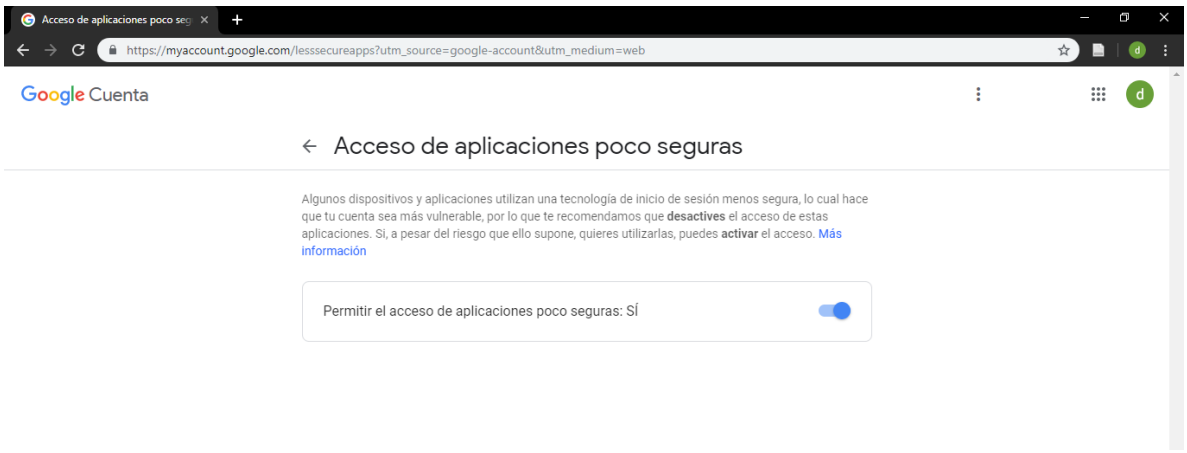
13.4. Ya ubicado en el panel principal de la cuenta de Google , seleccione la opción Seguridad



13.5. Busque la sección de Acceso de aplicaciones poco seguras y oprima la opción Activar acceso.



13.6. Permita el acceso de aplicaciones poco seguras

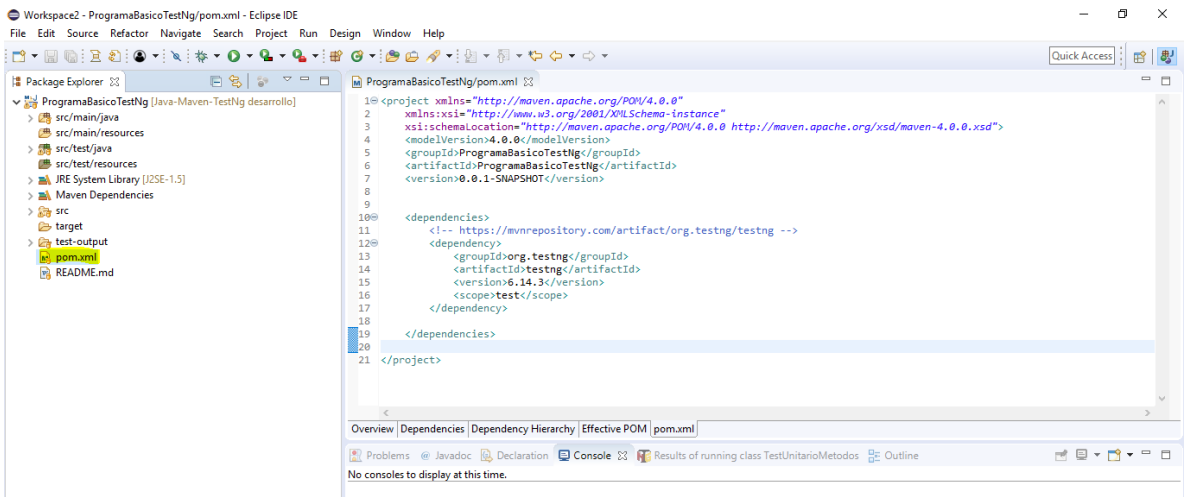


14. CONFIGURAR PROYECTO JAVA PARA QUE SONARQUBE PUEDA CONOCER SU COBERTURA.

Para que SonarQube conozca la cobertura del código por pruebas unitarias, se agrega el plugin JaCoCo en el archivo pom.xml del proyecto Java.

Este plugin se encarga de crear reportes e información necesaria para SonarQube acerca de la cobertura del proyecto.

14.1. Ejecute la herramienta “Eclipse” → Busque su proyecto Java versionado → Realice doble click sobre el archivo “pom.xml”



14.2. Ubique el cursor dentro del archivo pom.xml y después de la palabra **</dependencias> y antes de la palabra </project>** agregue las siguientes instrucciones

```
<build>
		<plugins>
			<plugin>
				<groupId>org.jacoco</groupId>
				<artifactId>jacoco-maven-plugin</artifactId>
				<version>0.7.5.201505241946</version>
				<executions>
					<execution>
						<id>prepare-agent</id>
						<goals>
							<goal>prepare-agent</goal>
						</goals>
					</execution>
					<execution>
						<id>report</id>
						<phase>prepare-package</phase>
						<goals>
							<goal>report</goal>
						</goals>
					</execution>
					<execution>
						<id>post-unit-test</id>
						<phase>test</phase>
						<goals>
							<goal>report</goal>
						</goals>
						<configuration>
							<excludes>

	<exclude>**/principal/*</exclude>
<exclude>**/test/*</exclude>
							</excludes>

	<dataFile>target/jacoco.exec</dataFile>
<outputDirectory>target/jacoco-
ut</outputDirectory>
						</configuration>
					</execution>
				</executions>
				<configuration>
					<systemPropertyVariables>
```

```
                                        <jacoco-
agent.destfile>target/jacoco.exec</jacoco-agent.destfile>
                                    </systemPropertyVariables>
                                </configuration>
                            </plugin>
                        </plugins>
                        </build>
```

14.3. El archivo será similar a la siguiente imagen.

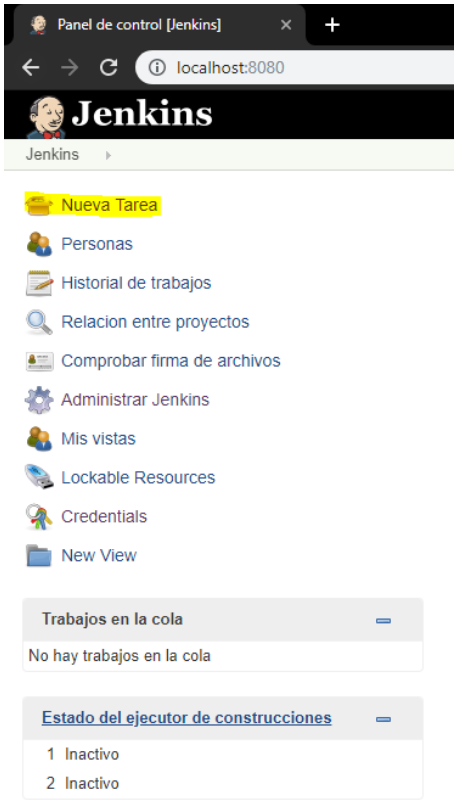
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>ProgramaBasicoTestNg</groupId>
    <artifactId>ProgramaBasicoTestNg</artifactId>
    <version>0.0.1-SNAPSHOT</version>


    <dependencies>
        <!-- https://mvnrepository.com/artifact/org.testng/testng -->
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>6.14.3</version>
            <scope>test</scope>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.jacoco</groupId>
                <artifactId>jacoco-maven-plugin</artifactId>
                <version>0.7.5.201505241946</version>
                <executions>
                    <execution>
                        <id>prepare-agent</id>
                        <goals>
                            <goal>prepare-agent</goal>
                        </goals>
                    </execution>
                    <execution>
                        <id>report</id>
                        <phase>prepare-package</phase>
                        <goals>
                            <goal>report</goal>
                        </goals>
                    </execution>
                    <execution>
                        <id>post-unit-test</id>
                        <phase>test</phase>
                        <goals>
                            <goal>report</goal>
                        </goals>
                        <configuration>
                            <excludes>
                                <exclude>**/principal/*</exclude>
                                <exclude>**/test/*</exclude>
                            </excludes>
                            <dataFile>target/jacoco.exec</dataFile>
                            <outputDirectory>target/jacoco-ut</outputDirectory>
                        </configuration>
                    </execution>
                </executions>
                <configuration>
                    <systemPropertyVariables>
                        <jacoco-agent.destfile>target/jacoco.exec</jacoco-agent.destfile>
                    </systemPropertyVariables>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

14.4. Después de agregar el código guarde los cambios con la opción guardar en la barra de herramientas superior u oprimiendo Ctrl + s y realice los pasos 4.28 → 4.29 → 4.30, con estos pasos se actualizarán las dependencias y descargara el plugin JaCoCo al proyecto java.

14.5. El proyecto posee cambios de código y el plugin JaCoCo, es necesario versionarlo nuevamente en la rama desarrollo, para esto siga los pasos 5.15 → 5.16 → 5.17 → 5.18.

15. CONFIGURACIÓN DE JOB (PIPELINE) EN JENKINS PARA LA EJECUCIÓN AUTOMÁTICA DE PRUEBAS UNITARIAS, REVISIÓN EN SONARQUBE Y REPORTE POR MEDIO DE CORREO ELECTRÓNICO.

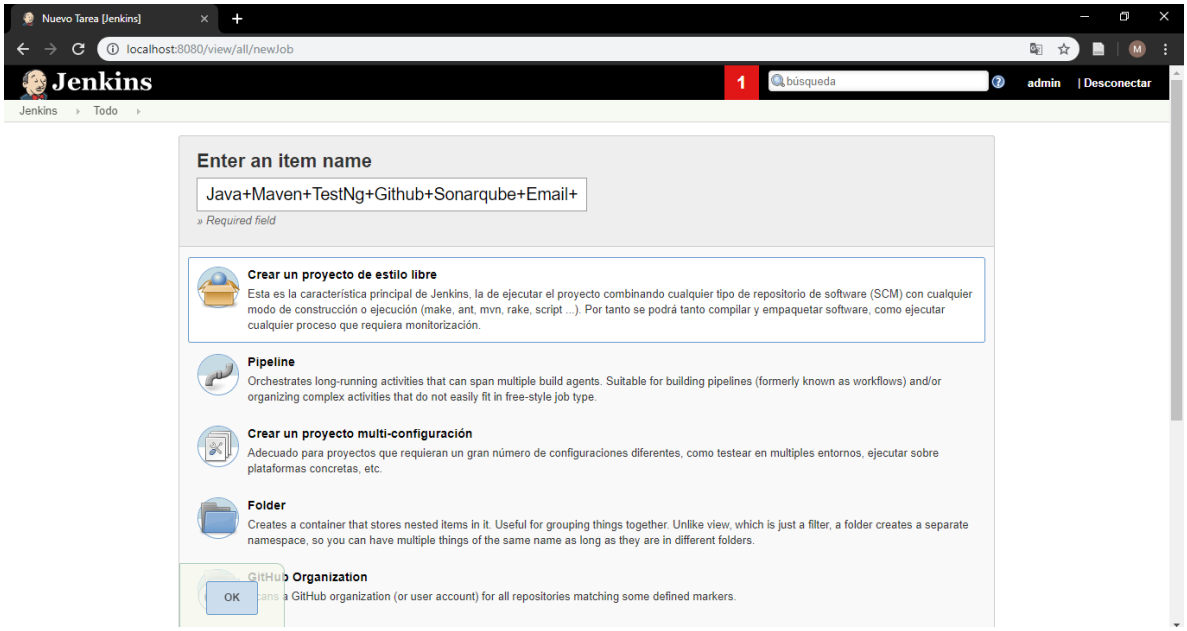
15.1. Ingrese a Jenkins → En el panel principal seleccione la opción Nueva Tarea.



15.2. Ingrese el nombre de la tarea en el campo Enter an ítem name, en este caso se ingresó: “Java+Maven+TestNg+Github+Sonarqube+Email+Jenkins”

Seleccione la opción: Crear un proyecto de estilo libre

Finalmente oprima el botón OK



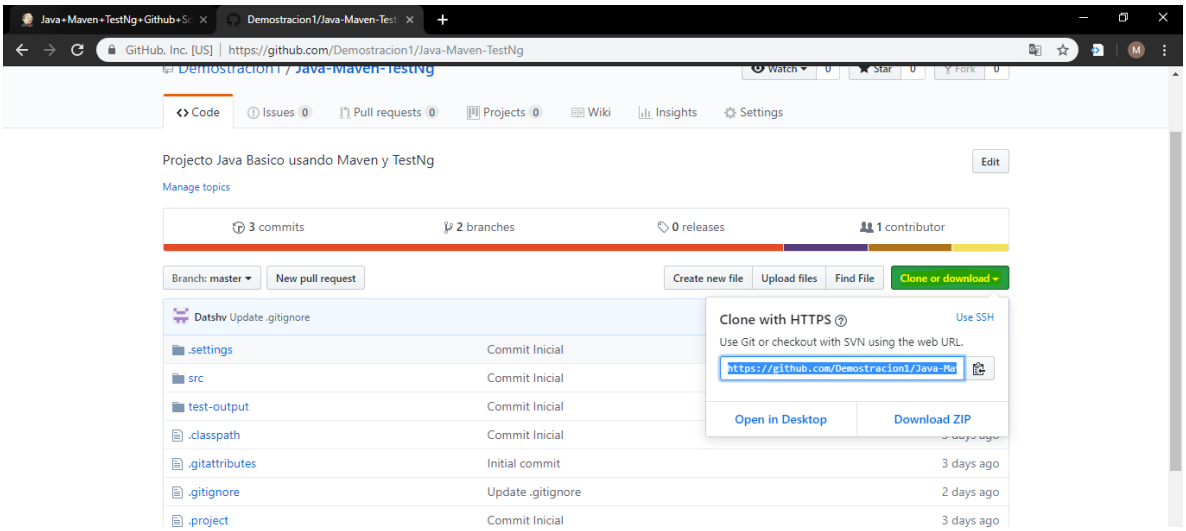
15.3. No cierre la ventana del explorador donde se encuentra Jenkins, y abra otra pestaña e ingrese a la cuenta de GitHub.com donde se encuentra el proyecto versionado de java, en este caso se utilizaron los siguientes datos:

- **Username** : ddemostracion6@gmail.com
- **Password**: Passwordparamuestra1

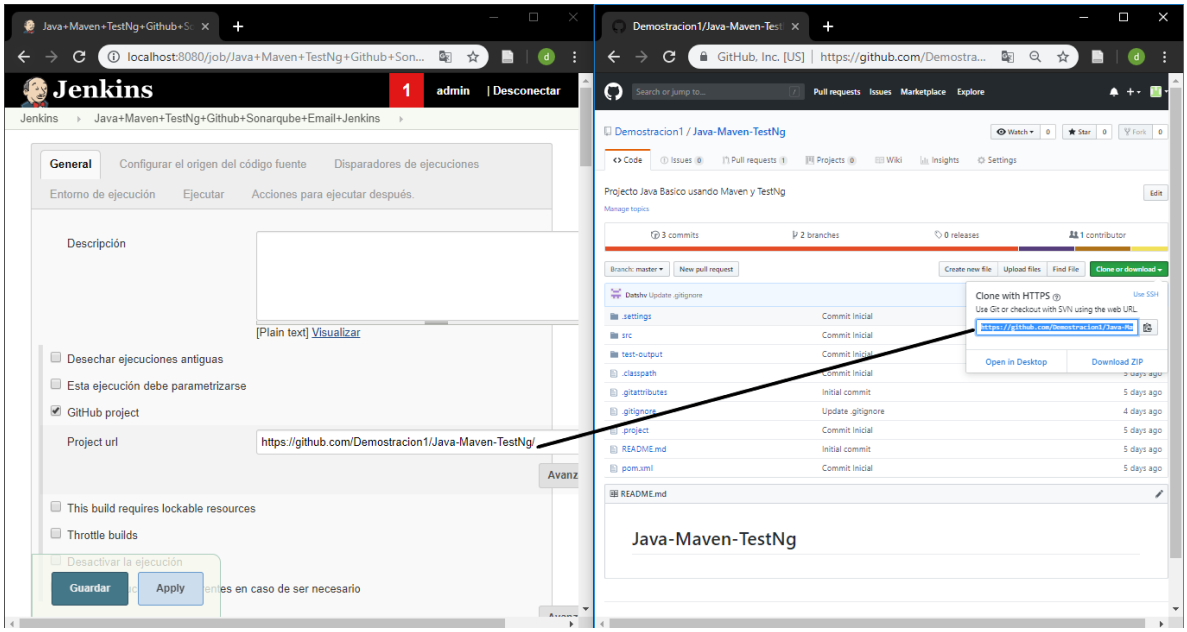
Ingrese al repositorio donde se encuentra su proyecto versionado→ oprima el botón Clone or download→ copie la Url de la ventana emergente , en este caso la Url era: https://github.com/Demostracion1/Java-Maven-TestNg.git , a esta Url remueva la palabra .git y agregue el símbolo / la Url lucirá del siguiente modo después de las modificaciones:

Original: https://github.com/Demostracion1/Java-Maven-TestNg.git

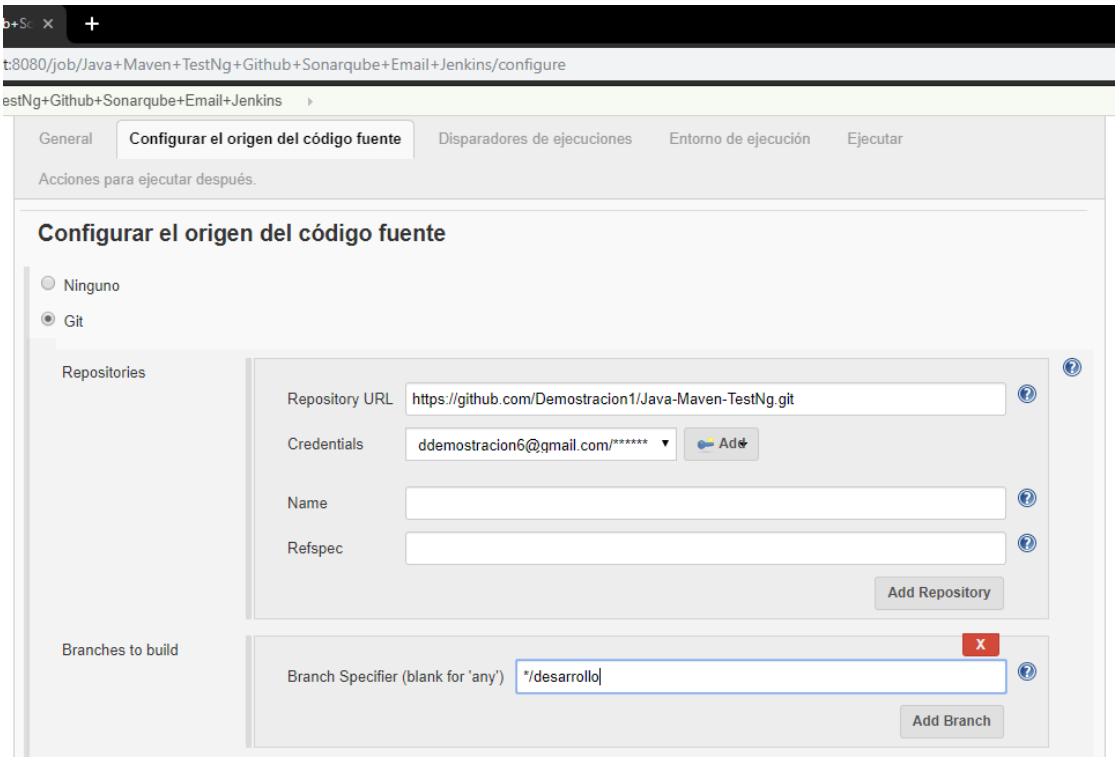
Modificada: https://github.com/Demostracion1/Java-Maven-TestNg/



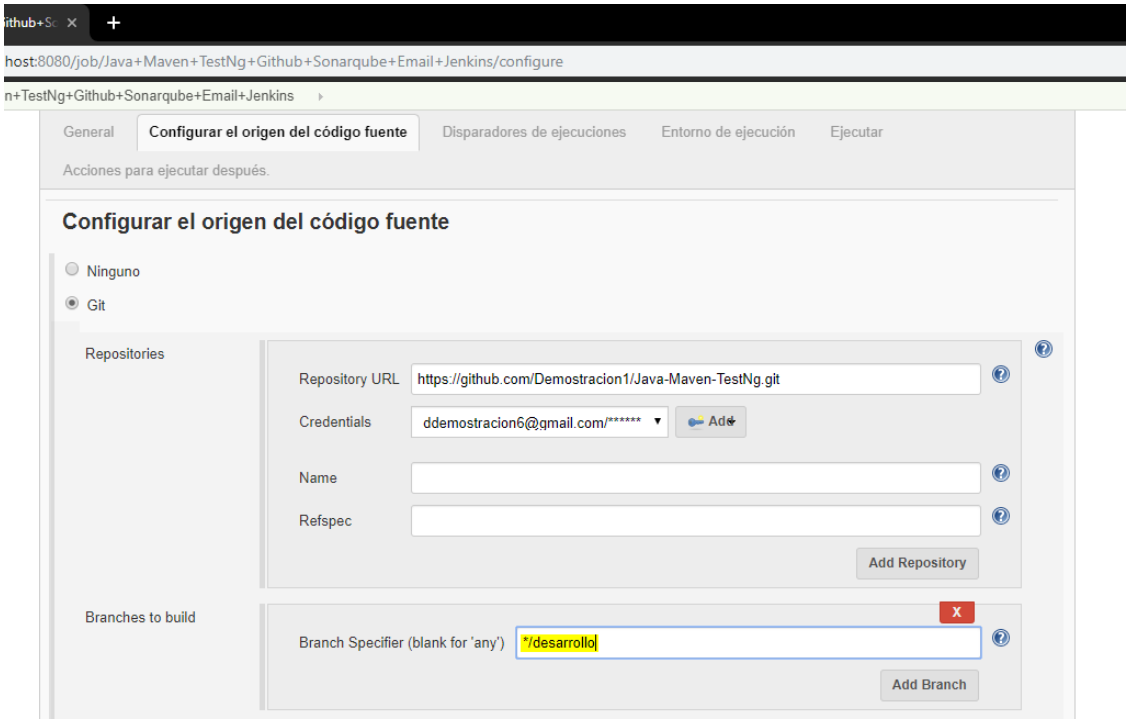
15.4. En la ventana de configuración del Job de Jenkins, en la sección General, marque como activo el check de GitHub Project y pegue la Url modificada desde github.com, en este caso sería: https://github.com/Demostracion1/Java-Maven-TestNg/



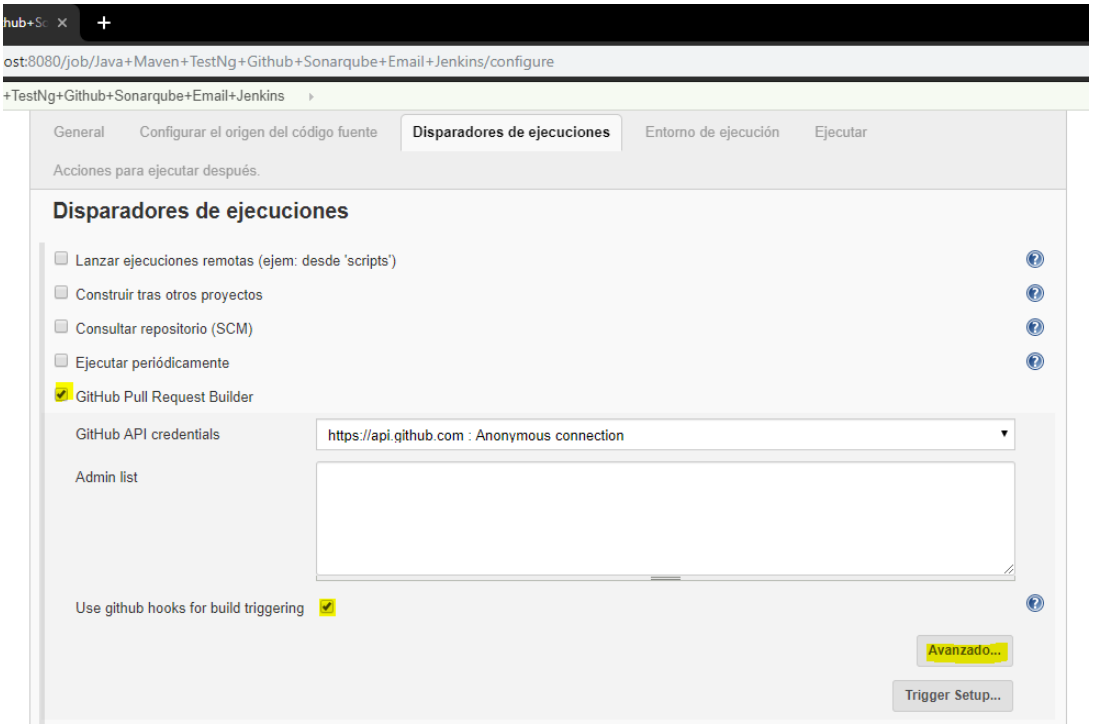
15.5. En la sección Configurar el origen del código fuente, marque como activo el check Git→ agregue en Repository URL la Url de Github.com perteneciente a su repositorio → En la lista Credentials seleccione las credenciales creadas anteriormente, las cuales pertenecen a los datos necesario para ingresar a la cuenta de Github.com que poseen el repositorio del proyecto java.



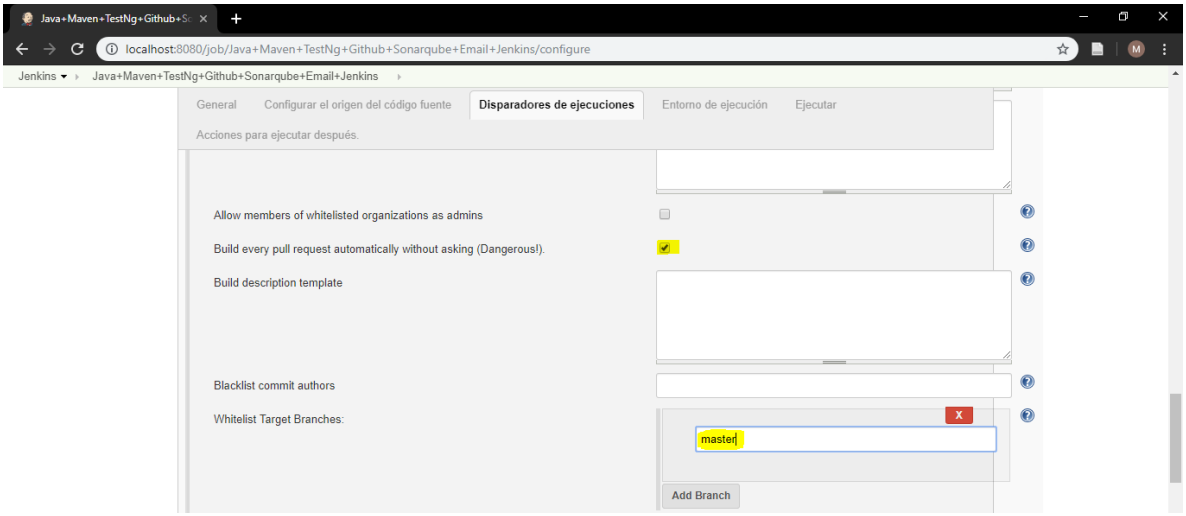
15.6. En el campo Branch Specifier (blank for ‘any’), agregue la rama que se construirá en Jenkins cuando se ejecuten las pruebas unitarias, el proyecto en github.com posee dos ramas, llamadas master y desarrollo, en este caso utilizaremos la rama desarrollo



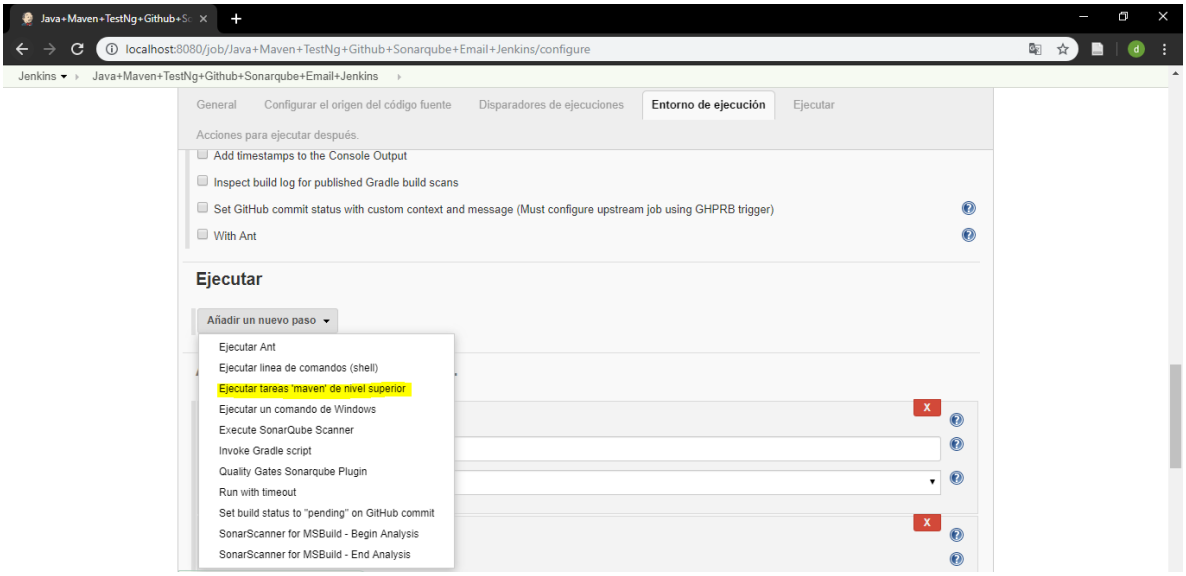
15.7. En la sección Disparadores de ejecución, marque como activa la casilla GitHub Pull Request Builder → NO modifique los datos de GitHub API credentials → marque como activo el check Use github hooks for build triggering→ oprima el botón Avanzado…



15.8. En la lista de opciones que se despliegan luego de realizar click en el botón Avanzado.. → Busque la opción Build every pull request automatically without asking (Dangerous!) y márquela como activa → en el campo Whitelist Target Branches ingrese el nombre de la rama master.



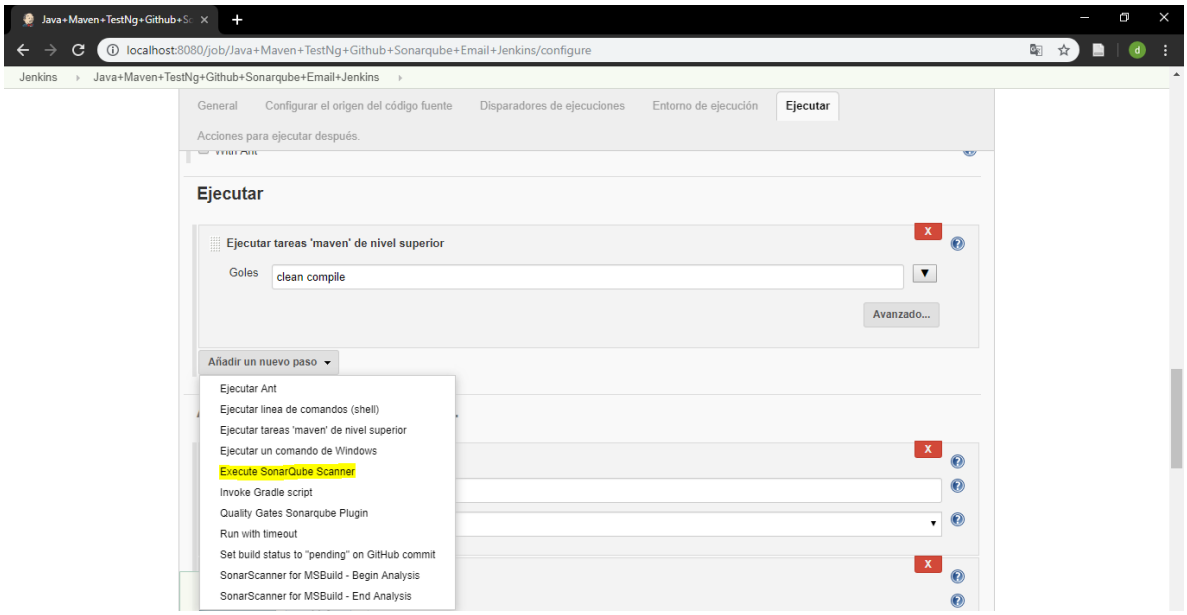
15.9. En la sección Ejecutar → click sobre la lista “Añadir un nuevo paso” → seleccione “Ejecutar tareas ‘maven’ de nivel superior”.



15.10. En el campo “Goles” agregue las siguientes instrucciones:

Clean compile test

15.11. Realice click sobre el botón “Añadir un nuevo paso” → seleccione la opción “Execute SonarQube Scanner”

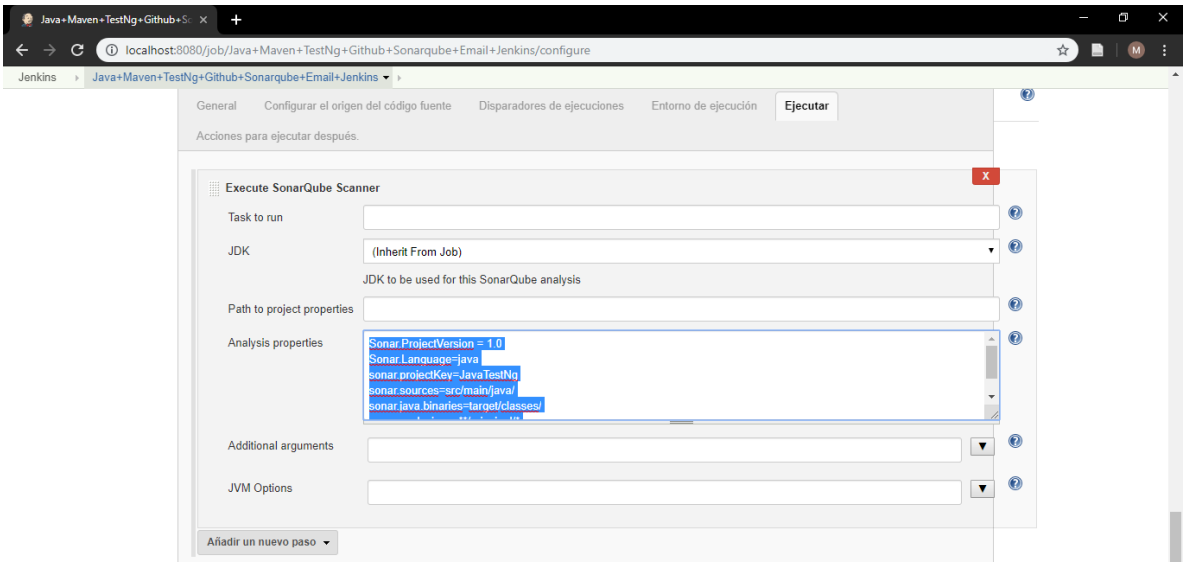


15.12. En el campo Analysis properties agregue el siguiente código

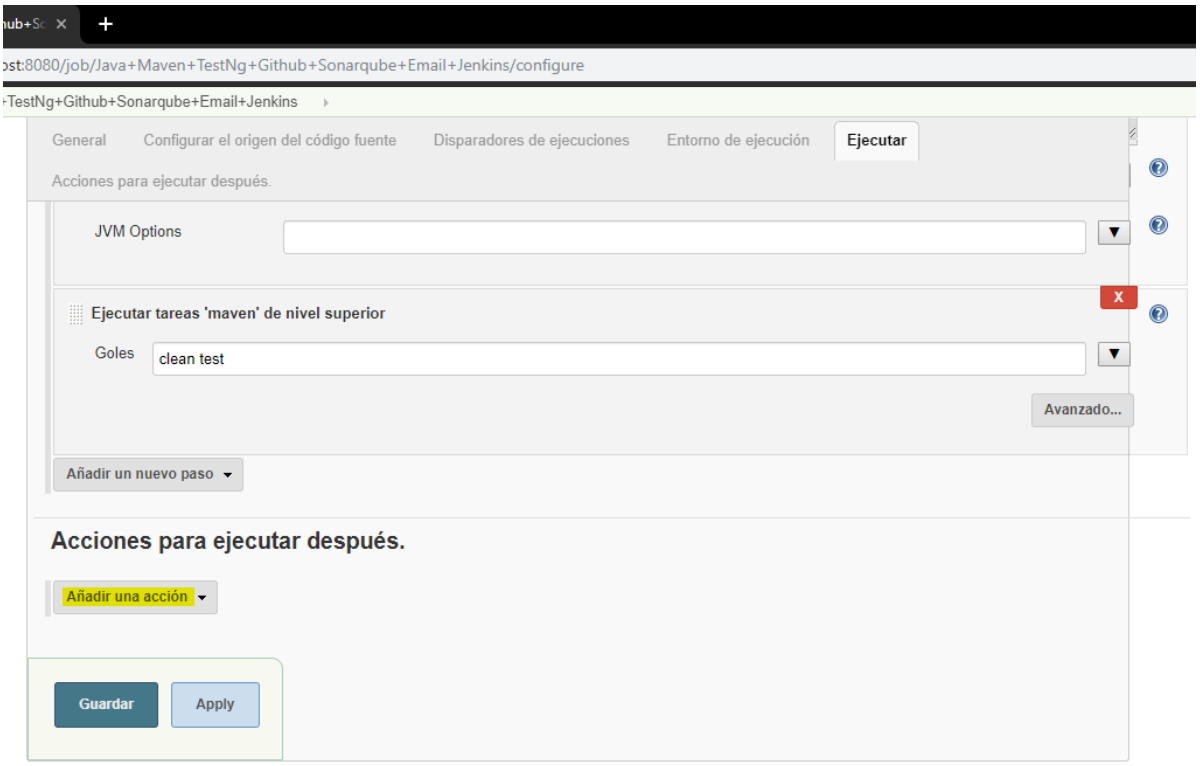
```
Sonar.ProjectVersion = 1.0
Sonar.Language=java
sonar.projectKey=JavaTestNg
sonar.sources=src/main
sonar.java.binaries=target/classes/
sonar.exclusions=**/principal/*
sonar.tests=src/test
```

Explicación del código:

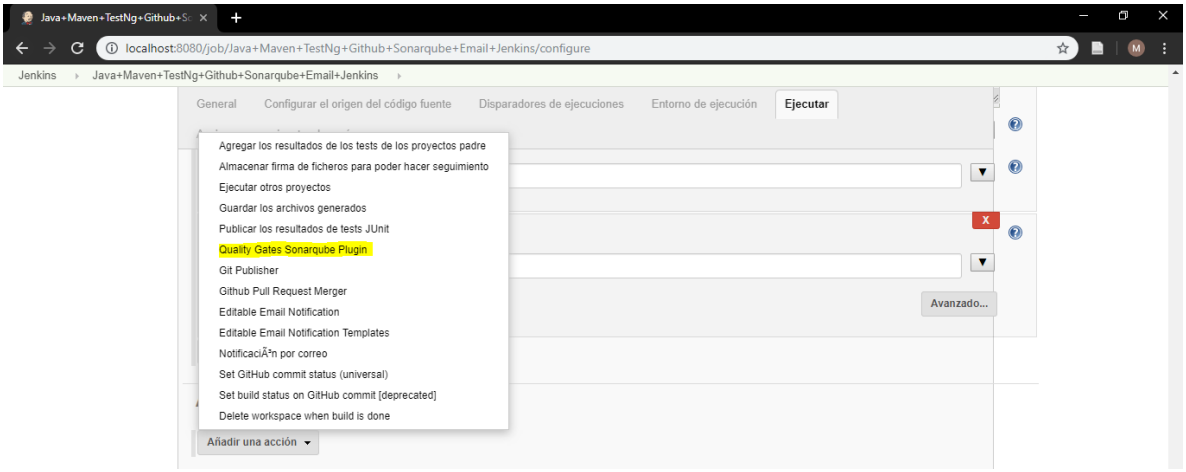
- Sonar.ProjectVersion: Es la versión del proyecto que se ejecutara en SonarQube
- Sonar.Language: Especifica el lenguaje en el cual el código fuente está escrito.
- sonar.projectKey: Es el nombre que el proyecto tendrá en SonarQube.
- Sonar.Sources: Indica la carpeta donde se encuentra el código que SonarQube revisara.
- sonar.java.binaries: Indica la carpeta en la cual se encuentran los binarios de las clases.
- sonar.exclusions: Indica que carpetas o archivos no serán revisados por SonarQube, la carpeta principal solo posee una clase llamada Principal.java, la cual no posee métodos necesarios para revisión, será excluida de la revisión.
- sonar.tests: Indica la carpeta o archivos pertenecientes a las pruebas unitarias sobre el código.



15.13. En la sección “Acciones para ejecutar después” → realice click sobre la lista desplegable “Añadir una acción”.



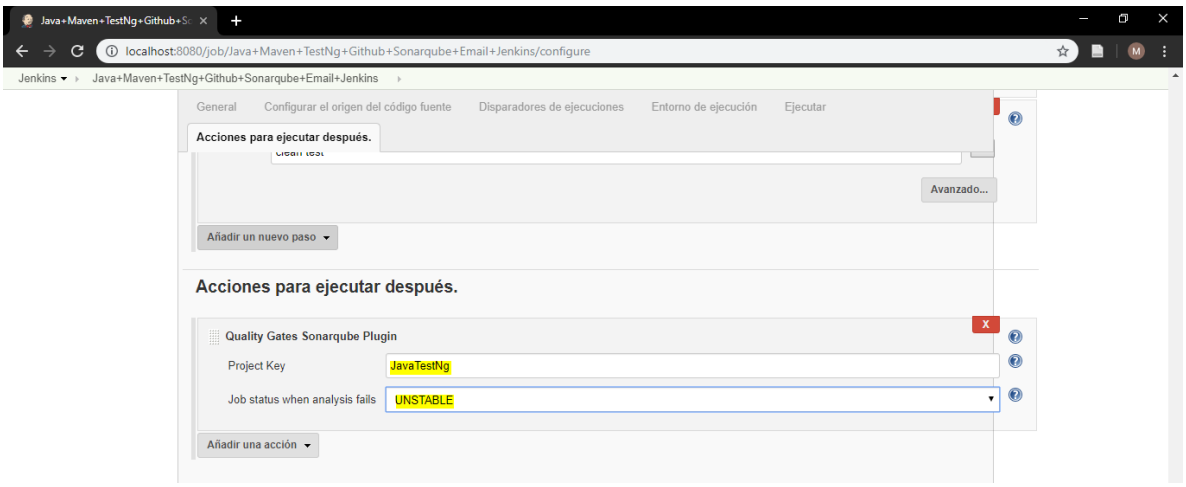
15.14. Seleccione la opción “Quality Gates Sonarqube Plugin”



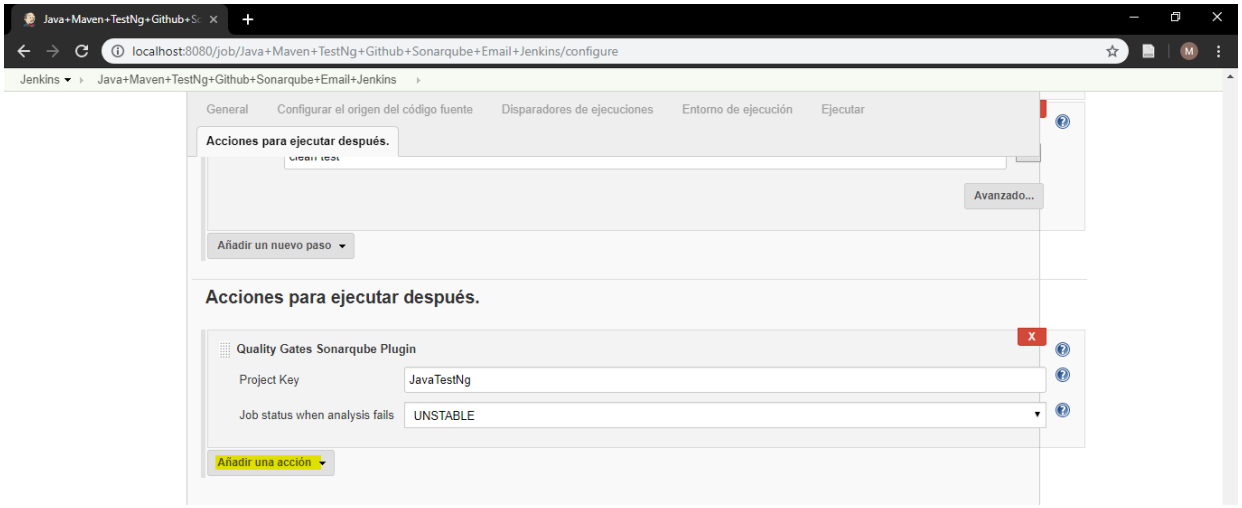
15.15. En el campo “Project Key” ingrese el nombre utilizado en la propiedad sonar.projectKey del paso 13.10.

En este caso práctico se usó JavaTestNg.

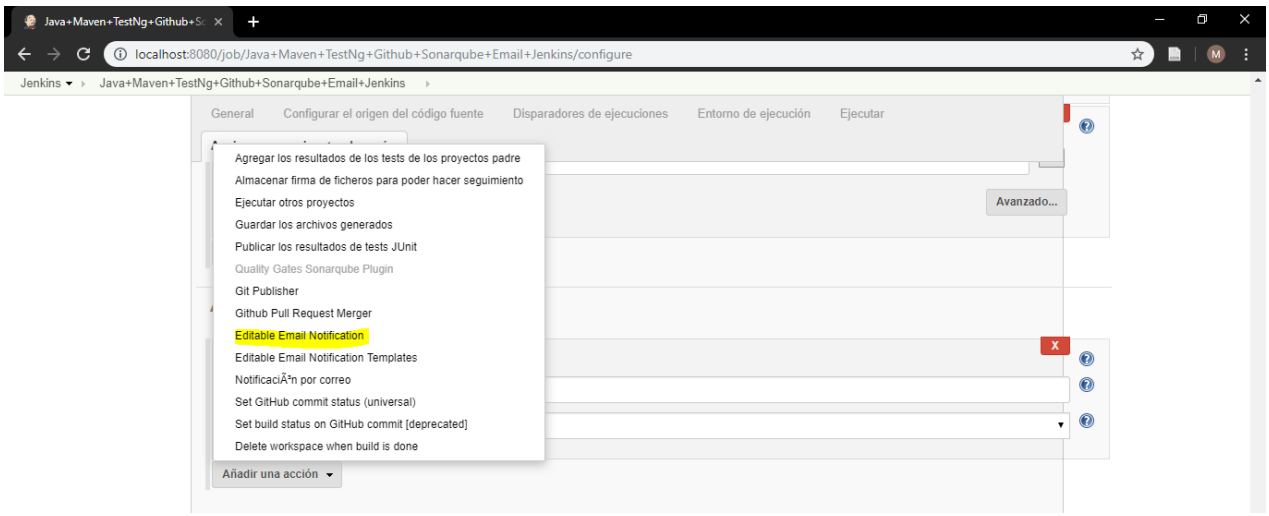
Después de llenar el campo realice click sobre la lista del campo “Job status when analysis fails” → seleccione “UNSTABLE”



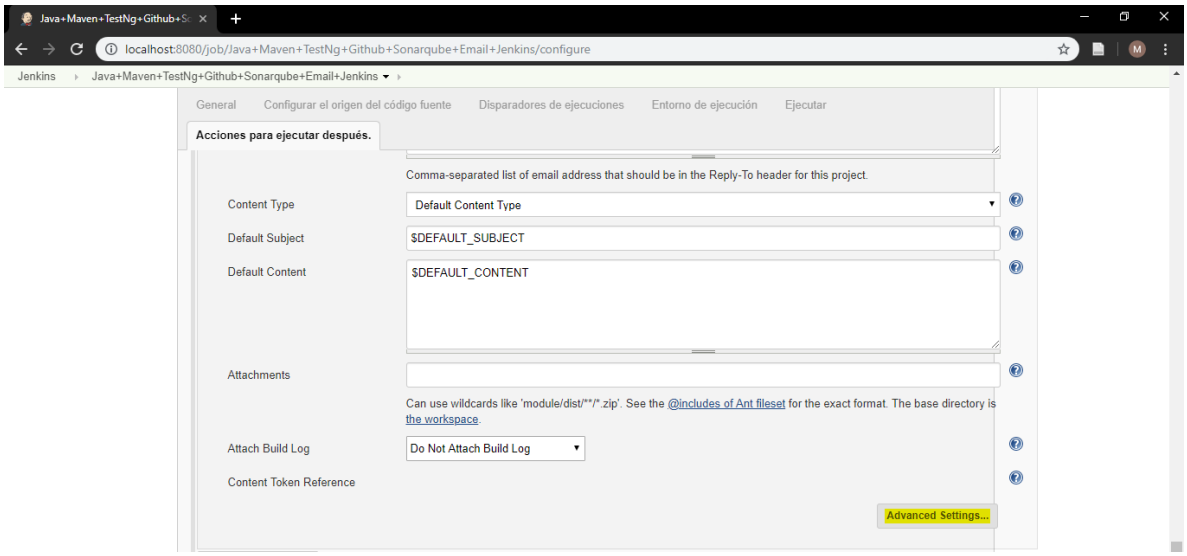
15.16. En la sección “Acciones para ejecutar después.” → realice click sobre el botón “Añadir una acción”.



15.17. Seleccione la opción “Editable Email Notification” de la lista desplegable



15.18. Realice click sobre el botón “Advanced Settings…”



15.19. En el campo “Pre-send script” agregue el siguiente código:

```
def reportPath = build.getWorkspace().child("target/surefire-reports/emailable-report.html")


def var = build.getEnvVars();
String mensaje = build.getLog();

int inicioDelMensaje = mensaje.indexOf("INFO: ANALYSIS SUCCESSFUL, you can browse",0)+41;
int finalDelMensaje = mensaje.indexOf("INFO: Note that you will be able to access the updated",0);
String Url ="SonarQube no fue ejecutado ya que existen pruebas fallando";
String calidad = "";
if(inicioDelMensaje > 0 && finalDelMensaje >0){
    Url= mensaje.substring(inicioDelMensaje,finalDelMensaje);
```

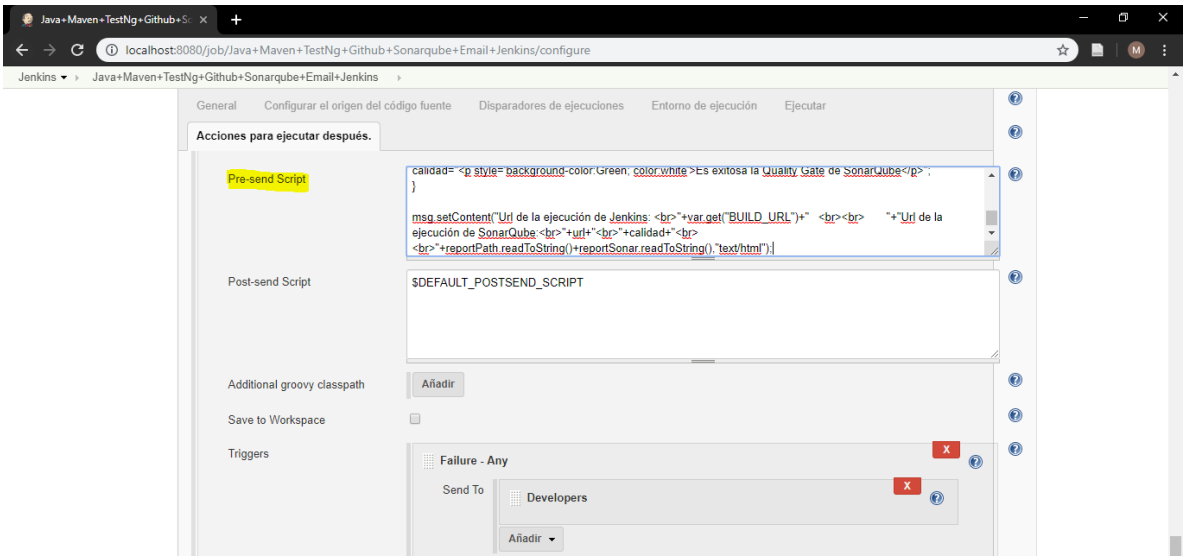
```
}
 inicioDelMensaje = mensaje.indexOf("Quality Gates plugin build passed:",0)+34;
finalDelMensaje = mensaje.indexOf("Build step 'Quality Gates",0);

if(inicioDelMensaje > 0 && finalDelMensaje >0 && !Url.equals("SonarQube no fue ejecutado ya que existen pruebas fallando")){
calidad= mensaje.substring(inicioDelMensaje,finalDelMensaje);
}else{
   calidad ="FALSE";
}


if(calidad.trim().equals("FALSE")){
    if(Url.equals("SonarQube no fue ejecutado ya que existen pruebas fallando")){
       calidad="";
   }else
   {
   calidad="<p style='background-color:red;color:white'>Ha fallado la Quality Gate de SonarQube porfavor revise.</p>";
   }
}else{

      calidad="<p style='background-color:Green; color:white'>Es exitosa la Quality Gate de SonarQube</p>";


}
```

msg.setContent("Url de la ejecución de Jenkins:
"+var.get("BUILD_URL")+"

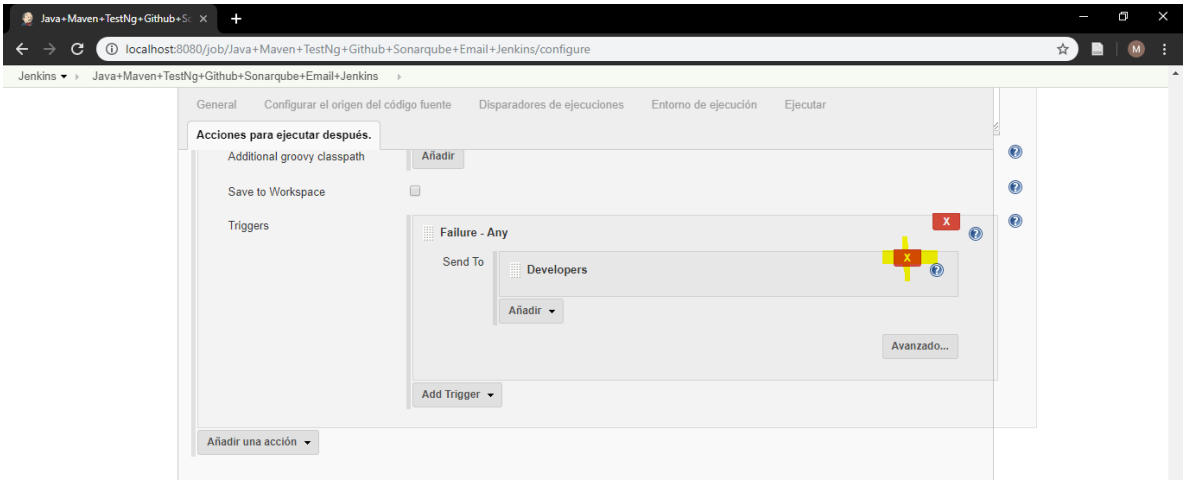
 "+"Url de la ejecución de SonarQube:
"+Url+"
"+calidad+"

"+reportPath.readToStrin
g(),"text/html");

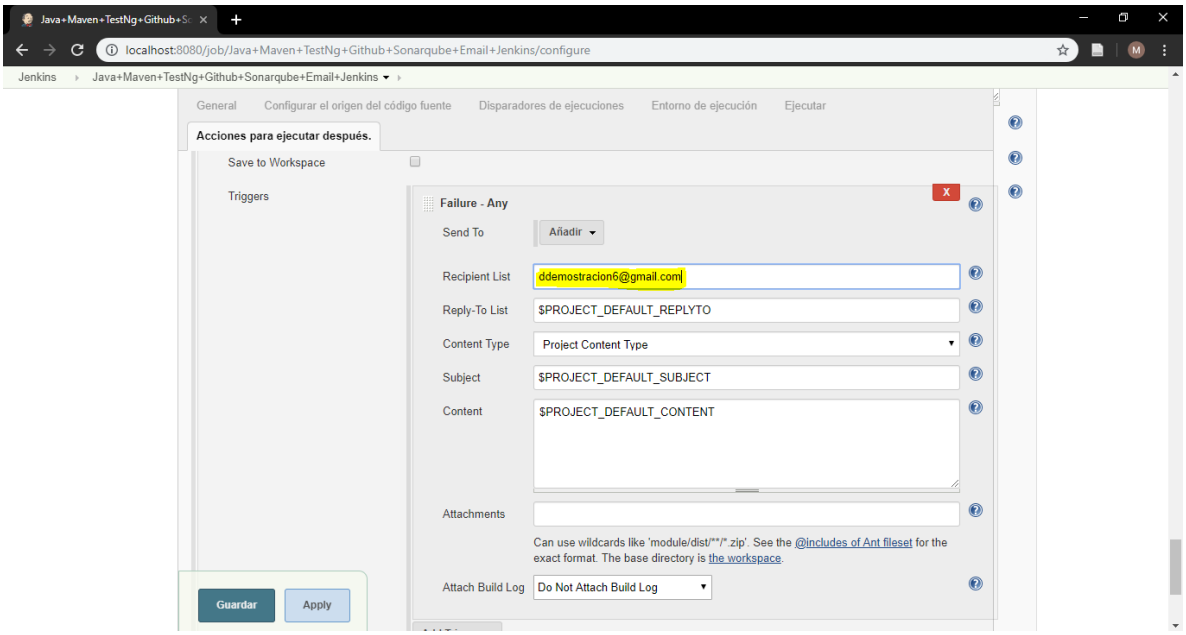
Este código generara el contenido del correo electrónico que se enviara en caso de que la Quality Gate se encuentre en estado Fallido o también en caso de que una de las pruebas unitarias falle.



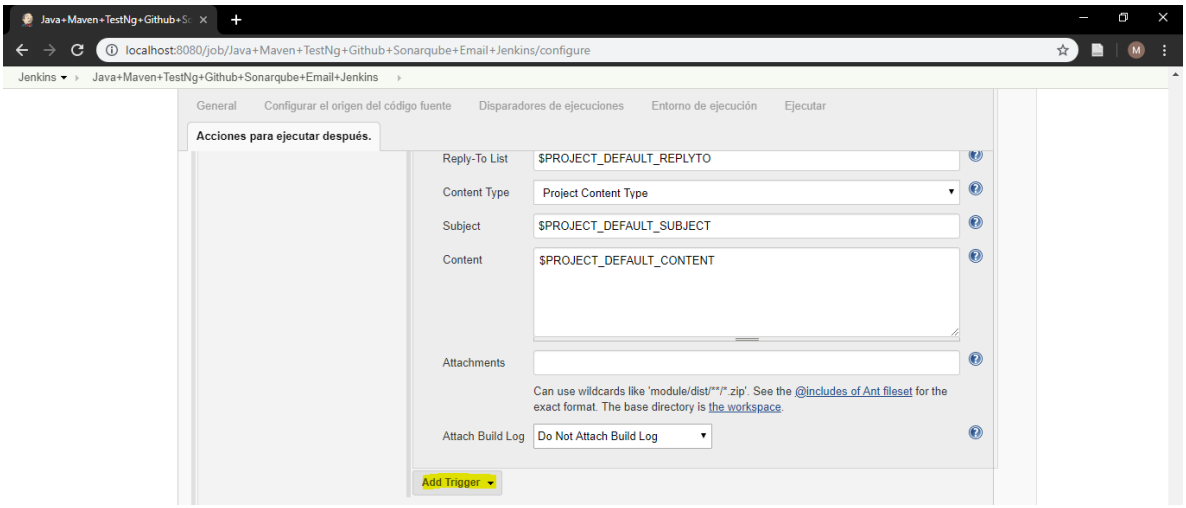
15.20. Realice click sobre el botón “X” de la casilla “Developers” para borrarla.



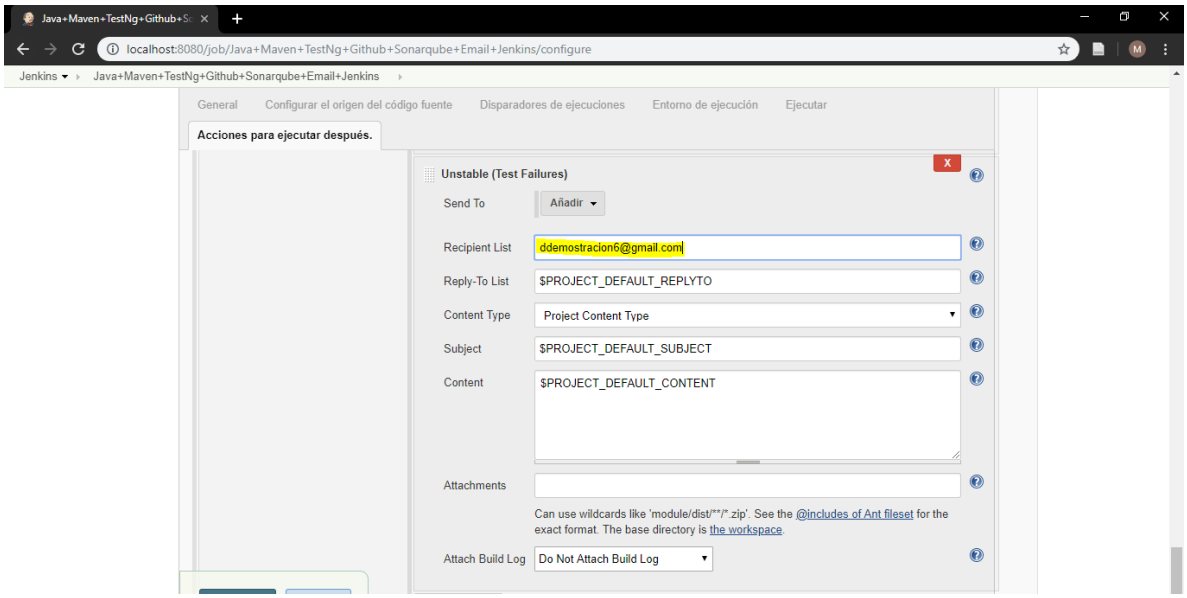
15.21. Oprima el botón “Avanzado..” → En el campo Recipient List escriba los correos a los cuales quiere notificar de la ejecución fallida de Jenkins, en este caso se notificará al correo ddemostracion6@gmail.com



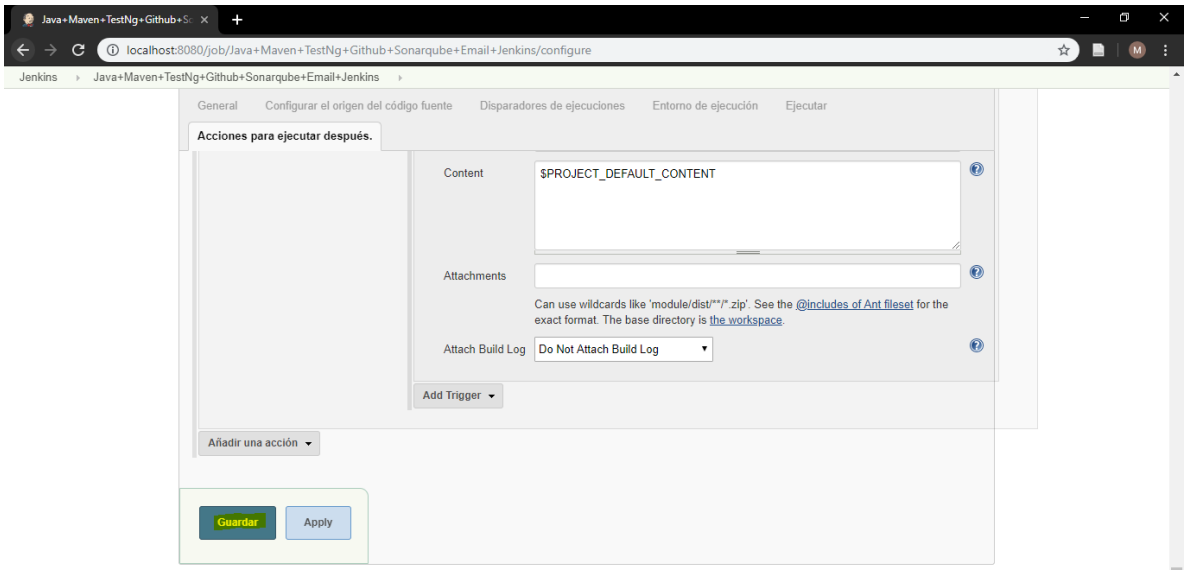
15.22. Oprima el botón Añadir Trigger



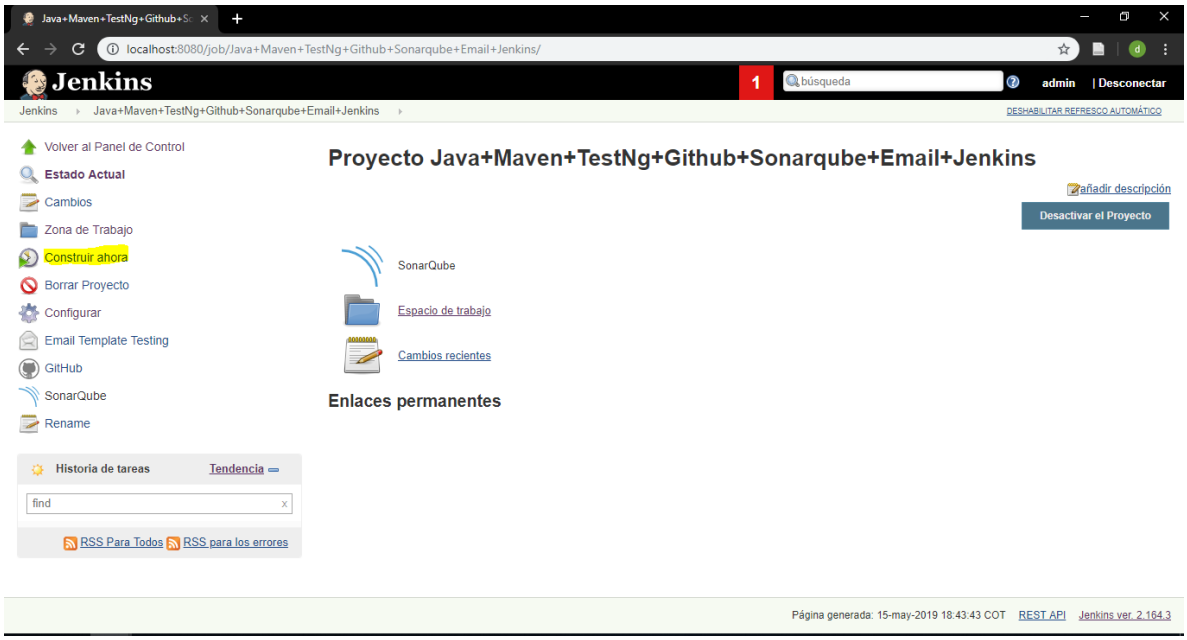
15.23. Seleccione la opción Unstable (Test Failures) y repita los pasos 13.20 y 13.21.



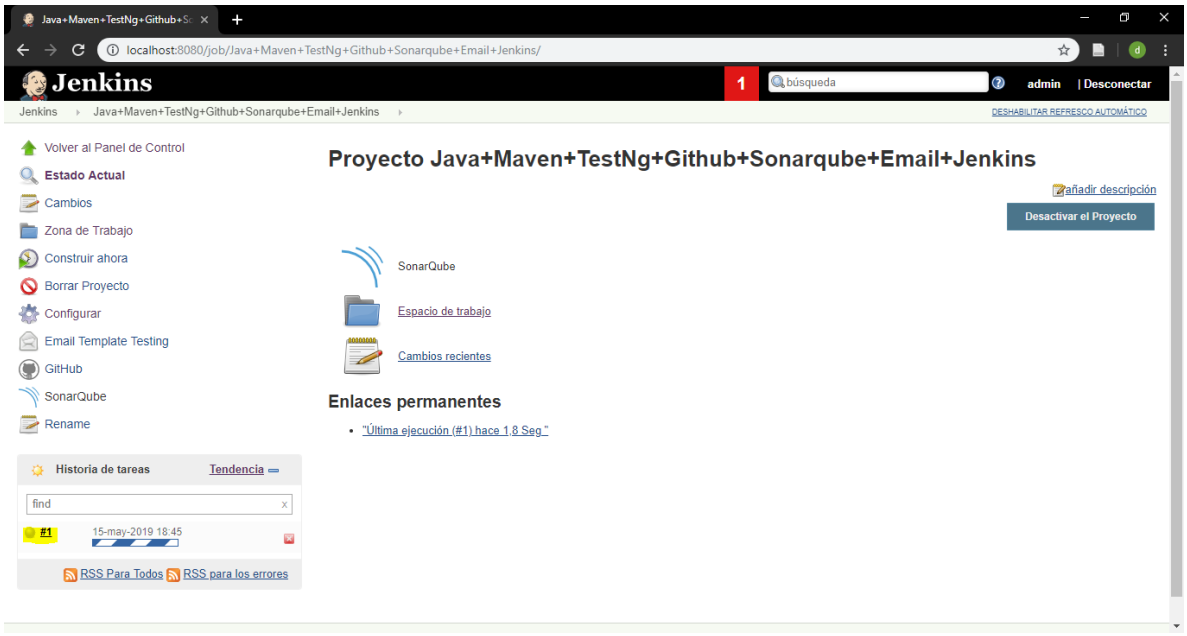
15.24. Oprima el botón Guardar



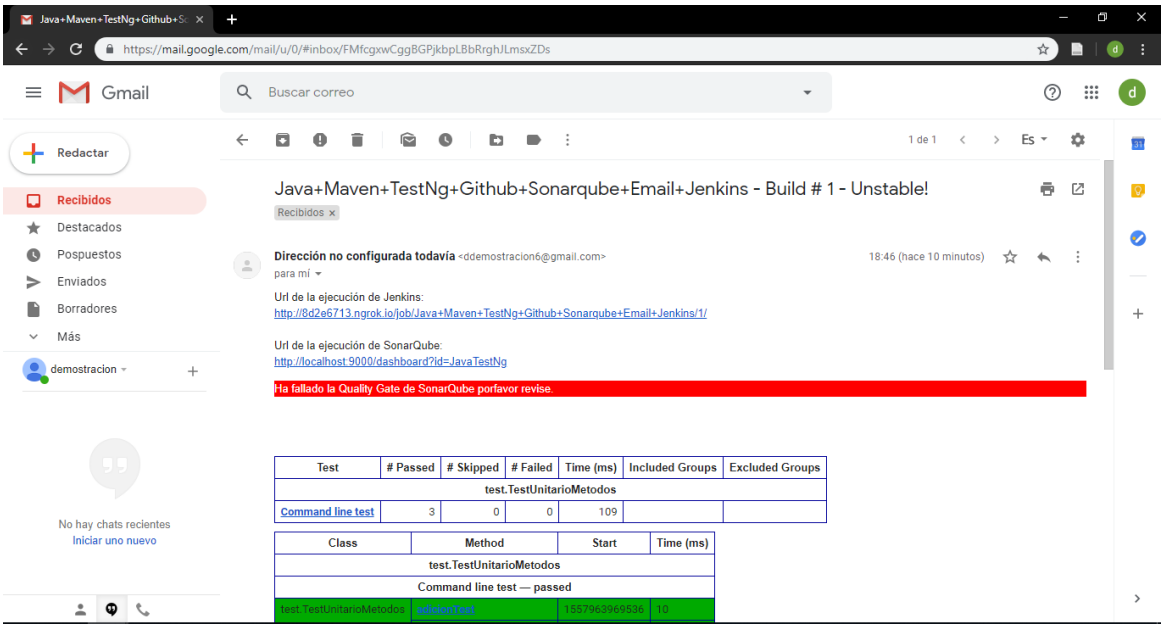
15.25. En la siguiente ventana oprima la opción “Construir ahora”



15.26. Jenkins iniciara con la ejecución.



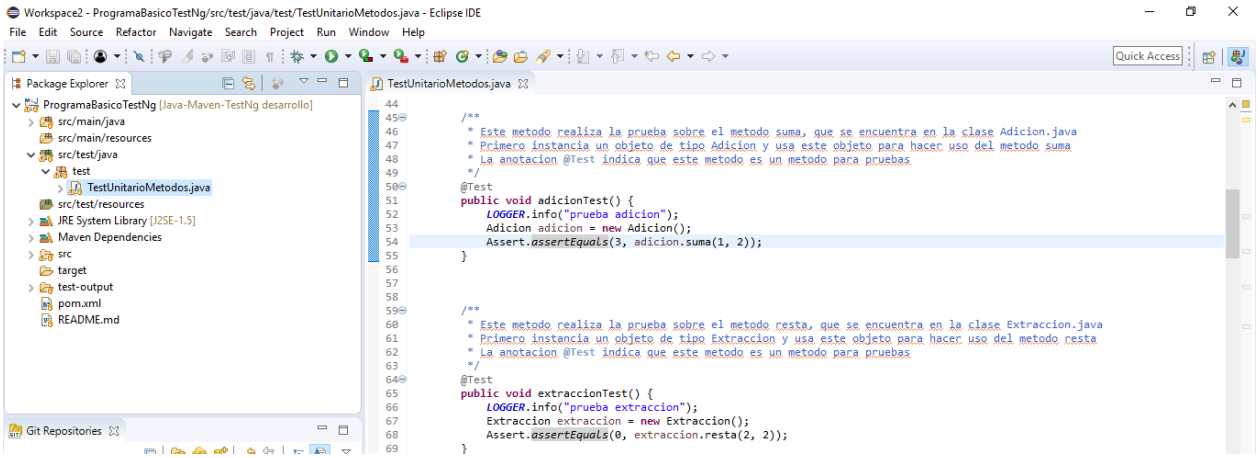
15.27. En este caso práctico la ejecución finalizó con errores, por lo que un e-mail fue enviado al correo electrónico configurado como Recipient List , el correo utilizado fue ddemostracion6@gmail.com



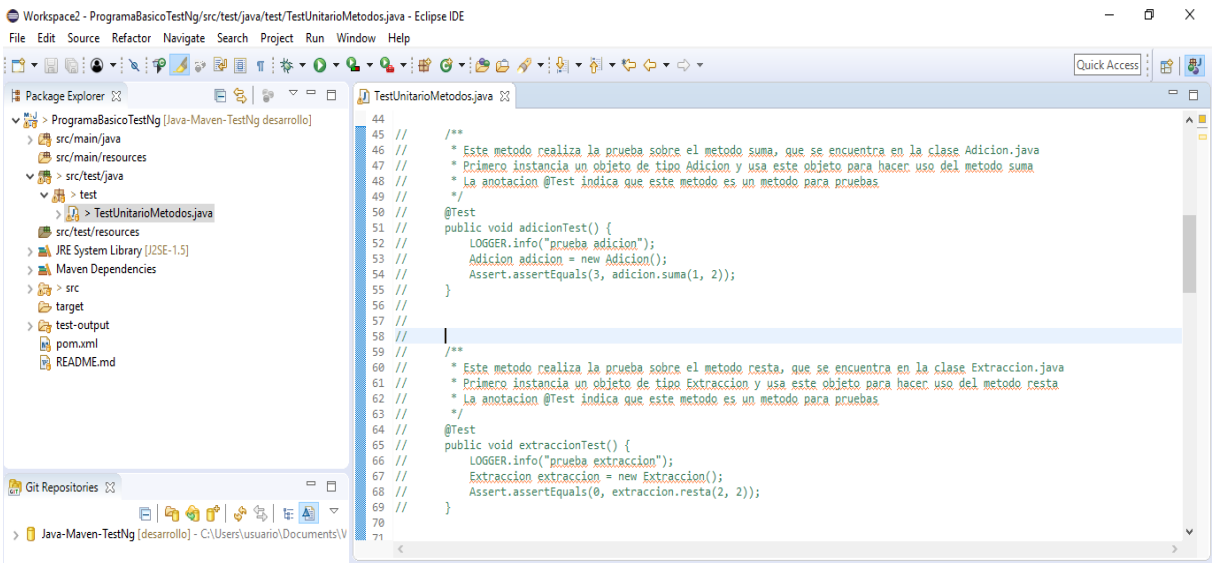
16. EJECUCIÓN AUTOMÁTICA DE JENKINS AL CREARSE UN PULL REQUEST EN GITHUB.

En esta sección se creará un pull request de la rama desarrollo hacia la rama master en GitHub, para desencadenar la ejecución automática de las pruebas unitarias y revisión de código por SonarQube por medio de Jenkins.

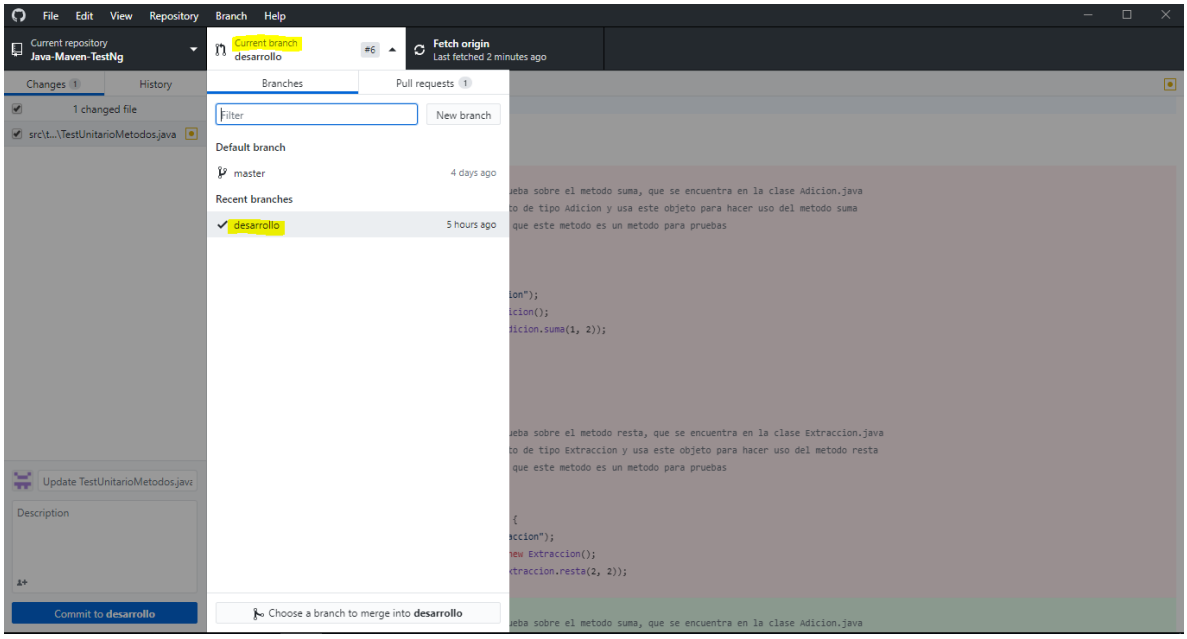
16.1. Ejecute el software Eclipse y abra el proyecto versionado de Java, realice doble click sobre la clase TestUnitarioMetodos.java.



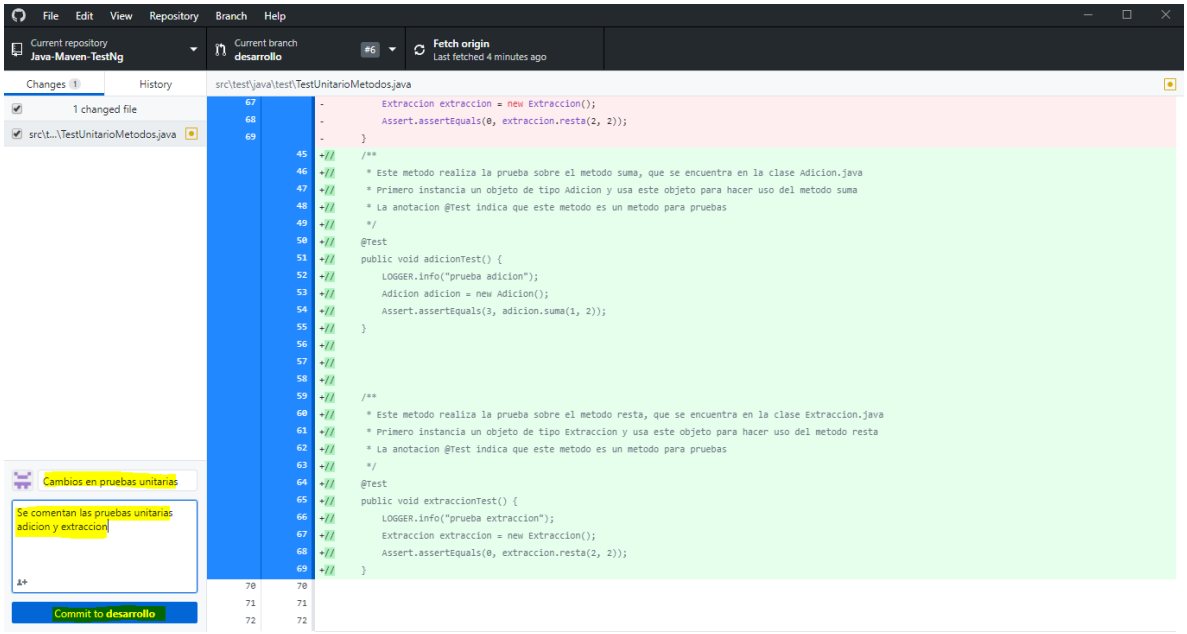
16.2. Comente el test adicionTest() y extraccionTest(), usando los símbolos // y guarde los cambios oprimiendo el botón guardar de la barra de herramientas superior o también puede hacerse oprimiendo Ctrl+s en el teclado.



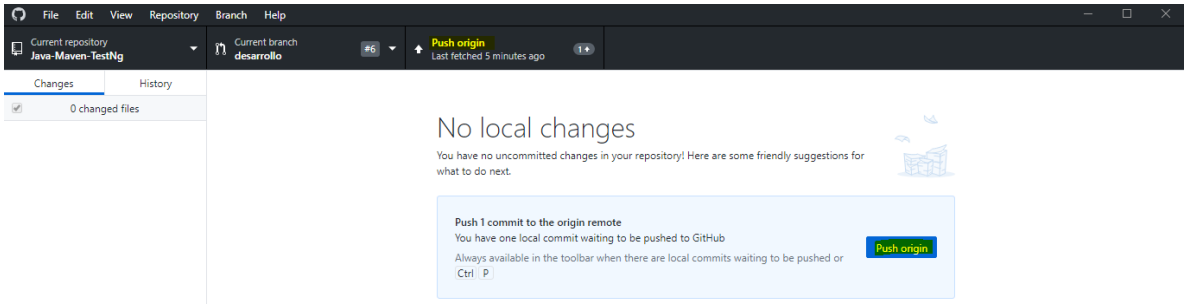
16.3. Versione el código en la rama desarrollo, para esto ejecute GitHub Desktop → click sobre la opción curren branch → click en la opción desarrollo.



16.4. Ingrese una descripción sobre los cambios y oprima el boton Commit to desarrollo

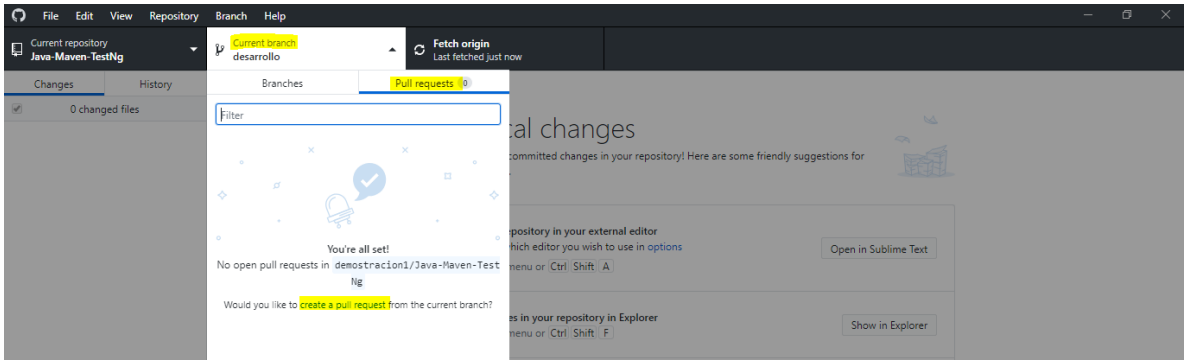


16.5. Después de hacer el commit a la rama desarrollo, oprima la opción Push origin

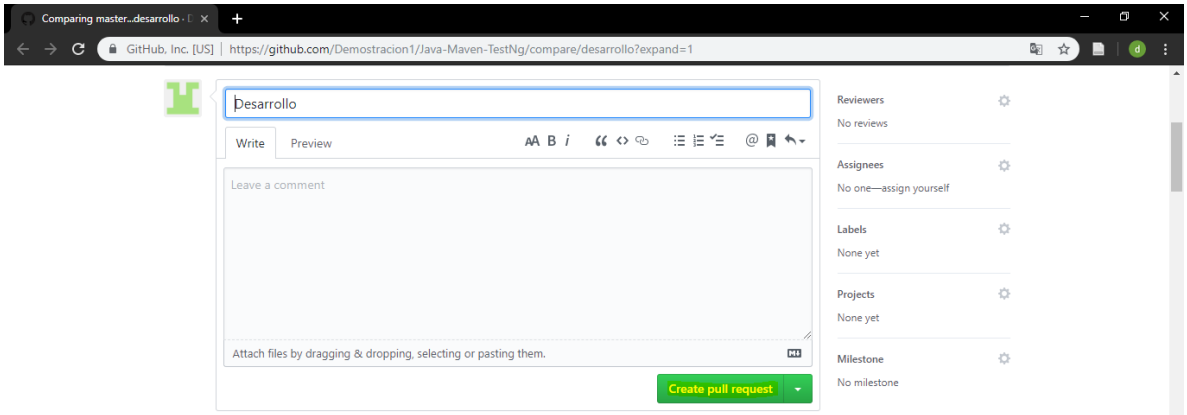


16.6. Sus cambios se encuentran versionados en el repositorio de GitHub.com

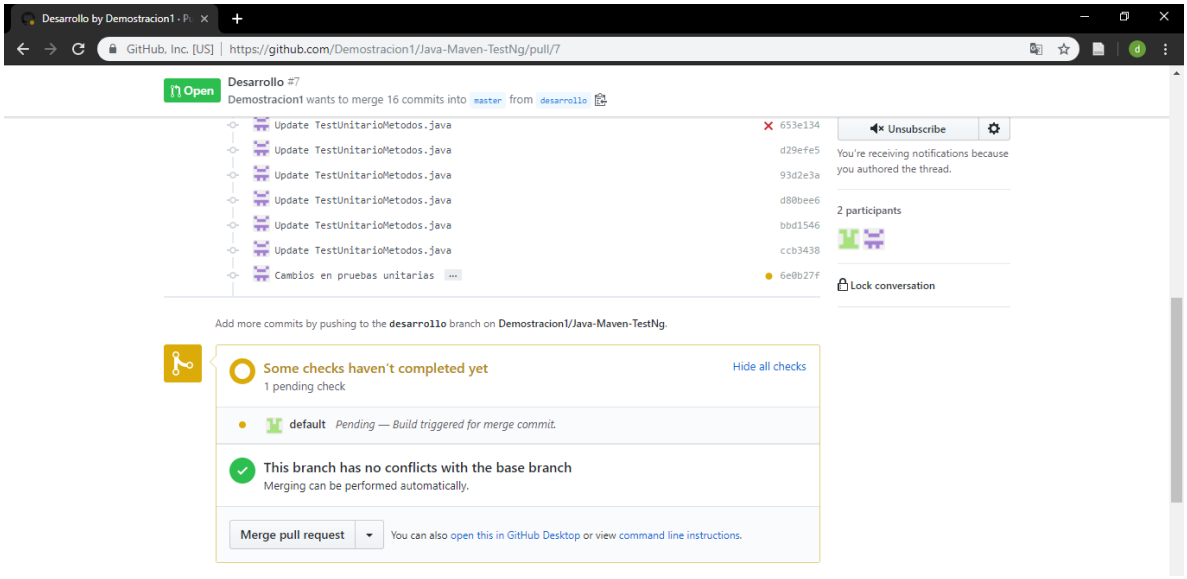
16.7. Oprima la opción Current branch→seleccione la pestaña Pull Request → seleccione la opción create a pull request



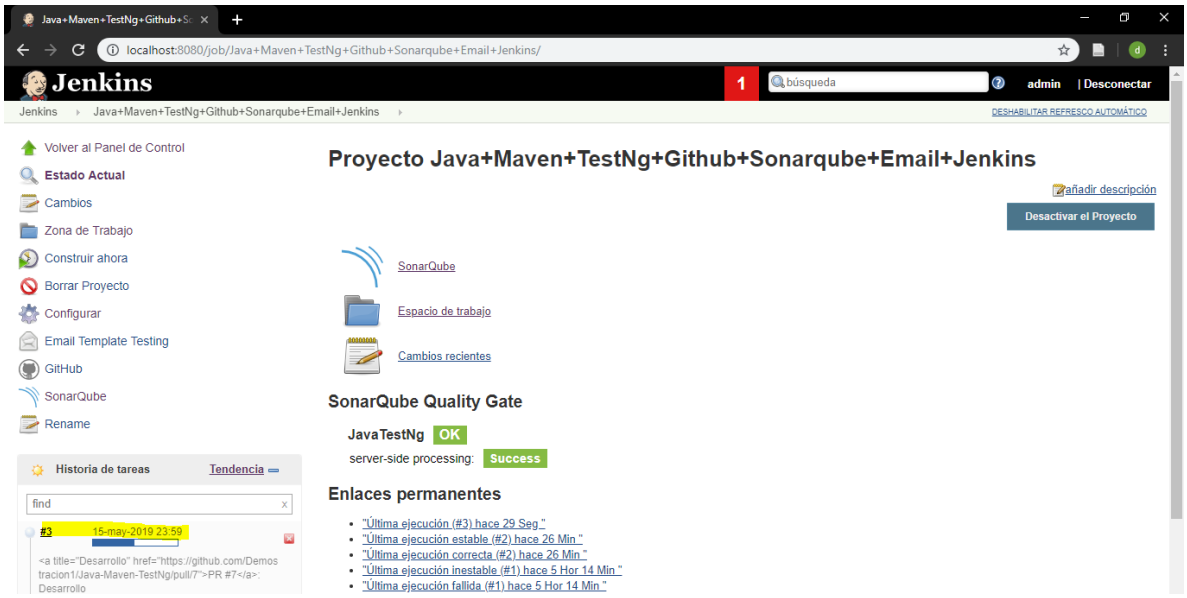
16.8. Después de oprimir la opción create a pull request, su navegador predeterminado abrirá una nueva pestaña de GitHub.com donde confirmará la creación del pull request → realice click sobre el botón Create pull request.



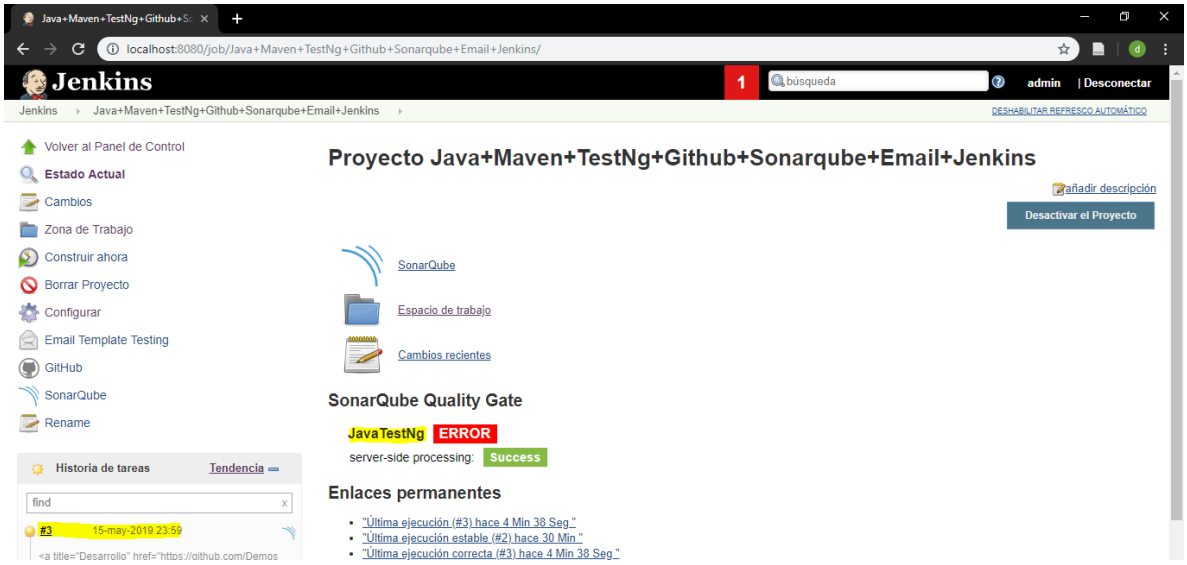
16.9. La creación del pull request de la rama desarrollo hacia la rama master, desencadena la ejecución de Jenkins y se muestra un mensaje en la ventana de GitHub.com que dice Some checks haven´t completed yet , que hace referencia a la ejecución automática de pruebas unitarias que realiza Jenkins.



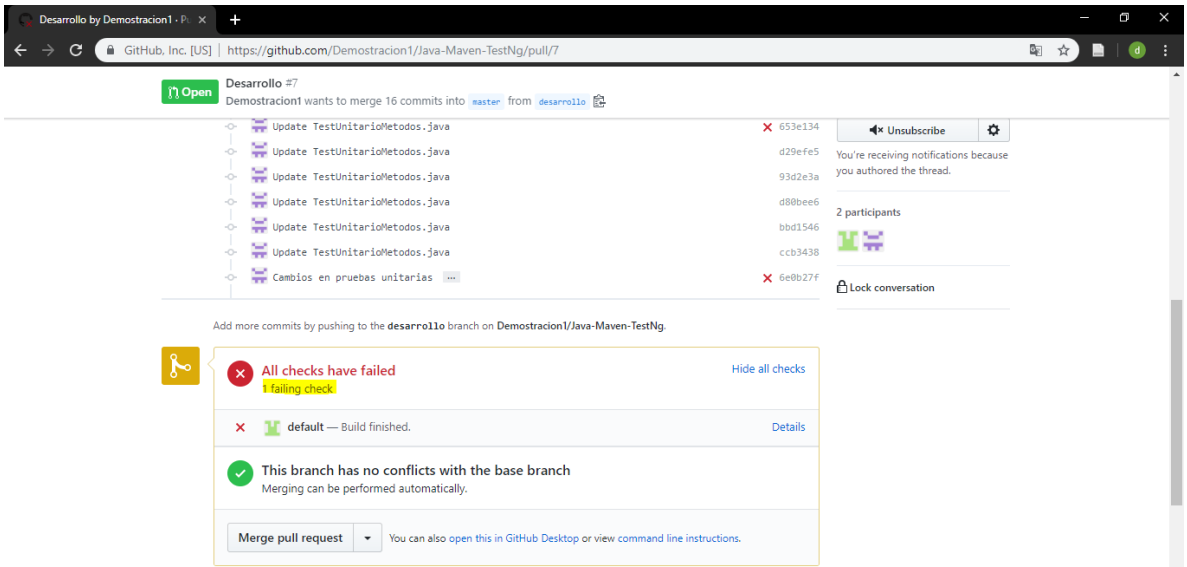
16.10. Si ingresa a Jenkins observará como una nueva tarea se está ejecutando de manera automática



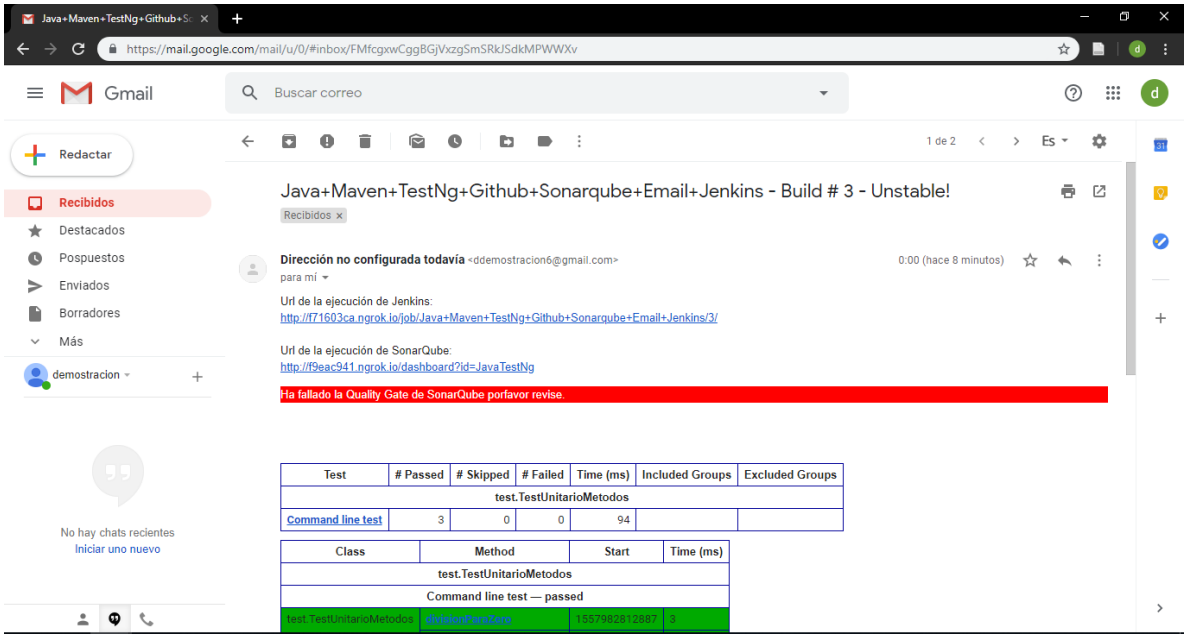
16.11. La ejecución de Jenkins finalizó con error porque no superó la Quality Gate de SonarQube, esto es por el código versionado anteriormente donde se encontraban varias pruebas unitarias comentadas.



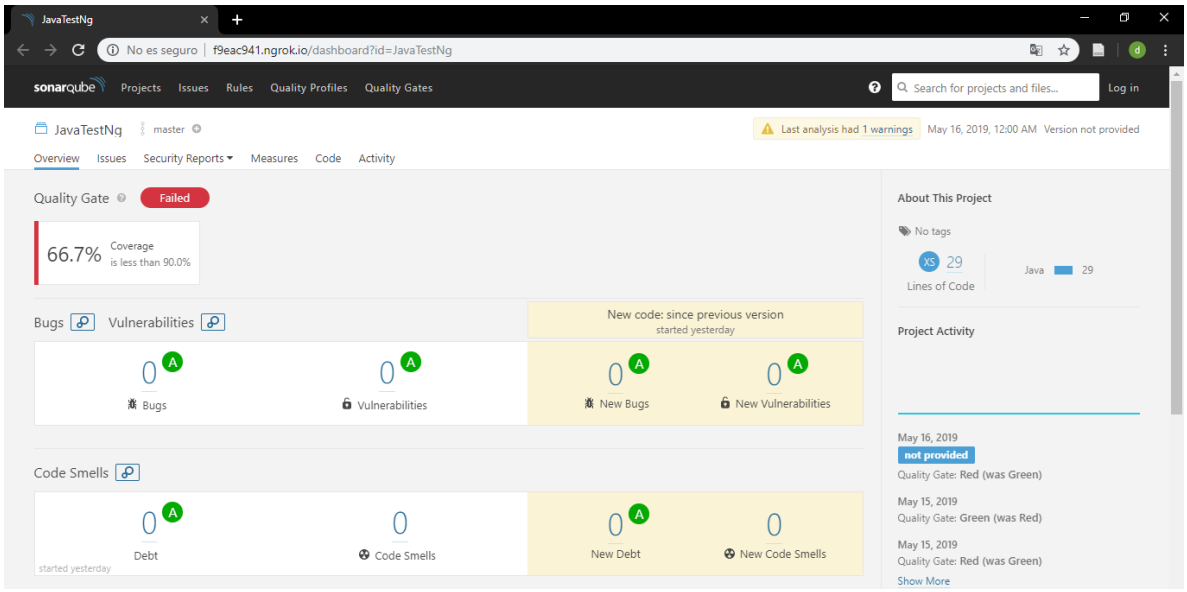
16.12. La ejecución de Jenkins finalizo, entonces el mensaje de GitHub.com se modificó y muestra si se pasaron o no todas las pruebas, en este caso la tarea de Jenkins finalizo con errores por lo cual no se pasaron todas las pruebas.



16.13. El sistema envió un correo electrónico a ddemostracion6@gmail.com indicando que la ejecución no fue exitosa.

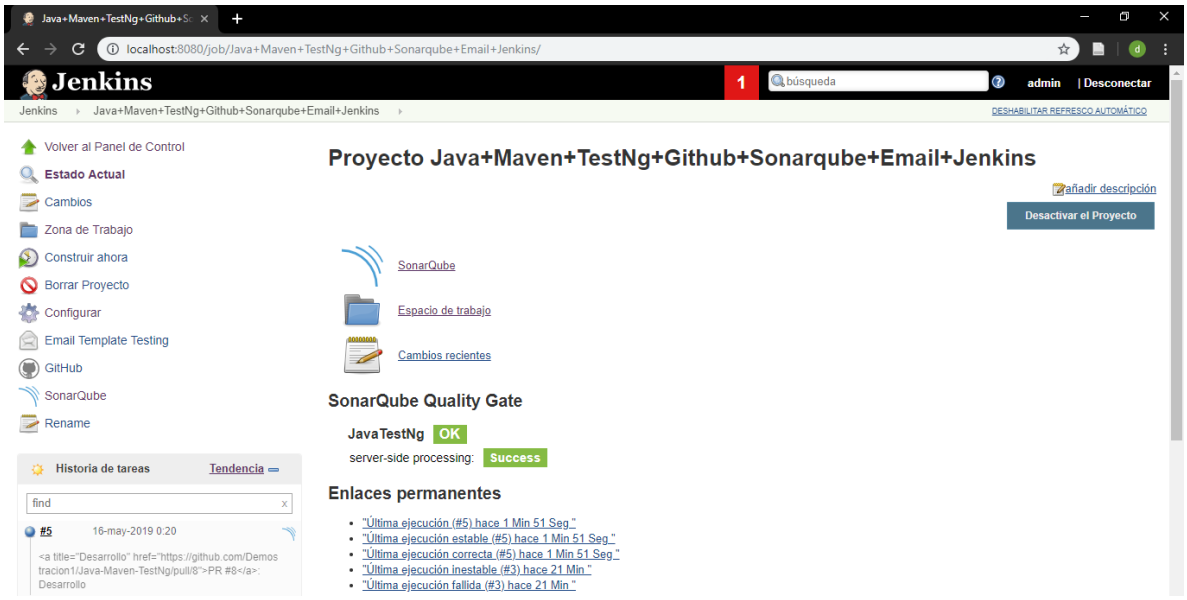


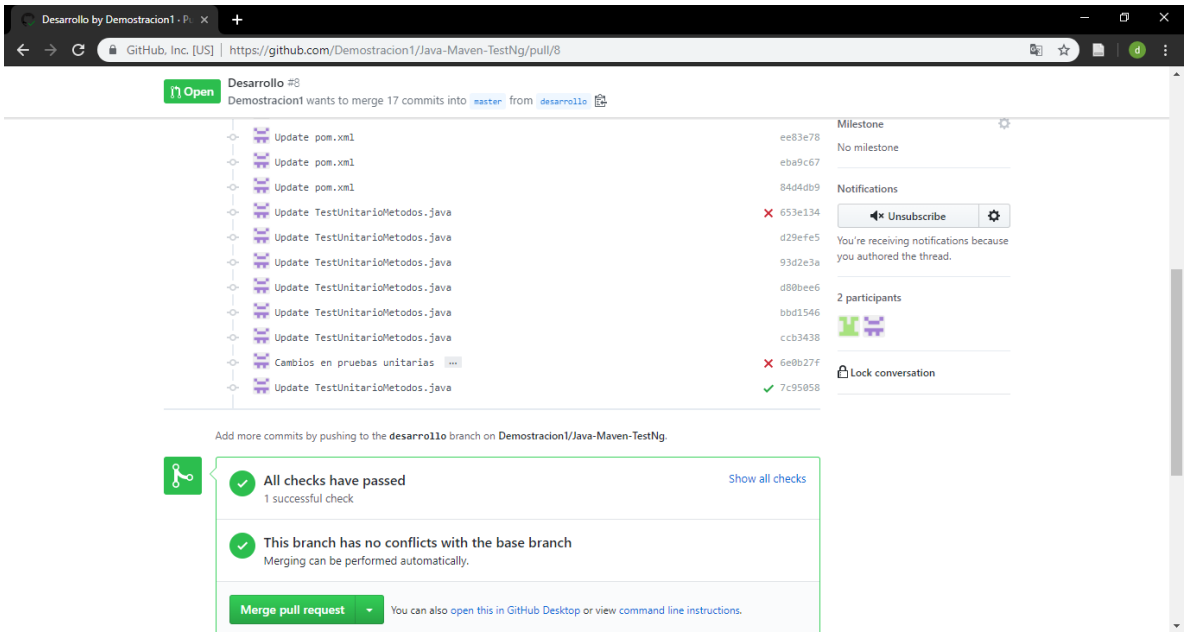
16.14. Al ingresar a SonarQube observará el porcentaje de cobertura del código por pruebas unitarias.

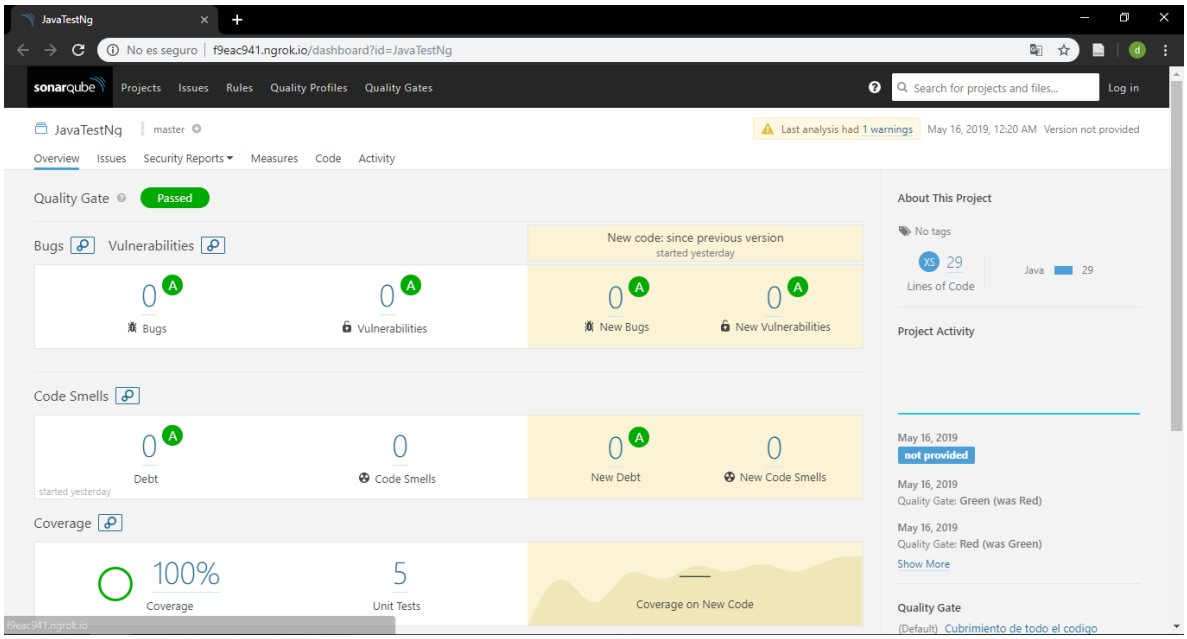


16.15. Ejecute el programa Eclipse, busque el proyecto versionado de Java y quite el comentario de las líneas anteriormente comentadas, guarde los cambios y versione nuevamente el código, cree una vez más un pull request de la rama desarrollo, puede apoyarse de los pasos 16.1 hasta 16.10.

En este caso la ejecución será exitosa y no se enviarán correos.






JavaTestNg
No es seguro | f9eac941.ngrok.io/dashboard?id=JavaTestNg
sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Search for projects and files...
Log in
JavaTestNg
master
Last analysis had 1 warnings
May 16, 2019, 12:20 AM Version not provided
Overview
Issues
Security Reports
Measures
Code
Activity
Quality Gate
Passed
Bugs
Vulnerabilities
New code: since previous version
started yesterday
0
Bugs
0
Vulnerabilities
0
New Bugs
0
New Vulnerabilities
Code Smells
0
Debt
0
Code Smells
0
New Debt
0
New Code Smells
started yesterday
Coverage
100%
Coverage
5
Unit Tests
Coverage on New Code
About This Project
No tags
XS 29
Lines of Code
Java 29
Project Activity
May 16, 2019
not provided
May 16, 2019
Quality Gate: Green (was Red)
May 16, 2019
Quality Gate: Red (was Green)
Show More
Quality Gate
(Default) Cubrimiento de todo el codigo
f9eac941.ngrok.io