Universidad de Pamplona

Facultad de Ingenierías y Arquitectura

Programa de Ingeniería de Sistemas

Tema:

DISEÑO DE PROCESO DE INTEGRACIÓN PARA LOS PROYECTOS DE DESARROLLO DE SOFTWARE DE LA EMPRESA BEGO.

Autor:

Adriana Lucia Villamizar Fuentes

Pamplona, Norte De Santander

Noviembre 2020

Universidad De Pamplona

Facultad De Ingenierías y Arquitectura

Programa De Ingeniería De Sistemas

Trabajo de grado presentado para optar al título de Ingeniera de Sistemas.

Tema:

DISEÑO DE PROCESO DE INTEGRACIÓN PARA LOS PROYECTOS DE DESARROLLO DE SOFTWARE DE LA EMPRESA BEGO.

Autor:

Adriana Lucia Villamizar Fuentes

Director:

William Mauricio Rojas Contreras

Magister en ciencias computacionales.

Pamplona, Norte de Santander.

Noviembre 2020.

Resumen

La integración continua es una práctica de desarrollo de software mediante la cual los desarrolladores combinan los cambios en el código en un repositorio central de forma periódica, tras lo cual se ejecutan versiones y pruebas automáticas. Al implementar esta práctica le permitirá a la empresa mejorar la calidad de los productos al liberar a los desarrolladores de las tareas manuales de integración además de que fomenta la disciplina en el proceso de desarrollo de software lo cual ayuda a reducir la cantidad de errores y bugs enviados a los clientes.

En el presente documento se expone el diseño de proceso de integración que permitirán a la empresa BeGo la optimización de recursos y tiempos, logrando así tener control de las versiones de desarrollo de software que se están llevando a cabo. Inicialmente se plasma el estado del arte para el proceso de integración continua de empresas dedicadas al desarrollo de Software con el fin de tener una base sólida para determinar qué herramientas y técnicas de integración se adaptan a los procesos operativos de la empresa, para su posterior selección y definición del proceso. También describe el estudio realizado para la construcción de un prototipo de búsqueda basado en IA (Inteligencia Artificial) empleando el proceso definido.

Abstract

Continuous integration is a software development practice whereby developers combine changes to code in a central repository on a regular basis, after which automatic versions and tests are run. By implementing this practice, it will allow the company to improve the quality of the products by freeing developers from manual integration tasks, as well as fostering discipline in the software development process which helps reduce the amount of errors and bugs. sent to customers. This document presents the design of integration policies that will allow the BeGo company to optimize resources and times, thus achieving control of the software development versions that are being carried out. Initially, the state of the art is reflected for the continuous integration process of companies dedicated to the development of Software in order to have a solid base to determine which integration tools and techniques are adapted to the operational processes of the company, for their subsequent selection and definition of the policy. It also describes the study conducted for the construction of a search prototype based on AI (Artificial Intelligence) using the defined policy.

Tabla de contenido

Tablas

Tabla de figuras

1 Descripción del proyecto

1.1 Planteamiento del problema

En la actualidad las empresas dedicadas a la producción de software deberían tener establecido un proceso de integración que permitan optimizar de forma planeada y disciplinada el proceso de la estructuración y división del ambiente de desarrollo, así como los flujos de trabajo y el versionamiento de proyectos.

La empresa BeGo aborda los requerimientos de desarrollo mediante la asignación de tareas y responsabilidades o en ocasiones por medio de descripciones realizadas de manera informal en donde se especifican de forma breve los comportamientos que se deben esperar para cumplir con las exigencias propuestas. El control actual de versiones del proyecto se encuentra organizado en un repositorio de GitHub que está segmentado en dos ramas, en la primera rama “Master” se almacena la lógica del negocio y el acceso a la base de datos y la segunda rama “PublicHtml” donde se hospeda la plataforma web y la aplicación móvil. Por lo general se hacen actualizaciones al repositorio una vez al día después de culminar la jornada de trabajo, esta actualización se hace independiente del estado y funcionamiento actual del código.

Además, esta plataforma cuenta con una barra de búsqueda la cual funciona de una manera poco eficiente, dejando a un lado una gran gama de variedad de sus productos por fuera a la hora que se efectúan las búsquedas.

Al no tener ningún proceso establecida se ve perjudicada la parte organizacional debido a que se reprocesa el análisis causando que los tiempos de la implementación de las tareas pueden aumentar más de lo esperado provocando una falta de optimización en los proyectos. Es por esto que es necesario establecer una guía que permita a la empresa BeGo llevar un mayor control y orden

sobre las tareas que realizan los desarrolladores permitiendo así ahorrar tiempo y recursos. Así mismo al proporcionar una barra de búsqueda le permitirá al usuario alcanzar con éxito su objetivo de una forma rápida y directa, haciendo la experiencia de compra más cómoda.

1.2 Justificación

El diseño de proceso de integración le permitirá a la empresa BeGo llevar un control de las versiones que se están trabajando, permitiendo identificar de manera más fácil donde se encuentra un error en el dado caso que se presenten, asimismo le aporta calidad y estabilidad al proyecto en general. Establecer un proceso de integración es una necesidad de la empresa BeGo y le permitirá perfeccionar la fase de desarrollo de su plataforma, así como también proporcionar estándares de control, planeación y protocolos de funcionamiento. Este proyecto busca lograr definir un modelo para la integración de software que mejore el rendimiento para el equipo de desarrollo, puesto que no sería necesario la inversión de gran cantidad de tiempo en realizar manualmente pruebas al sistema. Además, al validar dicho proceso con el sistema de búsqueda inteligente le permitirá al usuario tener una experiencia más amigable con la plataforma, ayudándolos a adquirir los productos de una manera rápida y directa.

1.3 Delimitaciones

- OBJETIVO GENERAL
  - Diseñar un proceso de integración para los proyectos de desarrollo de software de la empresa BeGo.
- OBJETIVOS ESPECÍFICOS
  - Establecer el estado del arte del proceso de integración continua en empresas dedicadas al desarrollo de Software.
  - Analizar las herramientas y técnicas para implementar el proceso de integración continua en la empresa BeGo.
  - Diseñar el proceso de integración continua para la empresa BeGo.
  - Validar el proceso de integración continua mediante el diseño de un prototipo de búsqueda basado en IA para los proyectos desarrollados en la empresa BeGo.

1.4 Acotaciones

En el presente trabajo se emplearán únicamente herramientas de integración de código libre (Open Source), así mismo cabe aclarar que la información, descripción, demostraciones funcionales o resultados serán limitadas bajo los acuerdos de confidencialidad de la empresa donde se realiza la práctica, el proceso de integración sólo será aplicada al prototipo de búsqueda con inteligencia artificial.

1.5 Metodología

La metodología que se escogió usar para este trabajo fue la investigación orientada a decisiones, ya que no se centra en hacer aportes teóricos si no que se busca soluciones al problema que se plantea en el documento

El procedimiento que se realizará consiste inicialmente el realizar una investigación con la cual se obtendrán las suficientes bases teóricas de las herramientas que se pueden llegar a utilizar en el proyecto. Una vez se cuente con la suficiente información pertinente se realizará un análisis comparativo para poder determinar cuáles de ellas se adaptan de una mejor manera a los procesos operativos que se llevan a cabo en la empresa BeGo y la última fase se llevará a cabo la implementación de las técnicas y herramientas que aporten mayor beneficio a los proyectos de desarrollo de la empresa.

Para este proyecto se recolectará información de guías, tutoriales, artículos, proyectos, que estén relacionados con la integración continúa implementadas a pequeñas o medianas empresas.

2 Marco teórico y estado del arte

2.1 Marco conceptual

2.1.1 Integración Continua.

La integración continua es una práctica de desarrollo de software mediante la cual los desarrolladores combinan los cambios en el código en un repositorio central de forma periódica, tras lo cual se ejecutan versiones y pruebas automáticas. La integración continua se refiere en su mayoría a la fase de creación o integración del proceso de publicación de software y conlleva un componente de automatización (p. ej., CI o servicio de versiones) y un componente cultural (p. ej., aprender a integrar con frecuencia). Los objetivos clave de la integración continua consisten en encontrar y arreglar errores con mayor rapidez, mejorar la calidad del software y reducir el tiempo que se tarda en validar y publicar nuevas actualizaciones de software.(*Integración Continua Del Software | Pruebas Automatizadas | AWS*, n.d.)

Martin Flower (2015), asegura que tener el desarrollo como un proceso disciplinado y automatizado, es esencial para un proyecto controlado, cuando esto se logra el equipo está más preparado para modificar el código cuando sea necesario, debido a la confianza en la identificación de los errores en la integración; una vez se realiza una integración los errores son detectados de inmediato, permitiendo identificar más fácilmente de donde proviene el error, reduciendo el tiempo que se toman los equipos para identificar errores cuando la compilación se hace posterior varias semanas o meses de desarrollo.(ANA MARÍA GARCÍA OROZCO, 2015)

Anteriormente, era común que los desarrolladores de un equipo trabajasen aislados durante un largo periodo de tiempo y solo intentasen combinar los cambios en la versión maestra una vez que habían completado el trabajo. Como consecuencia, la combinación de los cambios en el código

resultaba difícil y ardua, además de dar lugar a la acumulación de errores durante mucho tiempo que no se corregía. Estos factores hacían que resultase más difícil proporcionar las actualizaciones a los clientes con rapidez.

Con la integración continua, los desarrolladores envían los cambios de forma periódica a un repositorio compartido con un sistema de control de versiones como Git. Antes de cada envío, los desarrolladores pueden elegir ejecutar pruebas de unidad local en el código como medida de verificación adicional antes de la integración. Un servicio de integración continua crea y ejecuta automáticamente pruebas de unidad en los nuevos cambios realizados en el código para identificar inmediatamente cualquier error. (*Integración Continua Del Software | Pruebas Automatizadas | AWS*, n.d.)

La Integración Continua es una práctica que se aplica en todo tipo de entornos de desarrollo de software en la cual, para automatizar el proceso de construcción, pruebas, despliegue y para que se realice diariamente es necesario:

- Escoger un repositorio para almacenar las fuentes, donde se lleve el historial de todos los cambios del código y de donde sea posible obtener la última versión del proyecto.
- Automatizar el proceso de construcción de manera que a partir de las fuentes se pueda construir con un único comando todo el sistema.
- Realizar las pruebas de forma automática y de esta manera saber si todo está correcto o si existe algún problema.

Esta práctica consiste en llevar al límite el proceso de construcción automática y diaria. Si se logra automatizar este proceso, el modelo de integración continua debe estar bien diseñado, porque más que ventajas creará grandes problemas.

2.1.1.1 Ventajas de Integración Continua.

Según Damián Cervantes en su propuesta de entorno de integración continua en el centro de informatización universitaria, la integración continua es una práctica que no es muy fácil de aplicar pero su implementación trae consigo numerosas ventajas(Damián Cervantes Rodón, 2010):

- Disminuye la búsqueda de fallos a la hora de integrar el código.
- Permite identificar fallos en el entorno de producción en etapas tempranas.
- Disminuye el tiempo de retroalimentación de errores con el cliente.
- Aumenta la confianza de los desarrolladores al subir el código al control de versiones.
- Constante disponibilidad de las compilaciones realizadas.

2.1.1.2 Desventajas de Integración Continua.

De acuerdo con Damián Cervantes la implantación de la Integración Continua también trae consigo algunas desventajas que pueden relacionarse de la siguiente manera (Damián Cervantes Rodón, 2010):

- Sobrecarga por el mantenimiento del sistema.
- Necesidad potencial de un servidor dedicado a compilar.

- El impacto inmediato al subir código erróneo provoca que los desarrolladores no suban su código frecuentemente como sería conveniente como copia de seguridad.
- Implica introducir una nueva filosofía de desarrollo.

2.1.1.3 Control de versiones.

En el escrito hecho por Rosa Durante y Leonardo Pastrana declaran que los sistemas de control de versiones utilizan para su funcionamiento un repositorio central que es el lugar donde se almacenan los datos y la información.

Existen tres operaciones básicas en un control de versiones: bajar la versión completa del código, actualizar cambios y subir cambios. (Rosa MaDurante Lerate et al., n.d.)

El control de versiones es una práctica importante para cualquier grupo de desarrollo de software. Existen varias herramientas para el control del código fuente, pero sin importar el que se utilice, al menos deben cumplir con algunas de las siguientes características básicas:

- Proporcionar un lugar para almacenar el código fuente.
- Proveer un historial de lo que se ha hecho a lo largo del tiempo.
- Permitir el trabajo en paralelo de los desarrolladores, uniendo los esfuerzos más tarde.
- Proveer una manera de que los desarrolladores trabajen juntos sin interponerse en el camino del otro.

2.1.1.3.1 Control de versiones de sistemas centralizados.

Presentan la característica fundamental de que funcionan como un entorno clásico Cliente-Servidor. Es decir, se tiene un servidor en el que se aloja el repositorio del proyecto, con toda la información de los cambios, ficheros binarios añadidos. En estos sistemas, el cliente trabaja con

una “copia de trabajo” del servidor, la cual es realmente una copia de cómo estaba el servidor en una revisión determinada - normalmente es la más actualizada. El desarrollador hace cambios sobre esa copia de trabajo, y cuando considera que ha terminado con esa modificación la sube (commit) al servidor, el cual se encargará de fundir esos cambios en el repositorio central, resolver conflictos si pudiera, o informar al usuario de los errores que se hayan podido dar.(Rosa MaDurante Lerate et al., n.d.)

2.1.1.3.2 Control de versiones de sistemas distribuidos.

Si los sistemas centralizados utilizan un modelo clásico de entorno cliente-servidor, se podría decir que un sistema distribuido es similar a un sistema Peer-ti-Peer (P2P). En estos sistemas, en lugar de que cada cliente tenga una copia de trabajo del (único) servidor, la copia de trabajo de cada cliente es un repositorio en sí mismo, una rama nueva del proyecto central. De esta forma, la sincronización de las distintas ramas se realiza intercambiando “parches” con otros clientes del proyecto. Esto es claramente un enfoque muy diferente al de los sistemas centralizados, por diversos motivos:

- No hay una copia original del código del proyecto, solo existen las distintas copias de trabajo.
- Operaciones como los commits, mirar el historial o rehacer cambios, no necesitan de una conexión con un servidor central, esta conexión solo es necesaria al “compartir” tu rama con otro cliente del sistema.
- Cada copia de trabajo es una copia remota del código fuente y de la historia de cambios, dando una seguridad muy natural contra la pérdida de los datos.(Rosa MaDurante Lerate et al., n.d.)

2.1.1.4 Herramientas de integración continua.

2.1.1.4.1 Jenkins.

Es probablemente una de las herramientas de integración continua más conocidas del mercado. Este software escrito en Java ha continuado desarrollándose constantemente desde el año 2005 (entonces, bajo el nombre de Hudson) y cuenta en la actualidad con numerosas funciones que asisten no solo en la integración continua, sino también en el despliegue y la entrega continua.(*Las Mejores Herramientas de Integración Continua - IONOS*, 2019)

2.1.1.4.2 Travis CI.

Esta herramienta de integración continua trabaja en estrecha relación con el popular software de control de versiones GitHub. Esta herramienta puede configurarse con un sencillo archivo YAML que se guarda en el directorio raíz del proyecto. GitHub informa a Travis CI de todos los cambios efectuados en el repositorio y mantiene el proyecto actualizado. (*Las Mejores Herramientas de Integración Continua - IONOS*, 2019)

2.1.1.4.3 Bamboo.

La compañía Atlassian, que gestiona también el servicio de alojamiento de archivos Bitbucket, ofrece desde el año 2007 la herramienta de integración continua Bamboo. Esta herramienta no solo sirve de ayuda en la integración continua, sino también para funciones de despliegue y gestión de lanzamientos. Funciona a través de una interfaz web. (*Las Mejores Herramientas de Integración Continua - IONOS*, 2019)

2.1.1.4.4 GitLab CI.

GitLab CI forma parte del conocido sistema de control de versiones GitLab. Además de integración continua, GitLab ofrece despliegue y entrega continua. Al igual que con Travis CI, la configuración de GitLab CI se lleva a cabo con un archivo YAML. Por lo demás, su utilización es sencilla. (*Las Mejores Herramientas de Integración Continua - IONOS*, 2019)

2.1.1.4.5 Circle CI.

La herramienta de integración continua CircleCI funciona tanto con GitHub como con Bitbucket. En las fases de prueba, pueden emplearse tanto contenedores como máquinas virtuales. CircleCI confiere mucha importancia a la ejecución de procesos de desarrollo sin interferencias, por lo que arroja de forma automática builds compatibles con otros entornos. (*Las Mejores Herramientas de Integración Continua - IONOS*, 2019)

2.1.1.4.6 Cruise Control.

Cruise Control se encuentra entre las aplicaciones más antiguas de integración continua. La herramienta se lanzó al mercado en 2001 y ha continuado desarrollándose desde entonces —entre otros, por Martin Fowler, pionero en el ámbito de la integración continua—. Junto con un claro cuadro de mandos, los desarrolladores tienen a su disposición numerosos plugins que les facilitarán el trabajo. (*Las Mejores Herramientas de Integración Continua - IONOS*, 2019)

2.1.1.4.7 CodeShip

La herramienta de integración continua Codeship pertenece a CloudBee, que también cuenta con Jenkins en su catálogo. El programa está disponible en dos versiones: La versión básica, con una interfaz web sencilla, y la versión profesional, configurada con archivos en el repositorio. Aquellos que deseen trabajar con un contenedor Docker, tendrán que hacerse con la versión profesional. (*Las Mejores Herramientas de Integración Continua - IONOS*, 2019)

2.1.1.4.8 TeamCity

El software TeamCity destaca sobre todo por sus “gated commits”. Con ellos, la herramienta comprueba los cambios en el código antes de integrarlos a la línea principal. Únicamente cuando el código está libre de errores, pasa a formar parte del código base para todo el equipo. TeamCity lleva a cabo las pruebas automáticamente en un segundo plano, de modo que el desarrollador puede continuar trabajando. (*Las Mejores Herramientas de Integración Continua - IONOS*, 2019)

2.1.1.4.9 Git Actions

GitHub Actions permite crear flujos de trabajo (**workflows**) que se pueden utilizar para compilar, probar y desplegar código, dando la posibilidad de crear flujos de integración y despliegue continuo dentro del propio repositorio de git.

Los flujos de trabajo tienen que contener al menos un **job**. Estos incluyen una serie de pasos que ejecutan tareas individuales que pueden ser acciones o comandos. Un flujo de trabajo puede comenzar por distintos eventos que suceden dentro de GitHub, como un **push** al repositorio o un **pull request**.(Claudia Márquez, 2020)

Nombre	**Logo**	**Entrega Continua**	**Alojamiento en la nueve**	**Licencia**	**Precio versión Comercial**	**Versión Gratuita**	**Particularidades**
Jenkins		✓	✓	MIT	-	✓	Numerosos plugins

Travis CI	Travis CI	✘	✓	MIT	69-498/mes	✓	Conexión directa con GitHub
Bamboo	Bamboo	✓	✓	De propietario	10-126500/mes	✓	
GitLab CI	CI/CD	✓	✓	MIT/EE	4-99/Mes	✓	Conexión directa con otros productos de Atlassian
CircleCI	circleci	✓	✓	De propietario	50-3150/mes	✓	Fácil de utilizar
Cruise Control	cruisecontrol.	✘	✘	BSD	-	✓	Completamente gratuita
CodeShip	CODESHIP	✓	✓	De propietario	75-1500/mes	✓	Versión profesional y básica
TeamCity	TC	✓	✘	De propietario	299-21999 pago único	✓	Gated Commits

GitAction		✓	✓	MIT	-	✓	Conexión directa con GitHub

Tabla 1 Comparación de herramientas de integración continua (*Las Mejores Herramientas de Integración Continua - IONOS*, 2019)

2.1.2 Buscadores Inteligentes

un buscador inteligente sabe ofrecer respuestas diferentes. La inteligencia consiste en ser capaz de adaptar los resultados en función de qué persona, desde qué lugar o en qué momento realiza la búsqueda.

La inteligencia de un buscador permite ganar eficacia y eficiencia en la gestión del conocimiento (*Soluciones: Buscadores Inteligentes: 3.14*, n.d.)

Según el artículo realizado por digital guide Ionos toda página web comercial debería incluir una barra de búsqueda, sobre todo si se trata de una tienda online compuesta por gran número de artículos, pues así contribuirá positivamente a la satisfacción del cliente. Un estudio de la Universidad de Regensburg, solo disponible en alemán, aclara su importancia a través de datos concisos: el 80 % de las empresas **considera la barra de búsqueda esencial en su web**, motivo por el cual suelen incluirla y solo el 3 % de los encuestados no disponían en sus webs de estos buscadores.

A menudo el 60 % usa **módulos adicionales de terceros**. El 23 % utiliza la búsqueda estándar del sistema de la tienda y por lo menos el 8 % recurre a una solución desarrollada por ellos mismos. No obstante, las empresas que usan en sus webs funciones de búsqueda inteligente muestran por norma general un mayor grado de satisfacción que las que usan la búsqueda estándar. (*La Importancia de Una Barra de Búsqueda Inteligente - IONOS*, 2017)

2.1.2.1 *Importancia de una barra de búsqueda inteligente*

Si los visitantes de una web no saben qué están buscando, es más útil navegar por categorías, obviando así la barra de búsqueda. No obstante, por regla general, los consumidores saben con cierta exactitud lo que quieren o al menos conocen la marca o el tipo de producto que pretenden adquirir, por lo que encuentran en la barra de búsqueda de las webs una herramienta importante para poder **alcanzar con éxito su objetivo de forma rápida y directa**. Para cumplir con estas demandas en los sistemas de tiendas, primero es necesario **incluir la búsqueda inteligente sin olvidar su posterior y continua optimización**. La búsqueda estándar que integran originalmente los sistemas de gestión de tiendas y contenidos genera normalmente pocas ventas, ya que presenta una menor tolerancia de errores, por ejemplo, no reconoce como iguales a una palabra en singular o en plural e ignora el uso de sinónimos para denominar un determinado producto. Además, en torno a la mitad de los vendedores online no están satisfechos con el hecho de que la búsqueda estándar no ofrece ningún instrumento para fomentar las ventas.

- Los clientes encuentran el producto deseado más rápidamente y con mayor exactitud.
- Esto mejora la usabilidad.
- Normalmente la tasa de conversión aumenta al usar un sistema de búsqueda inteligente.
- Los tres factores anteriores elevan la fidelización de los clientes.

2.1.2.2 *Resultados de una barra de búsqueda inteligente*

Digital guide Ionos no dice que la respuesta debería tener en cuenta lo que realmente debe ofrecer una **búsqueda adaptada a los usuarios** para conseguir el efecto deseado y aumentar la satisfacción del cliente. Las principales características y líneas generales son:

- **Tolerancia de errores de la barra de búsqueda:** Al buscar un producto es normal equivocarse al escribir una palabra, cambiar el orden de las letras o de los números del nombre inconscientemente o errar al no conocerse con exactitud su denominación. No obstante, si a pesar de estos errores un usuario puede encontrar lo que busca, su experiencia será sin duda satisfactoria.
- **Reconocimiento de singular, plural y sinónimos:** El hecho de que un cliente escriba una palabra concreta en singular, mientras que otro introduce la misma en plural, no debe ser motivo para que los resultados de búsqueda difieran entre ambos. No se puede pretender que todos los clientes tengan el mismo vocabulario o utilicen exactamente el mismo término para referirse a un producto concreto, teniendo en cuenta además las variedades dialectales de un idioma: aunque en el español de España se suele utilizar la palabra ordenador, en ciertas zonas de Latinoamérica usan el término computadora. Con todo, se pretende señalar que si la búsqueda no reconoce la presencia de sinónimos o de singulares y plurales es posible que se pierdan resultados de búsqueda significativos.
- **Función de autocompletar:** Se pueden evitar los errores de escritura si la función de búsqueda reconoce y sugiere directamente los términos de búsqueda esperados. A esta función de búsqueda inteligente se la conoce como autocompletar y, como su nombre indica, completa la información de los términos introducidos en una lista bajo la barra de búsqueda o propone términos que podrían coincidir.
- **Mas Opciones de búsqueda:** Hay muchas más posibilidades para encontrar artículos determinados, ya que a menudo se insertan en el buscador datos incompletos sobre el producto o su denominación exacta. Un buen ejemplo de ello son los libros. Estos se

pueden encontrar por autor, título o número del ISBN; aunque la época, la corriente artística o el género pueden constituir términos de búsqueda relevantes.

- **Filtros y las categorías de una función de búsqueda inteligente:** a búsqueda inteligente permite buscar en categorías específicas y de este modo limitar los resultados. Con ello se consigue reducir también el número de resultados innecesarios y se evita que el usuario tenga que buscar entre estos de forma manual.(*La Importancia de Una Barra de Búsqueda Inteligente - IONOS*, 2017)

2.2 Estado del arte

2.2.1 Internacional

2.2.1.1 The Impact of Continuous Integration on Other Software Development Practices: A Large-Scale Empirical Study

Lugar: Cuba

Autores: Adrián Hernández Yeja

Año: 2014

Resumen:

La Integración Continua (CI) se ha convertido en una innovación disruptiva en el desarrollo de software: con el soporte y la adopción adecuados de las herramientas, se han demostrado efectos positivos para el rendimiento de las solicitudes de extracción y la ampliación del tamaño de los proyectos. Como cualquier otra innovación, la adopción de CI implica adecuar las prácticas existentes para aprovechar al máximo su potencial, y para ello se han propuesto “mejores prácticas”. Aquí se estudió la adaptación y evolución de la redacción y envío de código, el cierre de emisión y extracción, y las prácticas de prueba como TRAVISCI es adoptado por cientos de proyectos establecidos en GITHUB. Cualitativamente, para ayudar a esencializar los resultados

cuantitativos, se encuesto a desarrolladores de GITHUB sobre la experiencia con la adopción de TRAVISCI. El hallazgos sugieren una imagen más matizada de cómo los equipos de GITHUB se están adaptando y beneficiándose de la tecnología de integración continua que lo sugerido por trabajos anteriores.(Zhao et al., 2017).

2.2.1.2 APLICACIÓN DEL PROCESO DE INTEGRACIÓN CONTINUA EN EL CENTRO DE TELEMÁTICA DE LA UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Lugar: The Netherlands

Autores: Yangyang Zhao, Alexander Serebrenik, Yuming Zhou, Vladimir Filkov, Bogdan Vasilescu

Año: 2017

Resumen:

La Integración Continua es una práctica de software expuesta por Martin Fowler, que permite entre otras bondades la reducción de riesgos y la realización de tareas repetitivas, automatizando al máximo los procesos involucrados en el desarrollo de software y permitiendo la generación de software listo para desplegar e incremento de su calidad. Cuando se cumplen los principios que rigen la Integración Continua se obtiene una herramienta eficaz y útil que ayuda a la sinergia del desarrollo de los proyectos de software. En este trabajo se pretenden mostrar las experiencias en la utilización del proceso de Integración Continua en el centro de Telemática de la Universidad de las Ciencias Informáticas. Se presentan las etapas, herramientas utilizadas y resultados obtenidos con la aplicación de esta técnica, resaltando las ventajas que se obtienen para todas las personas involucradas, principalmente los desarrolladores, los cuales son los mayores implicados en este proceso. Se utilizan los lenguajes de programación PHP y Python por ser éstos los utilizados en el centro y como servidor de Integración Continua Jenkins, uno de los más populares, robusto y

fácil de usar. Las etapas y herramientas que fueron establecidas son libres de ser modificadas de acuerdo a las necesidades que se requieran.(Adrian Hernández Yeja, 2015)

2.2.1.3 La integración continúa aplicada en el desarrollo de software en el ámbito científico

Lugar: Argentina

Autores: Alicia Salamon, Patricio Maller, Alejandra Boggio, Natalia Mira, Sofia Perez, Francisco Coenda.

Año: 2015

Resumen:

El proceso de integración de componentes que se requiere en los proyectos no es una tarea simple. La integración de software es un problema complejo sobre todo en sistemas que involucran código desarrollado por diferentes personas, por esta razón es necesario contar con un entorno que garantice la adecuada integración de las partes de un proyecto y posibilite visualizar los resultados de la integración de una manera fácil y clara. En este marco la Integración Continua ofrece un esquema que permite realizar integraciones a medida que se lleva a cabo el desarrollo generando incrementos pequeños y mostrando los resultados obtenidos. En este sentido el presente trabajo plantea un modelo de referencia cuya finalidad es construir una solución open source que implementa la Integración Continua, y permite evaluar los beneficios que aporta al proceso de desarrollo de software científico–técnico(Salamon et al., n.d.).

2.2.2 Nacionales

2.2.2.1 LA INTEGRACIÓN CONTINUA Y SU APORTE AL ASEGURAMIENTO DE LA CALIDAD EN EL CICLO DE VIDA DEL DESARROLLO DE SOFTWARE

Lugar: Bogotá

Autores: Ana María García Orozco

Año: 2015

Resumen:

La calidad de software se ha convertido en un eje dentro de las empresas que utilizan procesos de desarrollo de software, en el cual se están invirtiendo grandes esfuerzos para lograr obtener productos de alta calidad para alcanzar a ser un referente en el mercado de la producción de software. Actualmente el desarrollo de aplicaciones se encuentra apoyado por diferentes normas, certificaciones de procesos, además prácticas y herramientas que aportan mejoras en el diseño, implementación y desarrollo de software, esta investigación se centra en una de las prácticas que se vienen implementando las empresas para agilizar los procesos de desarrollo y controlar la calidad del mismo. Según Martín Flower, "La Integración Continua es una práctica de desarrollo de software en el que los miembros de un equipo integran su trabajo con determinada frecuencia, generalmente una persona se integra mínimo una vez al día, lo cual resulta en muchas integraciones diarias, cada integración es verificada por un ciclo automatizado (incluyendo las pruebas), para detectar errores de integración tan rápido como sea posible"1. Una vez conocidos los problemas comunes de las empresas colombianas que desarrollan Software como lo son: código inestable, integraciones (despliegues o liberaciones) fallidas, dificultad para identificar lo que se debe probar, disponibilidad del rol de pruebas para realizar pruebas en cualquier momento que sea necesario, falta de cultura en la realización de pruebas unitarias documentadas, limitada disponibilidad del estado el proyecto de desarrollo de software en tiempo real, entre otros, se hace evidente la necesidad de minimizar dichos problemas para conseguir una mayor calidad del producto; se decide realizar esta investigación con el fin de dar a conocer los beneficios de aplicar la Integración Continua, lograr presentar sus beneficios y la mejor forma de implementar esta práctica, para aportar en el mejoramiento de los procesos de aseguramiento de la calidad y optimizar la

utilización de herramientas de uso gratuito, que ayudan a controlar alguna actividad dentro del desarrollo de software, de la misma manera, mostrar a los profesionales en ingeniería de sistemas la forma de mejorar su calidad de vida en su entorno laboral con respecto a los traumatismos que se generan en el momento de realizar despliegue de modificaciones a las aplicaciones en los ambientes de pruebas o productivo y a facilidad de identificar errores en el software ya que se compila en cortos espacios de tiempo el código generado por todo el equipo de desarrollo (Ana María García Orozco, 2015).

2.2.3 Aportes extraídos

Los autores de los anteriores documentos recalcan la importancia de implementar un conjunto de buenas prácticas en los entornos de desarrollo con el fin de que las organizaciones puedan implementar de manera formal la integración a sus procesos operativos y de desarrollo mejorando la calidad del software y propiciando métodos para que exista un mayor flujo y control en los diferentes ciclos de desarrollo.

3 Análisis preliminar

En este capítulo se realizará una investigación más profunda sobre las herramientas más conocidas y que se van a utilizar en el proyecto, determinando de cada una de estas herramientas las características más relevantes, con el fin de recopilar mayor información de las mismas y poder tomar una decisión que se adapte a las necesidades de la empresa y el proyecto.

3.1 Principales herramientas de integración continua

Las herramientas de integración continua ayudan en la creación de un repositorio, en la ejecución de las pruebas y en la compilación, así como en el control de versiones y, por supuesto, en la propia integración continua**.** (*Las Mejores Herramientas de Integración Continua - IONOS*, 2020)

3.1.1 Jenkins

Jenkins es una de las herramientas de desarrollo de CI / CD y automatización de compilación de código abierto autogestionadas más populares del mundo. Obtiene su increíble flexibilidad al incorporar capacidades de sus cientos de complementos disponibles, lo que le permite respaldar la construcción, implementación y automatización de cualquier proyecto(*Jenkins vs. GitLab | GitLab*, n.d.)

3.1.1.1 Características de Jenkins.

- **Integración continua y entrega continua:** Como servidor de automatización extensible, Jenkins puede usarse como un servidor CI simple o convertirse en el centro de entrega continua para cualquier proyecto.
- **Fácil instalación:** Jenkins es un programa autónomo basado en Java, listo para ejecutarse de inmediato, con paquetes para Windows, Mac OS X y otros sistemas operativos similares a Unix.

- **Fácil configuración:** Jenkins se puede instalar y configurar fácilmente a través de su interfaz web, que incluye verificaciones de errores sobre la marcha y ayuda incorporada.
- **Complementos:** Con cientos de complementos en el Centro de actualización, Jenkins se integra con prácticamente todas las herramientas en la cadena de herramientas de integración continua y entrega continua.
- **Extensible:** Jenkins se puede extender a través de su arquitectura de complementos, lo que brinda posibilidades casi infinitas de lo que Jenkins puede hacer.
- **Repartido:** Jenkins puede distribuir fácilmente el trabajo en varias máquinas, lo que ayuda a impulsar las compilaciones, las pruebas y las implementaciones en varias plataformas más rápidamente.(*Jenkins*, n.d.)

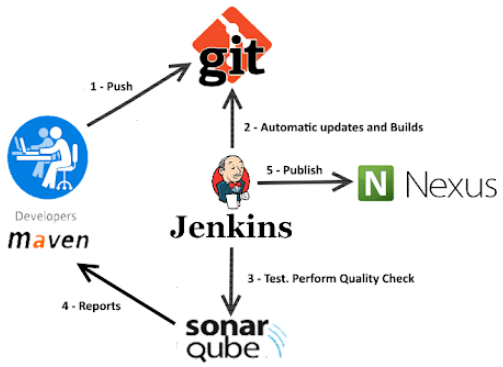


Ilustración 1 Configuración Jenkins

3.1.2 GitLab

GitLab es un servicio web de control de versiones y desarrollo de software colaborativo basado en Git. Además de gestor de repositorios, el servicio ofrece también alojamiento de wikis y un sistema de seguimiento de errores, todo ello publicado bajo una Licencia de código abierto.(*GitLAb Flujo de Trabajo Dinámicos En Un Solo Lugar*, 2020)

3.1.2.1 Características de GitLab

- **Seguimiento de problemas:** Planifique, organice y realice un seguimiento del progreso del proyecto con problemas, etiquetas, ponderaciones (puntos de la historia), hitos (sprints y lanzamientos), seguimiento del tiempo, fechas de vencimiento y asignados mediante Scrum, Kanban, SAFe y otras metodologías.
- **Seguimiento del tiempo:** Estimar, rastrear e informar sobre el tiempo dedicado a los problemas.
- **Tableros:** Priorice, gestione y realice un seguimiento visual de la ejecución del trabajo con placas potentes y flexibles.
- **Epics:** Planifique el próximo trabajo creando Epics y mapeando todos los problemas relevantes para ellos. Cree y realice un seguimiento de múltiples hitos a nivel de cartera para ver el estado de las horas extra y revisar el progreso hacia sus objetivos.
- **Hojas de rutas:** Planifique y mapee proyectos visualmente en una vista de hoja de ruta que se puede utilizar para el seguimiento y la comunicación.
- **Servicio de mesa:** Conecte a su equipo mediante problemas de GitLab con partes externas directamente por correo electrónico para recibir comentarios y asistencia, sin necesidad de herramientas adicionales.
- **Gestión de requerimientos:** Recopile y administre los casos de uso y los requisitos para cumplir con los objetivos comerciales.
- **Gestión calidad:** Planifique y realice un seguimiento de las pruebas y la calidad de su producto.(*GitLAb Flujo de Trabajo Dinámicos En Un Solo Lugar*, 2020)

Ilustración 2 Flujo de IC Git Lab

3.1.3 Travis

Travis-CI es un sistema de Integración Continua, gratuita para proyectos Open Source y de pago para proyectos privados. Se integra sin problemas con GitHub y automáticamente ejecuta el pipeline definido en cada push o pull requests. Testea y buildea aplicaciones escritas en Ruby, Node, Objective-C, Go, Java, C# y F#, entre otras (Federico Toledo, 2018).

3.1.3.1 Características de Travis CI

- **Configuración en segundos:** Inicie sesión con su plataforma en la nube, dígale a Travis CI que pruebe un proyecto y luego presione.
- **Apoya su plataforma:** Muchas bases de datos y servicios están preinstalados y pueden habilitarse en su configuración de compilación.
- **Prueba solicitudes de extracción:** Asegúrese de que todas las solicitudes de extracción de su proyecto se prueben antes de fusionarlas.
- **Implementar en cualquier lugar:** Actualizar la puesta en escena o la producción tan pronto como pasen las pruebas. (*Travis CI - Test and Deploy with Confidence*, n.d.)

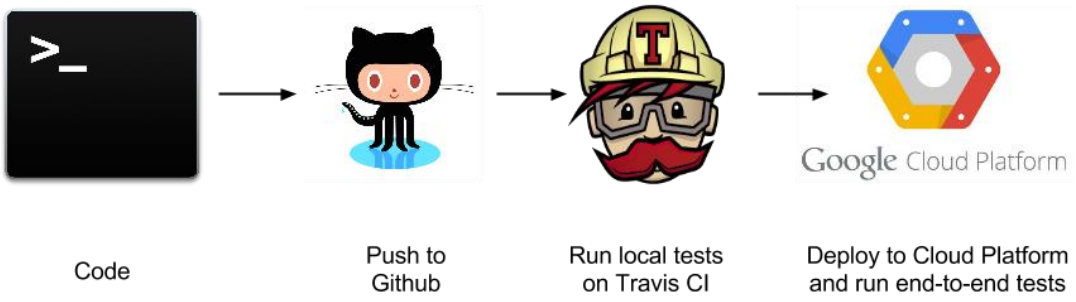


Ilustración 3 Flujo de IC Travis

3.1.4 Git Actions

GitHub es un repositorio de código colaborativo para alojar y revisar código, administrar proyectos y crear software. Ofrece todas las funciones de gestión de código fuente (SCM) y control de versiones distribuidas de Git, además de añadir sus propias funciones. Proporciona control de acceso y varias funciones de colaboración, como seguimiento de errores, solicitudes de funciones, gestión de tareas y wikis para cada proyecto. En noviembre de 2019, GitHub anunció la disponibilidad general de GitHub Actions para todos los usuarios. La función GitHub Actions permite ejecutar fragmentos de código en un contenedor en una amplia variedad de llamadas a la API de GitHub. Esto tiene la promesa de permitir a los usuarios orquestar sus flujos de trabajo en función de cualquier evento. Con las acciones de GitHub, los flujos de trabajo y los pasos son solo código en un repositorio. Las acciones permiten que GitHub ofrezca CI / CD, lo que facilita la automatización de la forma en que compila, prueba e implementa sus proyectos e incluye soporte de ejecución para Linux, macOS y Windows. Ejecuta sus flujos de trabajo en un contenedor o en una máquina virtual.(*GitHub vs. GitLab | GitLab*, n.d.)

3.1.4.1 Características de Git Actions

- **Linux, macOs, Windows, Arm y contenedores:** Los runners(Aplicación que ejecuta un job desde un workflow de GitHub Actions.(Thomas Boop, 2020)) alojados para todos los

sistemas operativos principales facilitan la creación y prueba de todos sus proyectos. Ejecutar directamente en una máquina virtual o dentro de un contenedor. Utilice sus propias máquinas virtuales, en la nube o en las instalaciones, con runners auto-hospedados.

- **Cualquier lenguaje**: Las acciones de GitHub son compatibles con Node.js, Python, Java, Ruby, PHP, Go, Rust, .NET y más. Cree, pruebe e implemente aplicaciones en el idioma que elija.
- **Flujos de trabajo impulsado por la comunidad:** GitHub Actions conecta todas sus herramientas para automatizar cada paso de su flujo de trabajo de desarrollo. Implemente fácilmente en cualquier nube, cree tickets en Jira o publique un paquete en npm.

3.2 Herramientas *de versionamiento de código*

Control y gestión de versiones es la gestión de los cambios que suceden sobre la configuración de un producto o elementos de este.

Un Sistema de Versionado de Código es algo esencial para todos los programadores, ya que este les ayuda a poder compartir nuestro código a los desarrolladores, y así estos obtendrían una copia local y publicar cambios a un repositorio.(Andrea Garcia, 2019)

3.2.1 SVC

CVS (Concurrent Versions System), un sistema de control de versiones de GNU disponible para la mayoría de las plataformas (Unix, Linux, Windows, etc.). Su funcionamiento es muy simple: el desarrollador se conecta con el servidor CVS y le pide la última versión disponible del proyecto, con lo cual puede ver qué cambios se han realizado respecto a su versión local y los conflictos que pudiera ocasionar el código que ha estado realizando con el que ya está disponible en el servidor.

En caso de que no sea problemático, se modifican los ficheros locales respetando los cambios efectuados por el desarrollador. El servidor CVS se encarga de manejar el histórico de lo que ocurre, mantener un registro de los cambios realizados a cada fichero y de servirlos según las necesidades del desarrollador. Además, gestiona diversas utilidades para controlar en qué ficheros se está trabajando, notificar a los autores de los ficheros de los cambios, etc. El cliente CVS se encarga de obtener las últimas versiones disponibles (o las que necesite), de confrontarlas con las copias locales, y de añadir el código del desarrollador al proyecto. (Agustín González et al., n.d.)

3.2.2 Subversión

Subversión es un software libre que permite el control de versiones, permite el acceso al repositorio a través de redes ofreciendo la posibilidad de trabajar desde diferentes equipos, consiguiendo de esta manera la colaboración entre distintos miembros del proyecto.

El lanzamiento oficial fue en el 2000, pero su última actualización fue hace apenas unos meses, su uso es bajo licencia Apache y es multilingüe. El funcionamiento de subversión se podría asimilar al de un gestor de ficheros de tipo repositorio.

Subversión se basa en un sistema centralizado, la principal **ventaja** de un **sistema centralizado** es su **simplicidad**, pero por el contrario no permite tener más de **un repositorio central** sobre el que trabajar, no se pueden realizar **confirmaciones** (commit) si no se está conectado al repositorio central y si se trabaja en un equipo de trabajo con muchos usuarios, la **colaboración** se complica.(Agustín González et al., n.d.)

3.2.3 Mercurial

Mercurial es un sistema de control de versiones multiplataforma, para desarrolladores de software. Esta implementado principalmente haciendo uso del lenguaje de programación Python, pero

incluye una implementación binaria descrita en C. Mercurial fue escrito originalmente para funcionar sobre GNU/Linux, pero ha sido adaptado para Windows, Mac OS X y otros sistemas tipo Unix.

Entre las principales metas de desarrollo de Mercurial se incluyen un gran rendimiento y escalabilidad. Es un desarrollo completamente distribuido, sin necesidad de un servidor central, provee una gestión robusta de archivos tanto de texto como binarios y capacidades avanzadas de ramificación e integración (todo ello manteniendo sencillez conceptual). También incluye una interfaz web integrada.(Mello Teggia & Tula, 2015)

3.2.4 Tortoise SVN

TortoiseSVN es un cliente gratuito de código abierto para el sistema de control de versiones *Apache™ Subversion®*. Esto significa que TortoiseSVN administra archivos y directorios a lo largo del tiempo. Los archivos se almacenan en un *repositorio* central. El repositorio es prácticamente lo mismo que un servidor de archivos ordinario, con la excepción de que recuerda todos los cambios que se hayan hecho a sus archivos y directorios. Esto le permite al usuario recuperar versiones antiguas de sus archivos y examinar la historia de cómo y cuándo cambiaron sus datos, y quién hizo el cambio. Esta es la razón por la que mucha gente piensa en Subversion, y los sistemas de control de versiones en general, como una especie de “máquinas del tiempo”.

Algunos sistemas de control de versiones también son sistemas de manejo de configuración del software (SCM por sus iniciales en inglés). Estos sistemas están diseñados específicamente para manejar árboles de código fuente, y tienen muchas características que son específicas para el desarrollo de software - tales como el entendimiento nativo de los lenguajes de programación, o proporcionan herramientas para compilar software. Subversion, sin embargo, no es uno de estos

sistemas; es un sistema general que puede ser utilizado para manejar *cualquier* colección de archivos, incluyendo código fuente.(*Prefacio*, n.d.)

3.2.5 Git

Git es, con diferencia, el sistema de control de versiones moderno más utilizado del mundo. Git es un proyecto de código abierto maduro y con un mantenimiento activo que desarrolló originalmente Linus Torvalds, el famoso creador del kernel del sistema operativo Linux, en 2005. Un asombroso número de proyectos de software dependen de Git para el control de versiones, incluidos proyectos comerciales y de código abierto. Los desarrolladores que han trabajado con Git cuentan con una buena representación en la base de talentos disponibles para el desarrollo de software, y este sistema funciona a la perfección en una amplia variedad de sistemas operativos e IDE (entornos de desarrollo integrados).

Git, que presenta una arquitectura distribuida, es un ejemplo de DVCS (sistema de control de versiones distribuido, por sus siglas en inglés). En lugar de tener un único espacio para todo el historial de versiones del software, como sucede de manera habitual en los sistemas de control de versiones antaño populares, como CVS o Subversion (también conocido como SVN), en Git, la copia de trabajo del código de cada desarrollador es también un repositorio que puede albergar el historial completo de todos los cambios.

Además de contar con una arquitectura distribuida, Git se ha diseñado teniendo en cuenta el rendimiento, la seguridad y la flexibilidad.(*Qué Es Git: Conviértete En Todo Un Experto En Git Con Esta Guía*, n.d.)

3.3 Herramientas para planificación y gestión

Este tipo de herramientas busca brindar a los desarrolladores una guía la cual permita planear y documentar los distintos casos de pruebas e incidencias, y además gestionar los resultados. Estas herramientas nos permiten tener una visión general de los casos de prueba del proyecto.

A continuación, se presentarán algunas de ellas.

3.3.1 Asana

Asana es una herramienta de productividad en línea para la planificación de proyectos y la gestión de tareas en un entorno de trabajo colaborativo. El objetivo principal de Asana es facilitar el trabajo en equipo para que el personal se concentre en los objetivos a alcanzar y en el trabajo a realizar en lugar de enfocarse en la organización del proyecto.

En el blog de (Anaraya Albornoz, 2020) se presentan varias de sus características:

- **Transparencia:** Poco importa el número de tareas, proyectos, documentos o informaciones, Asana permite conservar un espacio legible e intuitivo.
- **Agilidad:** Asana simplifica y flexibiliza la organización tradicional del trabajo. Todos los participantes pueden alimentar una o varias tareas y el responsable de proyecto se encarga de controlar la viabilidad y ejecución. La herramienta es transparente y reúne estrategias y operaciones para el éxito del proyecto.
- **Adaptabilidad:** Asana permite diferentes vistas, filtros, clasificaciones y organizaciones para que cada colaborador pueda navegar en la herramienta y acceder a la información según su funcionamiento personal y sus necesidades profesionales.
- **Integración:** Asana puede ser considerado un hub colaborativo para gestionar la colaboración en empresas, la comunicación formal e informal, la gestión documental y

mucho más gracias a cientos de integraciones posibles (Google Drive, DropBox, Slack, Gmail, etc.). La centralización de la información refuerza la capacidad de la herramienta para aumentar la productividad.

- **Universalidad:** Asana es un software que conviene tanto al freelance que busca un gestor de tareas como a las grandes corporaciones, con cientos de empleados, que buscan una solución para la gestión de carteras de proyectos o PPM (Project Portfolio Management) para la gestión de proyectos complejos. Las diferentes vistas y la flexibilidad en la experiencia del utilizador permiten a todos los miembros del equipo sentirse cómodos en la utilización del programa.

3.3.2 Trello

Esta es una herramienta digital, entre muchas otras de su tipo, que permite organizar visualmente contenidos diversos y compartirlos con otras personas. Contiene una interfaz visual que funciona de forma muy intuitiva. Además, es muy flexible. Actualmente es usada por millones de personas y empresas de todos los tipos y tamaños, en especial para apoyar la gestión de proyectos y tareas en equipo.

Para que el uso de Trello se sostenga en el tiempo y genere valor, es fundamental que la cultura organizacional, propia del equipo, se oriente al uso efectivo de la herramienta. Esto implica ciertos aspectos que se mencionan el autor (Javier Guillot, 2019) en su apartado web

- **Asegurar un propósito compartido:** identificar colectivamente la necesidad de gestionar tareas en equipo. Para ello, es útil invitar a un ejercicio grupal de reflexión en el que se identifiquen los “dolores colectivos” relacionados con la ausencia de procesos y herramientas compartidas para gestionar tareas (ejemplos: ineficiencias, errores de

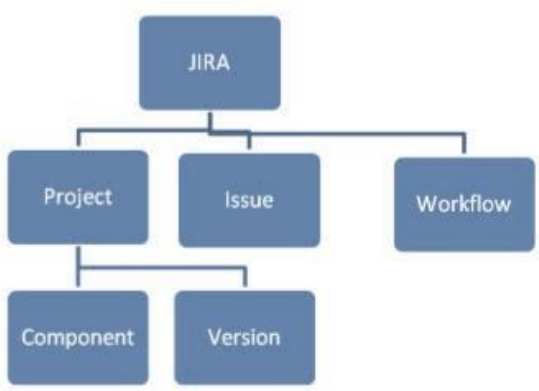
comunicación, dificultades de coordinación) y los posibles beneficios de usar una única herramienta para este propósito.

- **Asegurar el liderazgo:** es imprescindible que las personas que ocupan roles de liderazgo promuevan el uso de la herramienta mediante su ejemplo constante, usándola para la gestión de sus propias tareas y como referencia para la coordinación, seguimiento y reporte.
- **Balancear reglas fijas de uso en equipo con flexibilidad de uso individual**: es necesario construir y compartir reglas en equipo para que el uso como herramienta de gestión y coordinación sea consistente. También es necesario dejar espacio para la flexibilidad en la aproximación individual a la definición de tareas, en especial motivando a que cada persona agregue sus propias tareas y las actualice según su progreso.
- **Adoptar una lógica de experimentación iterativa:** evaluar periódicamente el uso de Trello y recabar evidencia sobre fortalezas y oportunidades de mejora (por ejemplo, mediante el uso de formularios anónimos en línea).

3.3.3 Jira

Es una herramienta de seguimiento de problemas o gestión de proyectos. Por lo general, se utiliza para el seguimiento de errores o problemas, y la gestión de proyectos.

Los aspectos fundacionales conceptuales de JIRA son el problema, el proyecto y el flujo de trabajo. Seguidamente se ilustra como lo explica (YANA GUSTI, 2019) en su apartado WEB



- **Issue (Problema):** se puede considerar como cualquier actividad que se cree y se rastree a través de Jira como, por ejemplo:
  - Error de software
  - La tarea de un proyecto
  - El formulario de solicitud de licencia
  - Ticket de ayuda.
- **Flujo de trabajo:** Conjunto de estados y transiciones por los que se mueve durante todo su ciclo de vida, representando los procesos internos de la organización, estas etapas son:
  - Tema abierto
  - Problema resuelto
  - Problema en progreso
  - Problema reabierto
  - Problema cerrado.
- **Proyecto:** Consiste en una colección de temas, sus principales atributos son:
  - Nombre seleccionado por el administrador
  - Identificador para el nombre del proyecto
  - Agrupación lógica de problemas dentro de un proyecto

Este software de gestión proporciona múltiples utilidades, una de las más relevante es que permite tratar con errores o bugs dentro del software El potente motor de flujo de trabajo de Jira garantiza que los errores se asignen y prioricen automáticamente una vez que se capturan. Luego, los equipos pueden rastrear un error hasta su finalización.

3.4 Motores de Búsqueda

Cole Thienes en su artículo expone algunos de los buscadores basados en inteligencia artificial diferentes al de Google.

3.4.1 Okapi BM25

En la recuperación de información, **Okapi BM25** (BM es una abreviatura de *mejor coincidencia*) es una función de clasificación utilizada por los motores de búsqueda para estimar la relevancia de los documentos para una consulta de búsqueda determinada. Se basa en el marco de recuperación probabilístico desarrollado en las décadas de 1970 y 1980 por Stephen E. Robertson, Karen Spärck Jones y otros.

BM25 es una función de recuperación de bolsa de palabras que clasifica un conjunto de documentos según los términos de consulta que aparecen en cada documento, independientemente de su proximidad dentro del documento. Es una familia de funciones de puntuación con componentes y parámetros ligeramente diferentes. Una de las instancias más destacadas de la función es la siguiente.(Sparck Jones et al., 2000)

3.4.2 MS MARCO

MS MARCO es una colección de conjuntos de datos centrados en el aprendizaje profundo en la búsqueda.

El primer conjunto de datos fue un conjunto de datos de respuesta a preguntas con 100.000 preguntas reales de Bing y una respuesta generada por humanos. Desde entonces, publicamos un conjunto de datos de 1.000.000 de preguntas, un conjunto de datos de generación de lenguaje natural, un conjunto de datos de clasificación de pasajes, un conjunto de datos de extracción de frases clave, un conjunto de datos de rastreo y una búsqueda conversacional.

El proceso general se ejecuta de la siguiente manera.(*GitHub - Microsoft / MSMARCO-Question-Answering: MS MARCO (Microsoft Machine Reading Comprehension) Es Un Conjunto de Datos a Gran Escala Centrado En La Comprensión de Lectura de La Máquina y La Respuesta a Preguntas*, 2020)

- Los registros de Bing se muestrean, filtran y anonimizan para asegurarnos de que las consultas que se recopilaron sean útiles para la comunidad de investigación y respetuosas con nuestros usuarios y fans de Bing.
- Usando las consultas de muestra y anónimas, Bing genera los 10 pasajes más relevantes para la consulta.
- Los jueces altamente capacitados leen la consulta y sus pasajes relacionados y si hay una respuesta presente, se anotan los pasajes de apoyo y se genera una respuesta en lenguaje natural.
- Una menor proporción de consultas (~ 17% del conjunto de datos general con 182,887 consultas únicas) luego se pasa a una segunda ronda de jueces a quienes se les pide que verifiquen que la respuesta sea correcta y que reescriban (si es posible) la consulta para que sea una respuesta bien formada. Estas respuestas están diseñadas para ser entendidas sin un contexto perfecto y están diseñadas con altavoces inteligentes / asistentes digitales en mente.

3.4.3 NBoost

Es una plataforma escalable que impulsa los motores de búsqueda para desarrollar e implementar modelos de última generación para mejorar la relevancia de los resultados de búsqueda.

Nboost aprovecha los modelos optimizados para producir motores de búsqueda neuronales específicos del dominio. La plataforma también puede mejorar otras tareas posteriores que requieren información clasificada, como la respuesta a preguntas.(Cole Thienes, 2019)

3.4.4 Elasticsearch

Elasticsearch es un motor de analítica y análisis distribuido y open source para todos los tipos de datos, incluidos textuales, numéricos, geoespaciales, estructurados y desestructurados. Elasticsearch está desarrollado en Apache Lucene y fue presentado por primera vez en 2010 por Elasticsearch N.V. (ahora conocido como Elastic). Conocido por sus API REST simples, naturaleza distribuida, velocidad y escalabilidad, Elasticsearch es el componente principal del Elastic Stack, un conjunto de herramientas open source para la ingesta, el enriquecimiento, el almacenamiento, el análisis y la visualización de datos. Comúnmente referido como el ELK Stack (por Elasticsearch, Logstash y Kibana), el Elastic Stack ahora incluye una gran colección de agentes de envío conocidos como Beats para enviar los datos a Elasticsearch.(*¿Qué Es Elasticsearch? | Elastic*, n.d.)

4 Definición base.

Como resultado del capítulo 3, se obtiene las principales bases teóricas para realizar la toma de decisiones sobre los tipos de herramientas que se van a seleccionar para llevar a cabo con la finalidad del proyecto. En este capítulo se describe que factores se tuvieron en cuenta para la selección de las herramientas realizando cuadros comparativos dándole mayor importancia a la relación costo beneficio y así brindarle una mayor ganancia en el proyecto de la empresa BeGo.

4.1 Ventajas y Desventajas de los tipos de herramientas para el versionamiento de repositorio

El versionamiento del código es un punto importante para destacar dado que es una herramienta que puede aportar numerosos beneficios al equipo de desarrolladores teniendo en cuenta que se trabaja en colaboración en un mismo proyecto, pero en diferentes funcionalidades, causando choques y errores en el momento que se realiza la unión de los diferentes códigos de cada desarrollador.

Para la elección de esta herramienta se tuvo en cuenta, los antecedentes que tienen estas para la resolución de conflictos y la rapidez con la cual se resuelven, la existencia de copias de seguridad, y la fácil implementación.

	SVN	GIT	CVS	Mercurial
Control de versiones	Centralizada	Distribuida	Centralizada	Distribuida
Repositorio	Un repositorio central donde se generan copias de trabajo	Copias locales del repositorio en las que se trabaja directamente	Almacén de datos persistente que coordina el acceso multiusuario a	cada usuario tiene su propio repositorio del cual es administrador y tiene el control para tomar

			los proyectos y a su contenido.	cualquier decisión, es decir, permite el manejo de múltiples flujos de trabajo simultáneos
Autorización de acceso	Dependiendo de la ruta de acceso	Para la totalidad del directorio		
Seguimiento de cambios	Basado en archivos	Basado en contenido	Basado en archivos	Basado en Directorio de archivos
Historial de cambios	Solo en el repositorio completo, las copias de trabajo incluyen únicamente la versión más reciente	Tanto el repositorio como las copias de trabajo individuales incluyen el historial completo		
Conectividad de red	Con cada acceso	Solo necesario para la sincronización		

Tabla 2 características de las herramientas para versionamiento (*Git vs. SVN: Una Comparativa Del Control de Versiones - IONOS*, 2020)

4.1.1 Ventajas y desventajas de cada una de las herramientas.

Estas ventajas y desventajas son tomadas de referencia de algunos sitios web únicamente como criterios.

4.1.1.1 Ventajas y desventajas SVN

Ventajas	Desventajas
○ Se sigue la historia de los archivos y directorios a través de copias y renombrados. ○ Las modificaciones (incluyendo cambios a varios archivos) son atómicas. ○ La creación de ramas y etiquetas es una operación más eficiente; Tiene costo de complejidad constante (O (1)) y no lineal (O(n)) como en CVS. ○ Se envían sólo las diferencias en ambas direcciones (en CVS siempre se envían al servidor archivos completos). ○ Puede ser servido, mediante Apache, sobre WebDAV/DeltaV. Esto permite que clientes WebDAV utilicen Subversion en forma transparente. ○ Maneja eficientemente archivos binarios (a diferencia de CVS que los trata internamente como si fueran de texto). ○ Permite selectivamente el bloqueo de archivos. Se usa en archivos binarios que, al no poder fusionarse fácilmente, conviene que no sean editados por más de una persona a la vez. ○ Cuando se usa integrado a Apache permite utilizar todas las opciones que este servidor provee a la hora de autentificar archivos (SQL, LDAP, PAM, etc.).	○ El manejo de cambio de nombres de archivos no es completo. Lo maneja como la suma de una operación de copia y una de borrado. ○ No resuelve el problema de aplicar repetidamente parches entre ramas, no facilita el llevar la cuenta de qué cambios se han trasladado. Esto se resuelve siendo cuidadoso con los mensajes de commit. Esta carencia será corregida en la próxima versión.

Tabla 3 Ventajas y desventajas SVN

4.1.2 Ventajas y desventajas de CVS

Ventajas	Desventajas
○ Utiliza arquitectura cliente- servidor. ○ El servidor guarda las versiones y el historial. ○ Los clientes se conectan y hacen una copia completa del proyecto. ○ Funciona en cualquier sistema operativo. ○ Varias personas pueden sacar una copia al mismo tiempo. ○ Puede mantener distintas ramas de un Proyecto (Gabriel Fernando Chiriboga Rogel, 2013)	○ No soporta refactorización de sistemas de forma automática o versionadas. ○ Limitada para UTF-8 Unicode o archivos con contenido diferente ACII. ○ El protocolo no soporta eliminación de directorios o re nombrarlos. ○ Visual Studio carece de soporte nativo a CVS (Gabriel Fernando Chiriboga Rogel, 2013)

Tabla 4 Ventajas y desventajas de CVS

4.1.3 Ventajas y desventajas de Mercurial

Ventajas	Desventajas
○ Escalable y adaptable al tamaño y exigencias del proyecto. ○ Escrito en Python por ende es más fácil su manejo. ○ Funciona bien sobre páginas y directorios web (*Mercurial (Sistema de Control de Versiones) - EcuRed*, n.d.)	○ Pocas caracteristicas añadidas por default ○ Comunidad de desarrollo muy pequeña. (*Mercurial (Sistema de Control de Versiones) - EcuRed*, n.d.)

Tabla 5 Ventajas y desventajas de Mercurial

4.1.4 Ventajas y desventajas de GitHub

Ventajas	Desventajas
○ **Sistema distribuido**, sin un punto central de fallo, que permite el trabajo incluso sin conexión. ○ **Super rápido y ligero**, optimizado para hacer operaciones de control muy rápidas. ○ **Crear ramas y mezclarlas** es rápido y poco propenso a problemas, al contrario que en otros sistemas tradicionales. ○ **La integridad de la información está asegurada** gracias a su modelo de almacenamiento, que permite predecir este tipo de problemas. En sistemas tradicionales este era un problema grave. ○ **Permite flujos de trabajo muy flexibles**. ○ **El concepto de área de preparación** o *staging* permite versionar los cambios como nos convenga, no todo o nada. ○ **¡Es gratis!** y de código abierto. (José Manuel Alarcón, 2020)	○ **Es más complejo** que los sistemas centralizados tradicionales porque entran en juego más repositorios, más operaciones y más posibilidades para trabajar en equipo, que hay que decidir. ○ **La curva de aprendizaje es empinada**. Lo básico lo aprendes enseguida, pero la realidad te demuestra que no es suficiente "tocar de oído" con él. La documentación es tan compleja que muchas veces no resulta de ayuda. ○ **Los comandos y algunos conceptos** que usa pueden llegar a ser **confusos**, al igual que algunos mensajes que muestra. ○ Por defecto, **se lleva mal con archivos binarios muy grandes**, como vídeos o documentos gráficos muy pesados. Por suerte existen soluciones para ello (Git LFS). (José Manuel Alarcón, 2020)

Tabla 6 Ventajas y desventajas de GitHub

4.2 Ventajas y Desventajas de los tipos de herramientas para planificación y gestión del proyecto

Los programas software de gestión de proyectos ayudan a visualizar de forma global el ciclo de vida del proyecto y a hacer un seguimiento de todo lo que conlleva de manera mucho más simple y sencilla, evitando de esta manera desviaciones tanto en plazo, presupuesto y/o calidad. Además, permiten visibilizarlos diferentes proyectos en los que se está trabajando, los miembros del equipo, sus tare as asignadas y la evolución en la ejecución de las mismas (Belén Soto Lull, 2016).

	Ventajas	**Desventajas**
Asana	○ **Actualizaciones en tiempo real** que permiten ver rápidamente los últimos cambios hechos. ○ **Interfaz sencilla e intuitiva** que facilitan mucho el uso de la herramienta y reducen la curva de aprendizaje. ○ **Función de prioridad de tareas**, para ayudarte a ti y a tu equipo a ser más productivo y eficaz. ○ **Versión gratuita muy completa.** ○ **Visualización de objetivos** generales para supervisar el progreso del proyecto. (Javier Gobea, 2018)	○ No tiene acceso fuera de línea. ○ La función de búsqueda necesita algo más de desarrollo, ya que es un poco torpe. ○ Los planes de pago incluyen funcionalidades que pueden resultar poco útiles. (Javier Gobea, 2018)
Trello	○ **Sistema de drag & drop (arrastrar y soltar) muy sencillo** que permite cambiar el estado de una tarea de forma muy cómoda. ○ **Eficaz a la hora de compartir información** relacionada con una tarea: puedes adjuntar explicaciones y archivos para cada tarea en su	○ **Puede resultar demasiado simple** si quieres gestionar proyectos complejos. ○ **No puedes filtrar tareas** por usuario o fechas límite. ○ Si una columna tiene demasiadas tarjetas de tareas, se pierde la

	correspondiente tarjeta y estar disponible para todos los miembros de tu equipo. ○ **Tableros visuales** que facilitan la organización, gestión y el seguimiento de tareas. ○ **Posibilidad de ampliar el plan** gratuito a uno de pago para conseguir más funcionalidades, integraciones y seguridad avanzada (Javier Gobea, 2018)	posibilidad de verlo todo de un vistazo porque obliga a hacer scroll con el ratón. (Javier Gobea, 2018)
Jira	○ **Colaboración en tiempo real** entre los miembros del equipo. ○ **Integración** con otras herramientas de trabajo. ○ **Actualización** continua del desarrollo de los flujos de trabajo. (Javier Gobea, 2018)	○ Personalización limitada. ○ Base de conocimiento no integrada. ○ No hay gestión de los elementos de configuración. (Javier Gobea, 2018)

Tabla 7 Comparación entre las ventajas y desventajas de las herramientas para la planificación y gestión del proyecto

4.3 Ventajas y Desventajas de los tipos de herramientas para integración continua.

Para la selección de esta, se realizó una previa exploración de las principales herramientas utilizadas en este campo que cumplieran con los requisitos que se contaban para el proyecto, como lo es:

- Open source
- Fácil implementación
- Automatización del proyecto
- No necesite de muchas herramientas adicionales.
- Se adapte a cualquier tipo de proyecto tanto actual como futuro.
- Maneje la mayor cantidad de lenguajes.

4.3.1 Jenkins

Ventajas	Desventajas
○ Es de código abierto y es fácil de usar, fácil de instalar y no requiere instalaciones o componentes adicionales. ○ Es gratis. ○ Fácilmente configurable. Jenkins se puede modificar y extender fácilmente. Implementa código de forma instantánea, genera informes de prueba. Jenkins se puede configurar de acuerdo con los requisitos de integraciones continuas y entrega continua. ○ Plataforma independiente. Jenkins está disponible para todas las plataformas y diferentes sistemas operativos, ya sea OS X, Windows o Linux. ○ Rich Plugin ecosystem. El amplio conjunto de complementos hace que Jenkins sea flexible y permita construir, implementar y automatizar en varias plataformas. ○ Fácil soporte debido a que es de código abierto y ampliamente utilizado, no hay escasez de soporte de grandes comunidades en línea de equipos ágiles. ○ El desarrollador escribe las pruebas para detectar los errores de su código lo más rápido posible. De modo que el tiempo del desarrollador se guarda sin desperdiciar integraciones plagadas de errores a gran escala.	○ Su interfaz está desactualizada y no es fácil de usar en comparación con las tendencias actuales de la interfaz de usuario. ○ Aunque Jenkins es amado por muchos desarrolladores, su mantenimiento no es tan fácil porque Jenkins se ejecuta en un servidor y requiere algunas habilidades como administrador del servidor para monitorear su actividad. ○ Una de las razones por las que muchas personas no implementan Jenkins se debe a su dificultad para instalar y configurar Jenkins. ○ Las integraciones continuas se rompen regularmente debido a algunos pequeños cambios de configuración. La integración continua se detendrá y, por lo tanto, requiere la atención del desarrollador. (*What Is Jenkins? Continuous Integration (CI) Tool*, n.d.)

○ Los problemas se detectan y resuelven casi de inmediato, lo que mantiene el software en un estado en el que se puede liberar en cualquier momento de forma segura. ○ La mayor parte del trabajo de integración está automatizado. Por lo tanto, los problemas de integración son menores. Esto ahorra tiempo y dinero durante la vida útil de un proyecto. (*What Is Jenkins? Continuous Integration (CI) Tool*, n.d.)	

Tabla 8 Ventajas y desventajas de Jenkins

4.3.2 Git Lab

Ventajas	Desventajas
○ Es gratis para los usuarios. Los usuarios de GitLab pueden crear repositorios ilimitados. Solo funciona con la edición Community, y si los usuarios desean una versión Enterprise, deben pagar. Las características adicionales de la versión Enterprise hacen que la experiencia de trabajo sea más rápida y sencilla en términos de operación de herramientas virtuales, control de actividades del servidor y administración del trabajo. ○ GitLab tiene licencia de código abierto. ○ Tiene la capacidad de rastrear errores y corregirlos en línea. ○ Está integrado con Lightweight Directory Access Protocol, por lo que es posible ubicar y acceder a diferentes recursos en la web. La edición GitLab Enterprise funciona con numerosos	○ Interfaz comparativamente lenta. ○ A menudo surgen problemas habituales con los repositorios. (*GitHub vs. GitLab vs. Bitbucket: GitLab*, n.d.)

servicios LDAP y sincronización de equipos. ○ Funciona con la importación de Git. (*GitHub vs. GitLab vs. Bitbucket: GitLab*, n.d.)	

Tabla 9 Ventajas y desventajas de Git Lab

4.3.3 Travis

Ventajas	Desventajas
○ Ligero y fácil de configurar ○ Gratis para proyectos de código abierto ○ No se necesita servidor dedicado ○ Función de matriz de construcción (Cody Arsenault, 2017)	○ Los planes empresariales tienen un costo ○ Opciones limitadas de personalización (Cody Arsenault, 2017)

Tabla 10 Ventajas y desventajas de Travis

4.3.4 Git Actions

Ventajas	Desventajas
○ Notificará de cualquier PR o *merge ti master* (se pueden establecer políticas de notificación). ○ Creará un entorno limpio (por ejemplo, en nuestro caso tomará una imagen de **Ubuntu** + **Nodejs**). ○ Descargará el estado de código adecuado desde el repositorio. ○ Ejecutará los tests. (Braulio Diez, 2020)	○ Poco claro en la documentación disponible (Braulio Diez, 2020)

4.4 Selección tipos de herramientas.

Teniendo en cuenta la información recopilada, se puede realizar una toma de decisión sobre las herramientas con las que se van a trabajar a lo largo del proyecto, tanto como para planificación y gestión del proyecto, como para el versionamiento del código y para la integración continua.

Es importante destacar que para la elección de dichas herramientas se realizaron varias reuniones con el equipo de trabajo donde se exponían las características, ventajas y desventajas de las mismas además de esto se tomó como punto clave las herramientas y la forma de trabajo que se venían realizando en la empresa. Además de esto se plateo la opción de que dichas herramientas trabajaran en conjunto volviendo más fácil en el ciclo de vida del proyecto.

En la conclusión de las diferentes reuniones se decidió que se iban a trabajar 2 herramientas que cubrían en totalidad con los criterios que se pedían.

Ilustración 4 Jira + Git + Git Actions

La primera herramienta a elegir fue para la creación de un repositorio donde se alojaría el código; esto fue un punto clave ya que dependiendo de este se elegirán las demás herramientas tanto para la gestión de proyectos como para la integración continua.

La herramienta que fue seleccionada por el equipo de trabajo fue Git Hub dado que cuenta con una amplia comunidad, es de código abierto, maneja diferentes tipos de lenguaje, tiene opciones para repositorios públicos o privados sin ningún costo adicional, es una herramienta muy fácil de usar y se adapta por completo a la forma de trabajo de la empresa.

Teniendo en cuenta la herramienta donde se encuentra alojado el código, se procedió a la selección de la herramienta para la gestión del proyecto siendo Jira el software elegido, debido a que este cuenta con un complemento que permite la integración directa con Git Hub, reportando cada commit de cada tarea que se esté trabajando, para esto el programador solo debe de tener cuenta que el mensaje del commit debe de tener como cabecera el código de la tarea que este asignada en Jira.

Por último, se eligió la herramienta de integración continua, seleccionado para esto el nuevo complemento de Git Hub, Git Actions, el cual permite realizar integración y despliegue continuo, aprovechando la ventaja de que ya se estaba trabajando con un repositorio Git, se decidió Que Git Actions fuese la seleccionada para realizar el integración continuo del proyecto.

.5

5 Diseño

Dada las herramientas que se seleccionaron para realizar el proyecto se explicara detalladamente la manera como estas deben trabajar para lograr el objetivo del proyecto, considerando las especificaciones dadas por la empresa.

5.1 Jira

Según la documentación oficial de Jira Software forma parte de una gama de productos diseñados para ayudar a equipos de todo tipo a gestionar el trabajo. En principio, Jira se diseñó como un gestor de incidencias y errores. Sin embargo, se ha convertido en una poderosa herramienta de gestión de trabajo para todo tipo de casos de uso, desde la gestión de requisitos y casos de prueba hasta el desarrollo de software ágil.(*¿Para Qué Sirve Jira? | Atlassian*, n.d.)

Jira Software Cloud proporciona herramientas de planificación y hojas de ruta para que los equipos puedan gestionar a los interesados, los presupuestos y los requisitos de las funciones desde el primer día. Jira se integra en una amplia variedad de herramientas de CI y CD para facilitar la transparencia durante el ciclo de vida de desarrollo de software. Una vez lista para la implementación, aparece la información sobre el estado del código de producción en la incidencia de Jira. Las herramientas integradas de notificación de funciones permiten a los equipos implementar nuevas funciones de forma gradual y segura. (*¿Para Qué Sirve Jira? | Atlassian*, n.d.)

Jira Software está diseñado para que todos los miembros de un equipo de software puedan:

- **Planificar:** planifica sprints y distribuye tareas entre un equipo de software.
- **Supervisar:** Prioriza y analiza el trabajo de un equipo en su contexto y con una completa visibilidad.

- **Lanza:** Realiza lanzamientos con confianza y seguridad, sabiendo que la información que tienes es siempre la más actualizada.
- **Informe:** Mejora el rendimiento del equipo con datos visuales en tiempo real que el equipo puede emplear.

5.1.1 Configuración de Jira

Jira cuenta con una amplia documentación, con las diferentes funcionalidades y herramientas que esta ofrece; para este proyecto en particular se enfocará en las herramientas destinadas para el desarrollo de software.

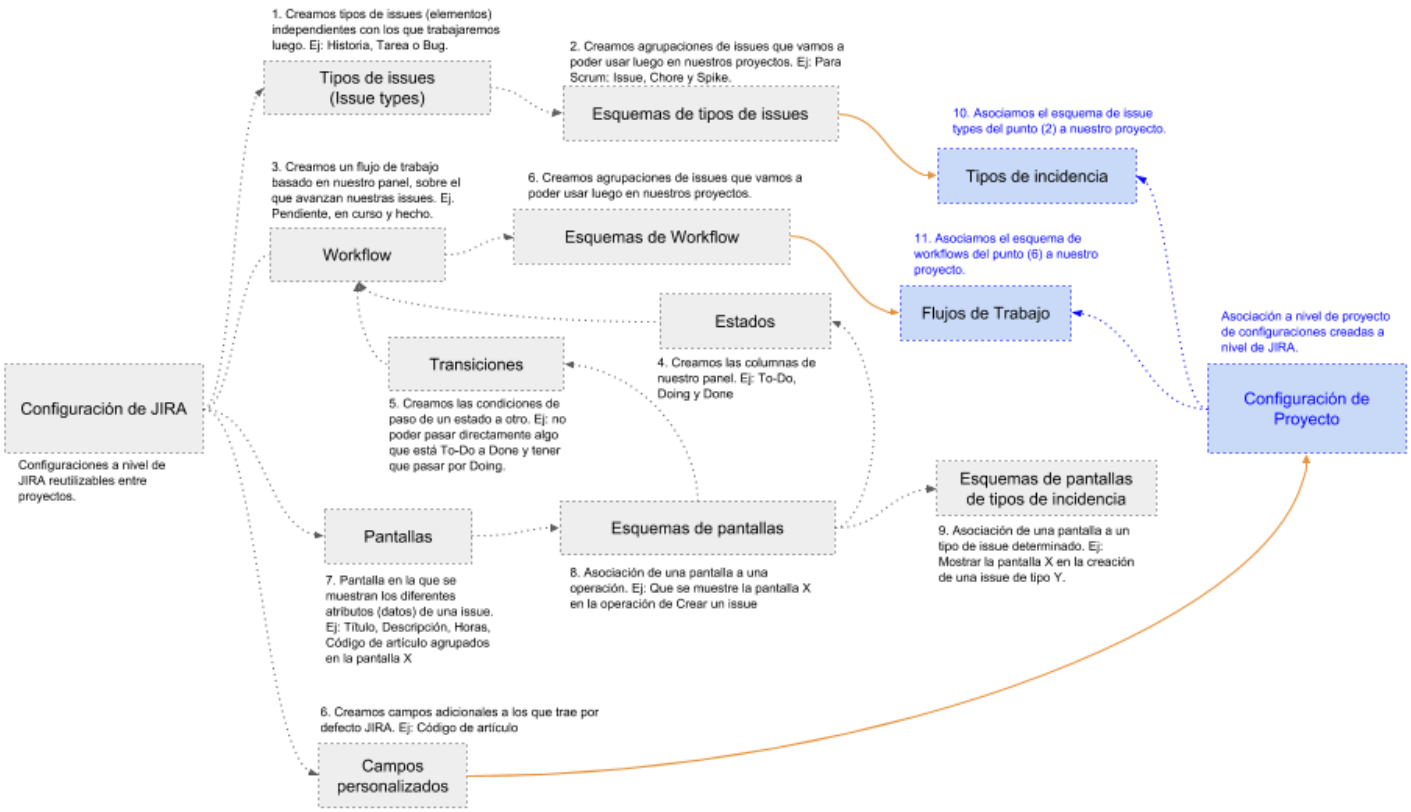


Ilustración 5 Ciclo de trabajo de Jira

Conforme como Jesús Agudelo expone en su artículo un proyecto de Jira este compuesto por (Jesús Angulo, 2019):

- **Incidencias:** realmente son tareas o trabajos (post-its). Todo JIRA es una *issue*, por lo que se emplea los **tipos de incidencias** (*issue types)* para diferenciarlos. Se pueden agrupar varios tipos de incidencias en un bloque denominado **esquema de tipos de incidencias** (*issue types scheme*).
- **Flujos de trabajo:** (*workflows*). Es el «camino» por el que pueden pasar una incidencia, y tenemos la flexibilidad de crear incluso workflows diferentes por tipo de incidencia. Un *workflow* está compuesto por **Estados** y **Transiciones.**
- **Pantalla:** (*screens*). Cuando se interactúa con una *issue* se hace mediante un formulario de JIRA, este formulario es lo que se denomina pantalla. Se puede personalizar esta pantalla para que muestre solo los **campos** necesarios, organizándolos mediante **pestañas** si fuese necesario. Se puede aplicar además pantallas diferentes según la **operación** (forma en la que interactuamos con el issue).

5.1.2 Implementación Jira.

Para este proyecto se decidió que se utilizaría las herramientas básicas de jira que cumple en su totalidad con los requerimientos del trabajo

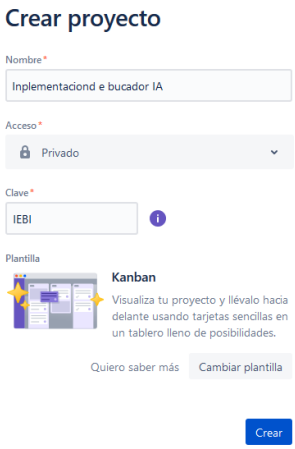


Ilustración 6 Crear proyecto de Jira

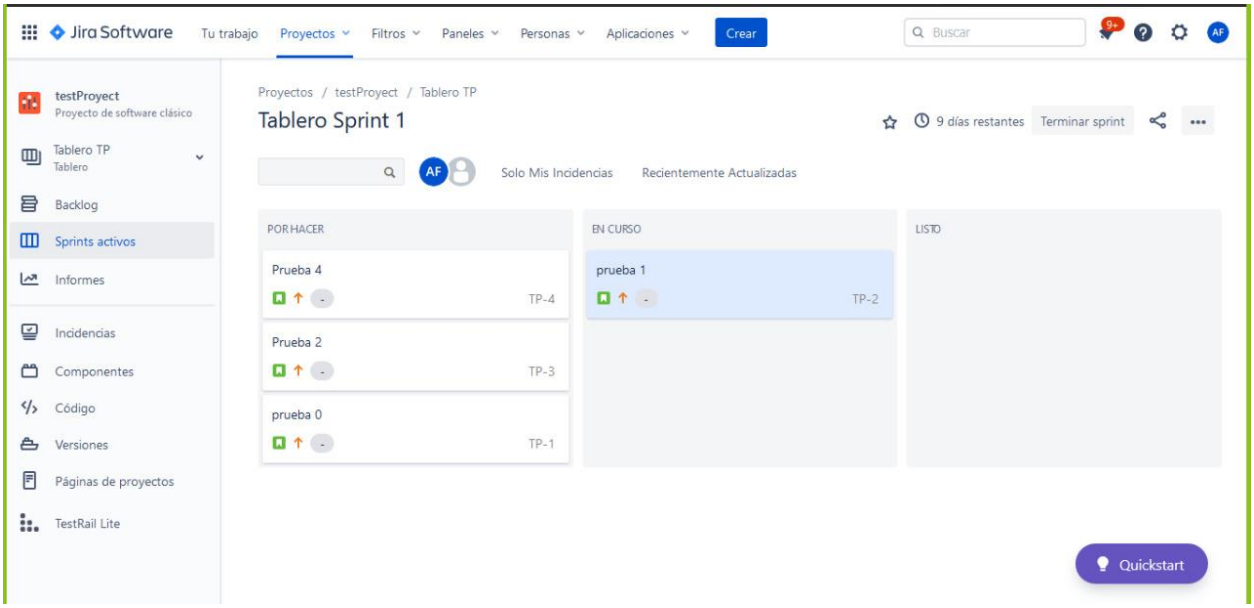


Ilustración 7 Ejemplo Básico de Jira

5.2 Git Hub

Git es un sistema de control de versiones desarrollado en 2005 por Linus Thorvalds, el creador de Linux, y publicado bajo la licencia de software libre GPLv2 de GNU. La particularidad de esta herramienta es que, aunque guarda un **repositorio central** para cada proyecto, todos los participantes descargan una **copia local** del mismo en su propio dispositivo. Cada una de estas copias constituye una copia completa de todo el contenido del repositorio, por lo que no es

necesario estar conectado a la red para trabajar. Además, estos archivos sirven como copia de seguridad en caso de que el repositorio principal falle o resulte dañado. Los **cambios** en los archivos pueden **intercambiarse con todos los demás participantes del proyecto** en cualquier momento y, si corresponde, añadirse al repositorio.(*Tutorial de Git Para Principiantes - IONOS*, 2020)

5.2.1 Comandos Git Hub

Git es un software de código abierto que se puede descargar tanto para Linux, Windows, Mac y Solaris, a continuación, se expondrán los comandos más usados de git, sus funcionalidades y la sintaxis como se debe manejar, estos comandos se tomaron de referencia de un tutorial realizado por Hostinger Tutoriales, la cual ofrecía documentación clara y completa acerca del tema (Gustavo B., 2020).

- **Git config**

  Uno de los comandos más usados en git es git config, que puede ser usado para establecer una configuración específica de usuario, como sería el caso del email, un algoritmo preferido para diff, nombre de usuario y tipo de formato, etc… Por ejemplo, el siguiente comando se usa para establecer un email:

  ```
  git config --global user.email sam@google.com
  ```

- **git init**

  Este comando se usa para crear un nuevo repositorio GIT:

  ```
  git init
  ```

- **git add**

Este comando puede ser usado para agregar archivos al index. Por ejemplo, el siguiente comando agrega un nombre de archivo temp.txt en el directorio local del index:

```
git add temp.txt
```

- **git clone**

Este comando se usa con el propósito de revisar repositorio. Si el repertorio está en un servidor remoto se tiene que usar el siguiente comando:

```
git clone alex@93.188.160.58:/path/to/repository
```

Pero si estás por crear una copia local funcional del repertorio, usa el comando:

git clone /path/ti/repository

- **git commit**

El comando commit es usado para cambiar a la cabecera. Ten en cuenta que cualquier cambio comprometido no afectara al repertorio remoto. Usa el comando:

```
git commit –m “Message to go with the commit here”
```

- **git status**

Este comando muestra la lista de los archivos que se han cambiado junto con los archivos que están por ser añadidos o comprometidos. git status

- **git push**

Este es uno de los comandos más básicos. Un simple push envía los cambios que se han hecho en la rama principal de los respositorios remotos que están asociados con el directorio que está trabajando. Por ejemplo:

```
git push  origin master
```

- **git checouk**

El comando checkout se puede usar para crear ramas o cambiar entre ellas. Por ejemplo, el siguiente comando crea una nueva y se cambia a ella:

```
command git checkout -b <banch-name>
```

Para cambiar de una rama a otra solo usa:

```
git checkout <branch-name>
```

- **git remote**

  El comando git se usa para conectar a un repositorio remoto. El siguiente comando muestra los repositorios remotos que están configurados actualmente:

  ```
  git remote -v
  ```

- **git branch**

  Este comando se usa para listar, crear o borrar ramas. Para listar todas las ramas se usa:

  ```
  git Branch
  ```

  Para borrar la rama:

  ```
  git branch -d <branch-name>
  ```

- **git pull**

  Para poder fusionar todos los cambios que se han hecho en el repositorio local trabajando, el comando que se usa es:

  ```
  git pull origin <branch-name>
  ```

- **git merge**

  Este comando se usa para fusionar una rama con otra rama activa:

  ```
  git merge <branch-name>
  ```

- **git diff**

Este comando se usa para hacer una lista de conflictos. Para poder ver conflictos con el archivo base usa:

```
git diff --base <file-name>
```

- **git log**

  Ejecutar este comando muestra una lista de commits en una rama junto con todos los detalles. Por ejemplo:

```
commit 15f4b6c44b3c8344caasdac9e4be13246e21sadw
Author: Alex Hunter alexh@gmail.com
```

- **git reset**

  Para resetear el index y el directorio que está trabajando al último estado comprometido se usa este comando:

```
git reset - -hard HEAD
```

- **git stash**

  Este es uno de los comandos menos conocidos, pero ayuda a salvar cambios que no están por ser comprometidos inmediatamente, pero temporalmente:

```
git stash
```

- **git show**

  Se usa para mostrar información sobre cualquier objeto git. Por ejemplo:

```
git show
```

- **git fetch**

Este comando le permite al usuario buscar todos los objetos de un repositorio remoto que actualmente no reside en el directorio local que está trabajando. Por ejemplo:

```
git fetch origin
```

5.2.2 Implementación de GitHub

Para la implementación de Git Hub en la empresa como repositorio de código, se decidió separar el proyecto en 2 repositorios, en el primer repositorio se encuentra todo el código relacionado a desarrollo móvil con react native, y en el segundo repositorio el desarrollo web con react, en ambos repositorios cada programador cuenta con su propia rama en la que se trabaja en las correspondientes funcionalidades asignadas.

En cada uno de estos repositorios se tienen ciertas reglas de cómo se debe trabajar para que no existan choques entre ramas o errores en la rama principal

- **Rama Master:** Ninguno de los programadores puede trabajar sobre esta rama, ya que es donde se encuentra el proyecto principal.
- **Rama Dev:** cada desarrollador puede sacar su propia rama y trabajar en lo que corresponde,
- Si existen conflictos se deben corregir generalmente el encargado de esta tarea es el líder del proyecto o de la integración continua
- Si una funcionalidad esta correctamente implementada y probada se deberá integrar inmediatamente con master, para que la rama principal siempre este actualizada

5.3 Git Actions

Tal como lo define la documentación oficial de Git, las acciones de GitHub ayudan a automatizar tareas dentro del ciclo de vida de tu desarrollo de software. Las acciones de GitHub están controladas por eventos, lo que significa que puede ejecutar una serie de comandos después de que se haya producido un evento específico.

Lo primero de todo es crear una carpeta en la cual se añadirá el fichero de configuración del workflow. Esta carpeta tiene por norma estar ubicada en github/workflows/ en el cual se creará un archivo workflow.yml en donde se encontrará todos los eventos el cual activa automáticamente el flujo de trabajo, que contiene un trabajo. Luego, el job usa pasos para controlar el orden en que se ejecutan las acciones. Estas acciones son los comandos que automatizan las pruebas de software y crean artefactos.(*Introduction to GitHub Actions - GitHub Docs*, n.d.)

5.3.1 Componente de Git Hub Actions

- **Workflows:** El flujo de trabajo es un procedimiento automatizado que agrega a su repositorio. Los flujos de trabajo se componen de uno o más trabajos y pueden ser programados o activados por un evento. El flujo de trabajo se puede usar para construir, probar, empaquetar, lanzar o implementar un proyecto en GitHub.
- **Events:** Un evento es una actividad específica que desencadena un flujo de trabajo. Por ejemplo, la actividad puede originarse en GitHub cuando alguien envía una confirmación a un repositorio o cuando se crea un problema o una solicitud de extracción. También puede utilizar el webhook de envío del repositorio para activar un flujo de trabajo cuando se produce un evento externo.
- **Jobs:** Un trabajo es un conjunto de pasos que se ejecutan en el mismo runner. De forma predeterminada, un workflows con varios jobs ejecutará esos jobs en paralelo. Además se

puede configurar un flujo de trabajo para ejecutar trabajos de forma secuencial. Por ejemplo, un flujo de trabajo puede tener dos trabajos secuenciales que compilan y prueban código, donde el trabajo de prueba depende del estado del trabajo de compilación. Si el trabajo de compilación falla, el trabajo de prueba no se ejecutará.

- **Steps:** Un paso es una tarea individual que puede ejecutar comandos (conocidos como acciones). Cada paso de un trabajo se ejecuta en el mismo corredor, lo que permite que las acciones de ese trabajo compartan datos entre sí.
- **Actions:** Las acciones son comandos independientes que se combinan en pasos para crear un trabajo. Las acciones son el bloque de construcción portátil más pequeño de un flujo de trabajo. Puede crear sus propias acciones o utilizar acciones creadas por la comunidad de GitHub. Para usar una acción en un flujo de trabajo, debe incluirla como un paso.
- **Runners:** Un corredor es un servidor que tiene instalada la aplicación de ejecución de acciones de GitHub. Puede utilizar un corredor alojado en GitHub, o puede alojar el suyo propio. Un corredor escucha los trabajos disponibles, ejecuta un trabajo a la vez e informa el progreso, los registros y los resultados a GitHub. Para los ejecutores alojados en GitHub, cada trabajo de un flujo de trabajo se ejecuta en un entorno virtual nuevo.

5.3.2 Implementación de Git Hub Actions en el proyecto.

En esta sección se mostrará la configuración que se usó para realizar integración continua, dicha configuración se realizó en un archivo. ymal el cual se encuentra ubicado en la ruta que sugerida la documentación oficial de git hub Actions.

El siguiente flujo de trabajo realizará una instalación limpia de las dependencias del nodo, creará el código fuente y ejecutará pruebas en diferentes versiones del nodo, estas acciones se ejecutarán cada vez que se quiera subir a cualquiera de las ramas

```yaml
name: Node.js CIL

on:
  push:
    branches: [ adriana, testBranch ]
  pull_request:
    branches: [ adriana, testBranch  ]
jobs:
  build:

    runs-on: ubuntu-latest

    strategy:
      matrix:
        node-version: [10.x, 12.x, 14.x]

    steps:
    - uses: actions/checkout@v2
    - name: Use Node.js ${{ matrix.node-version }}
      uses: actions/setup-node@v1
      with:
        node-version: ${{ matrix.node-version }}
    - run: npm i
    - run: npm run build
    - run: npm test
```

Ilustración 8 Ejemplo básico de flujo de trabajo git Actions

5.4 Consejos para implantar Integración continua.

Cuando se trabaja de este modo, deben acatarse una serie de reglas. En la mayoría de las ocasiones, los programadores se rigen por los principios que **Martin Fowler** describió para llevar a cabo con éxito una integración continua. En primer lugar, se ha de garantizar que todos los implicados se encuentran al mismo nivel, y que no hay nadie cuyo comportamiento pueda causar problemas.(*Integración Continua | Definición | Ventajas e Inconvenientes - IONOS*, 2019)

- **Un único código fuente:** Aunque parezca obvio, se trata de uno de los factores más importantes, ya que todos los integrantes del equipo deberán utilizar la misma herramienta,

es decir, el mismo repositorio, cuando trabajen en el código. Y esto no solo se aplica al código fuente. Para que funcionen las aplicaciones, se necesitan otros elementos como, por ejemplo, bases de datos, que también deberán estar contenidos en el mismo lugar. Por ello, Martin Fowler recomienda construir un repositorio de tal forma que cualquier ingeniero que se incorpore al proyecto con un equipo nuevo encuentre todos los archivos necesarios en un único lugar.

- **Automatizar la compilación del proyecto:** La obtención de un proyecto funcional a partir de un código fuente implica la compilación del mismo, actualizar bases de datos y mover ficheros de un sitio a otro. Todas estas tareas pueden automatizarse y debería ser posible ejecutar la compilación con un único comando.
- **Sistemas que realizan sus propias pruebas:** Un equipo podrá beneficiarse de aún más automatización y rapidez a través de la integración continua si se incorporan mecanismos de prueba en el proceso de compilación ("build"). Al igual que la propia compilación, las pruebas podrán llevarse a cabo en poco tiempo, y lo ideal será implementar un plan completo de mecanismos de prueba.
- **Integración Diaria:** Un proceso de continuous integration funcionará únicamente si todos los miembros del equipo respetan el sistema. En el momento en que algún miembro del equipo no integre su código en la línea principal, el resto de compañeros partirá de una premisa falsa. Todos los desarrolladores asumen que trabajan en un sistema estable, pero si alguien tarda más de un día en integrar su código y continúa trabajando en él, al final la búsqueda de errores podría convertirse en un verdadero problema. Asimismo, la comunicación también es un factor importante en la integración continua, ya que, si los

desarrolladores se mantienen al tanto, las pequeñas dificultades podrán aclararse con mayor rapidez.

- **Main line Operativo:** El código de la línea principal deberá probarse continuamente y encontrarse siempre operativo, por lo que los desarrolladores deberán construir el proyecto aquí y no solo en su copia local. En este contexto, todos deberán preocuparse también de que sus aportaciones sean válidas y de que su funcionamiento sea correcto, de modo que todos tendrán que comprobar el código y el build. Si aparece algún fallo, tendrán que solventarlo y garantizar que el código no contiene errores.
- **Reparación Inmediata:** Lo más importante de la integración continua es que no quede ninguna versión defectuosa en la línea principal, lo que implica que la solución de los fallos no podrá posponerse. En palabras de Martin Fowler, no existe ningún problema si los builds no funcionan y el código debe cambiarse, pero el sistema requiere que la reparación se lleve a cabo de inmediato. Todos los desarrolladores deben poder partir del hecho de que el código de la línea principal funciona correctamente, de lo contrario, podrían estar trabajando sobre un código defectuoso que acabará desencadenando una oleada de fallos.
- **Integración Rápida:** La integración completa del proyecto (incluida la fase de prueba) debería realizarse lo más rápido posible. La programación extrema (extreme programming, XP) prevé solo 10 minutos para esto. Puesto que un desarrollador debe realizar varias integraciones diarias, si no se establecieran mecanismos para acelerar el proceso se perdería una gran cantidad de tiempo. Para que no se demore demasiado, debe descartarse la idea de realizar todas las pruebas posibles directamente y aplicarse en lugar de ello un sistema de dos fases: en la primera fase, se realizan pruebas en las que la compilación del proyecto

pueda ser rápida. La segunda fase durará varias horas y en ella se llevarán a cabo pruebas más exhaustivas.

- **Pruebas en una réplica del entorno de producción:** Las pruebas deberán realizarse en un entorno seguro y con una configuración exactamente igual que la del entorno de producción. Bajo determinadas circunstancias, esto podría resultar muy costoso, pero la virtualización de los equipos hará que el factor de los costes se reduzca.
- **Accesibilidad:** Todos los involucrados en el desarrollo de un determinado software deberían poder obtener fácilmente el último ejecutable del programa y ejecutarlo. La implementación de este aspecto es relativamente sencilla, ya que la integración continua exige que todos los archivos se encuentren en un único repositorio que todos conocen. De este modo, se puede comenzar a realizar pruebas adicionales en el proceso de programación, los accionistas pueden utilizar archivos ejecutables con fines demostrativos y los directores de calidad pueden examinar las cifras.
- **Buena Comunicación:** No solo resulta importante que todos los implicados tengan acceso al código fuente y puedan ejecutar el archivo, sino que, además, debe quedar constancia de quién ha realizado qué modificación. Asimismo, los desarrolladores deben informar cuando se encuentren en un proceso de compilación, para lo que algunos equipos utilizan elementos visuales que indican que se está trabajando en la integración.

5.5 Diagrama de flujo de implementación integración continua en la empresa BeGo

A continuación, se mostrará en detalle cómo debería comportarse el ciclo de vida del proyecto, y su correspondiente descripción por actividad

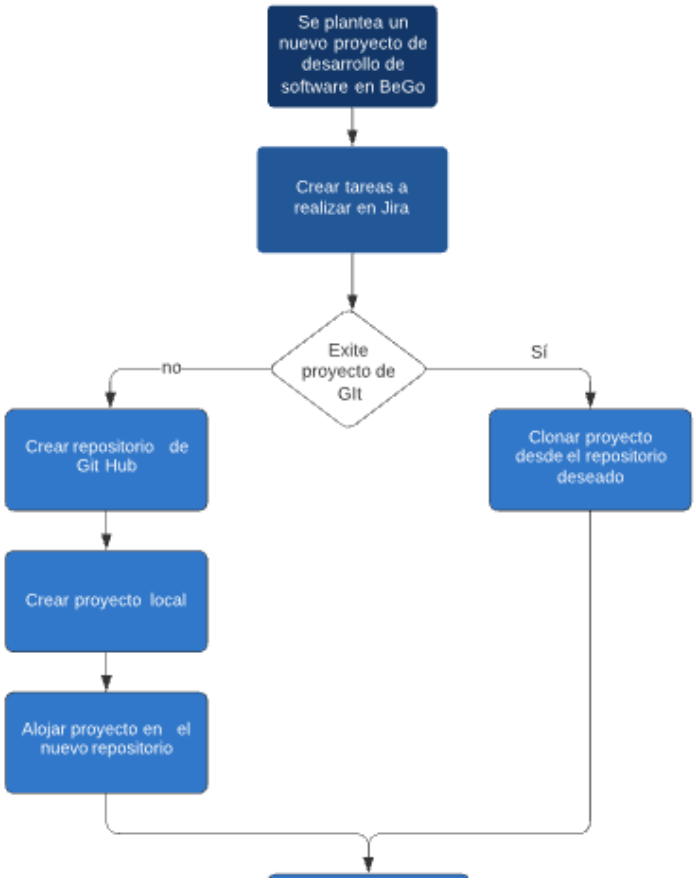


Ilustración 9 Diagrama de flujo de implementación integración continua en la empresa BeGo1

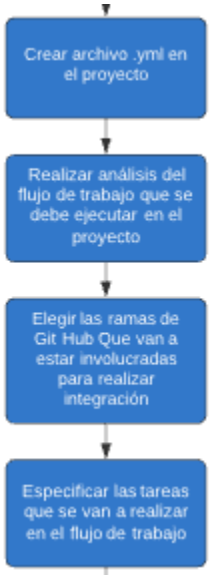


Ilustración 10 Diagrama de flujo de implementación integración continua en la empresa BeGo2

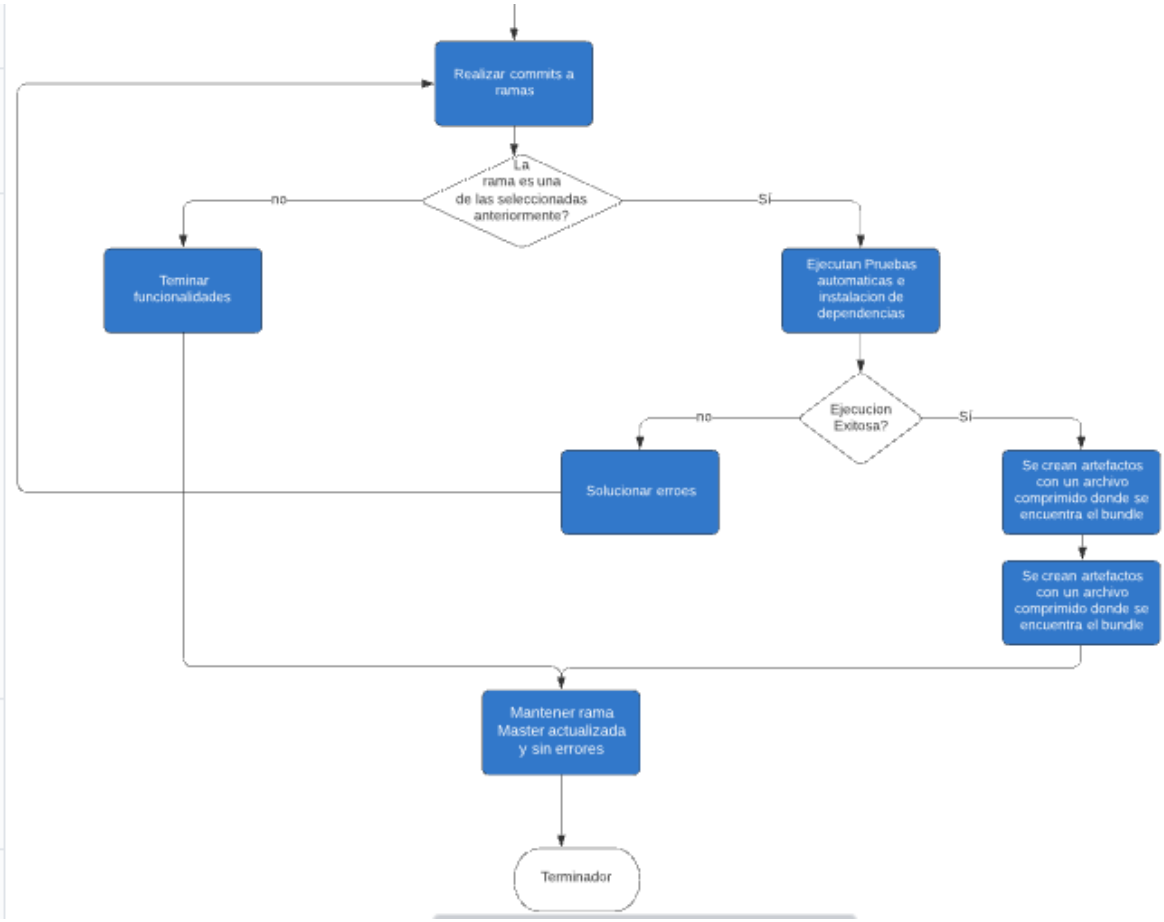


Ilustración 11Diagrama de flujo de implementación integración continua en la empresa BeGo3

ACTIVIDAD	DESCRIPCION
Se plantea un nuevo proyecto de desarrollo de software en BeGo	Se plantea un nuevo proyectó o nueva funcionalidad que se este necesitando en la empresa BeGo
Crear tareas a realizar en Jira	Se divide el proyecto en tareas que se deben realizar y se le asignan a cada desarrollador, con su respectivo nivel de importancia en el tablero de Jira.
Crear repositorio de Git Hub	Se crea un nuevo repositorio si este lo amerita, o si es un proyecto con nuevas funcionalidades.
Crear proyecto o clonarlo desde un repositorio	SI el proyecto ya se encuentra alojado en un repositorio se clona y se guarda localmente, de la contrario se crea un proyecto local
Alojar proyecto en el nuevo repositorio	Si el proyecto es nuevo, se realizará la correspondiente conexión con El nuevo repositorio de git Hub
Crear archivo .yml en el proyecto	Se creará un archivo. yaml en la ruta que nos recomienda la documentación de Git Action, en este archivo estará todas tareas que se realizaran al momento de realizar las acciones especificadas
Realizar análisis del flujo de trabajo que se debe ejecutar en el proyecto	Se realizará un análisis de las tareas que se deben de ejecutar, tratando de ahorrar la cantidad de tiempo en ejecuciones posibles, dado que solo se tienen 2000 minutos de ejecución gratuitos
Elegir las ramas de Git Hub Que van a estar involucradas para realizar integración	Dado que se cuenta con una limitación de tiempo se debe realizar un análisis de las ramas que deben de ejecutar las acciones
Especificar las tareas que se van a realizar en el flujo de trabajo	Especificar las tareas que se van a realizar cada vez que se ejecute el archivo, teniendo en cuanta las necesidades del proyecto
Realizar Commits a las ramas de proyecto con sus respectivas funcionalidades	Cada desarrollador debe de realizar sus correspondiente commits cada vez que terminen una funcionalidad.
Realizar integración diaria	Los respectivos commits deben de realizarse por lo menos una vez al día, por cada desarrollador
Mantener rama Master actualizada y sin errores	La rama principal debe de mantenerse actualizada con cada una de las tareas del proyecto y sin errores es decir esta rama también debe de actualizarse por lo menos una vez al día

6 Validación

Para validar todo el procedimiento que comprende integración continua se mostrara los resultados y los avances del proyecto con el trabajo conjunto de las aplicaciones elegidas, para poder observar la diferencia entre el modo de desarrollo del proyecto anterior con el actual se realizara una comparativa de estos procesos con el antes y el después.

6.1 Validación de proceso de asignación de tareas

En la empresa BeGo el proceso de asignación de tareas se realizaba a través de una aplicación llamada Trello la cual permitía que el equipo organizara las tareas y se priorizaran los proyectos, en esta herramienta se asignaban tareas de cualquier índole ya sea de desarrollo o logística, causando desorden y conflicto entre las mismas tareas.

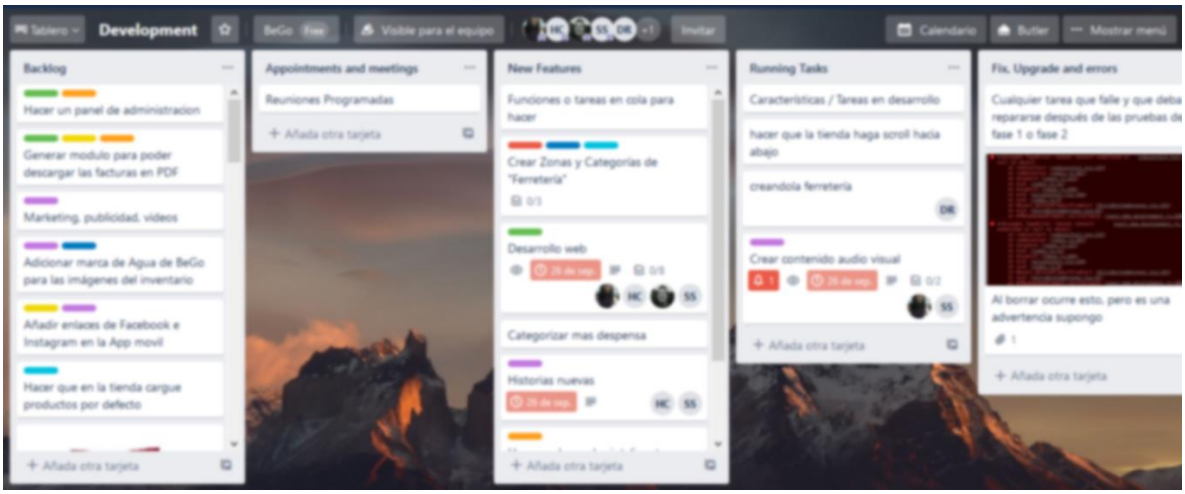


Ilustración 12 Tablero de trello

Para poder mejorar este proceso en la empresa se realizó una investigación de las diferentes herramientas que existe para asignación de tareas sus ventajas y funcionalidades que están ofrecían. En esta investigación se encontró con la herramienta de Jira la cual permite dividir las tareas del proyecto de una manera más organizada, además de que se puede integrar con distintas

herramientas como lo es Git Hub, esta fue una de las ventajas que fue crucial para la toma de decisiones ya que reportaba en cada tarea los commits que se realizaban de cada funcionalidad y la persona que lo realizaban, dando mayor control y orden sobre el proyecto en el cual se está trabajando. Además, esta herramienta ofrece una herramienta estadística en la cual se permite visualizar los avances o problemas que se presentaron en el ciclo de vida del proyecto.

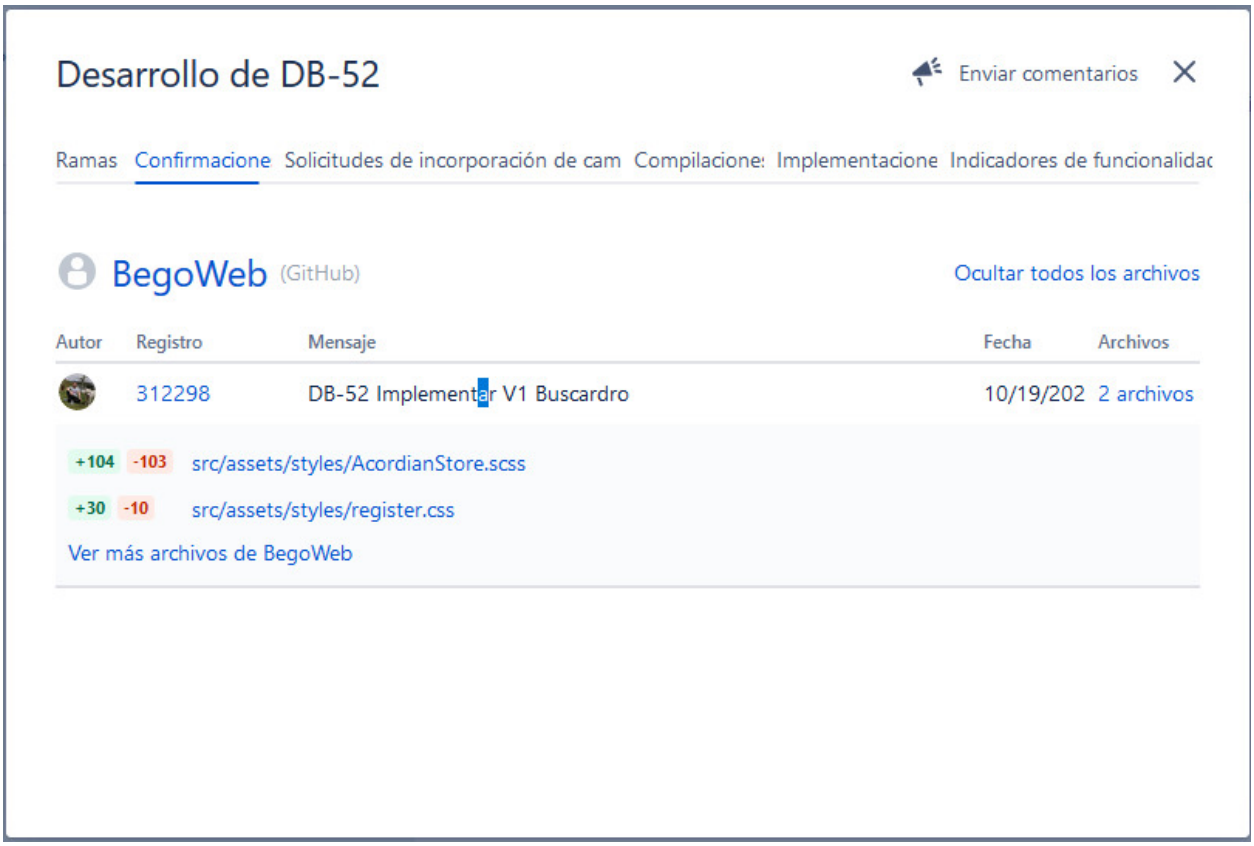


Ilustración 13 Visualización de un commit de Git en Jira

6.2 Validación de proceso de gestión de versiones

El proceso de gestión de versiones es un proceso fundamental en el ciclo de vida de un proyecto, permite llevar un control de todas las versiones de código que se están llevando a cabo. El equipo de software antes llevaba un proceso un poco ortodoxo utilizando la herramienta Live Share donde todos los desarrolladores compartían el mismo código en tiempo real, y cada uno trabajando en

funcionalidades diferentes, esto provocaba conflictos a la hora de realizar pruebas al código, además de que los errores era difíciles encontrar y resolver y no se tenían copias del proyecto.

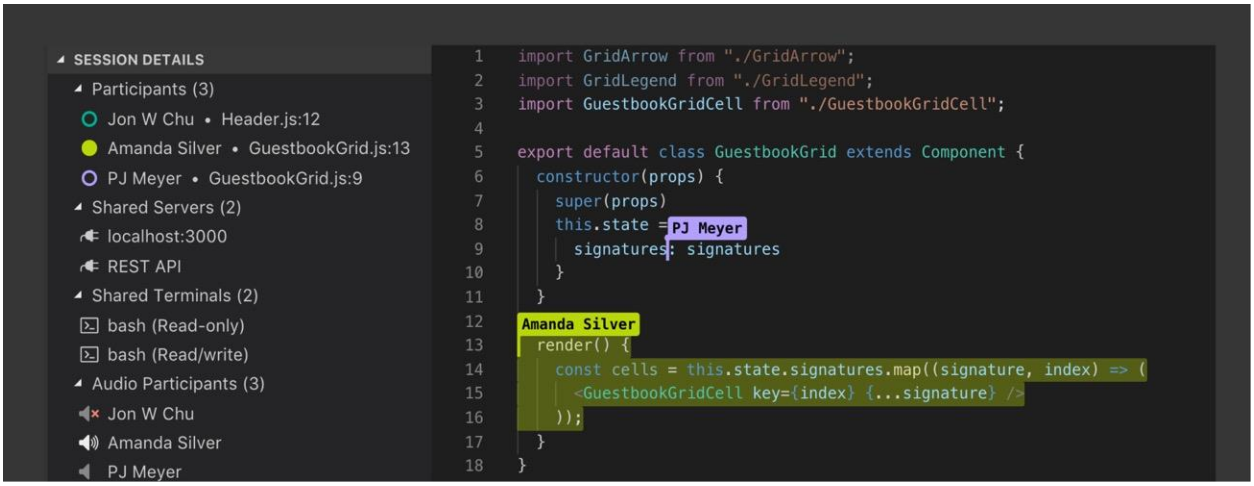


Ilustración 14 Ejemplo de función de live share

Viendo la gran cantidad de problemas que existían se decidió trabajar una herramienta de versiona miento de código encontrándonos con una herramienta muy útil y que se aptaba a la perfección a las necesidades que tenía la empresa en el momento. Antes de implementar esta herramienta se realizó un investigación de las maneras que trabajaba y algunas características que se deben de tener en cuenta para que el versiónamiento de código se lleve de manera correcta, lo primero que se realizó para realizar la implementación fue crear un nuevo repositorio donde iba estar alojado nuestro código, el código principal se encontraría en la rama principal (‘Master’), y para el resto de programadores se creó una rama totalmente independiente donde pueden trabajar en las correspondientes funcionalidades.

Con esta herramienta se logró detectar de una manera más fácil y rápida los errores que se presentaban en el código, permitiendo que se resuelvan de una manera más eficaz.

Ilustración 15 Brach de Repositorio de Git en BeGo

6.3 Validación de proceso de IC

Al no tener ni un repositorio para el versionamiento, tampoco contaban con ningún tipo de proceso para integración continua; todos los procesos que esto conllevaba se realizaban de manera manual y poco efectiva, al implementar la herramienta de Git Hub se decidió probar un nuevo complemento de esta que ofrece, Git Actions la cual permitió automatizar el proceso de integración continua, de manera rápida y sencilla sin necesidad de buscar una herramienta adicional para la realización de este proceso.

En la configuración de Git Actions se realizó un flujo de trabajo que contenía 4 tareas donde cada una de ella ejecutaba una acción diferente, en la primera tarea se comprueba si el flujo de trabajo puede acceder al repositorio

```
- name: Checkout repository
  uses: actions/checkout@v2
```

Ilustración 16 Acceso al repositorio

El resultado de Git Actions para esta tarea es:

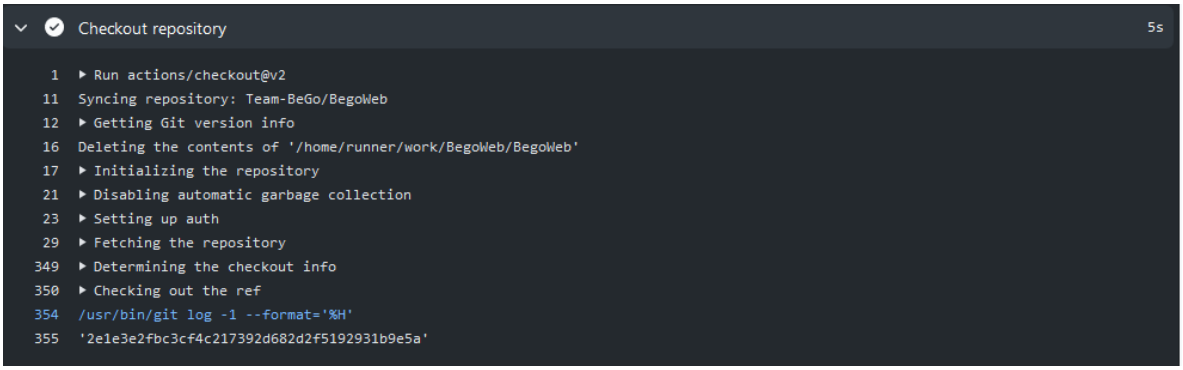


Ilustración 17 Resultado acceso al repositorio

La segunda tarea se encarga de realizar una instalación limpia de las dependencias, crear el código fuente.

```
- name: npm install, build, and test
  run: |
    npm install
    npm run build --if-present
```

Ilustración 18 instalación de dependencia

La tercera tarea es la encargada de ejecutar los test, esto se ejecuta con el comando npm test

```
-name: Run npm test
  run: |
    npm test
```

Ilustración 19 Ejecución de test

Como resultado de esta tarea:

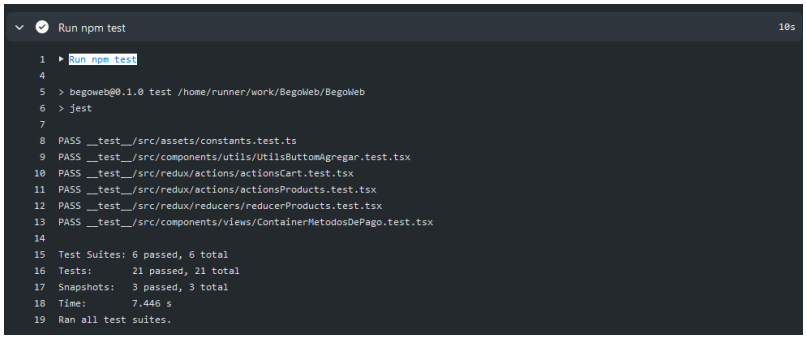


Ilustración 20 Resultado de la ejecución del test

La cuarta tarea se encarga de crear los artefactos lo que le permite compartir datos entre trabajos y almacenar datos una vez que se completa un workflow, en este caso en específico creía un archivo comprimido donde se va a encontrar el bundle listo para pasar a la etapa de producción.

```yaml
- name: Archive production artifacts
  uses: actions/upload-artifact@v2
  with:
    name: dist-without-markdown
    path: |
      dist
      !dist/**/*.md
```

Ilustración 21 Crear artefacto con el bundle

Como resultado de esta tarea se tiene:

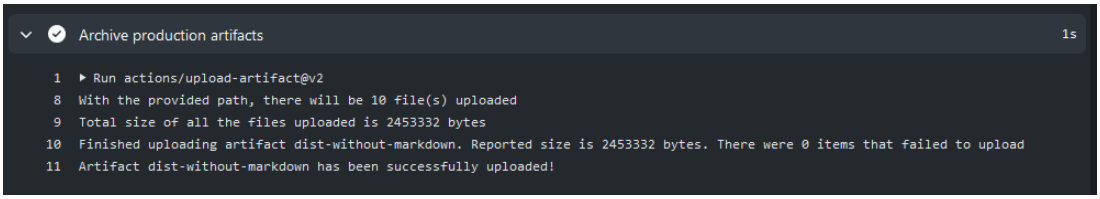


Ilustración 22 Resultado de creación de artefacto con el bundle

Ilustración 23 Resultado de creación de artefacto con el bundle 2

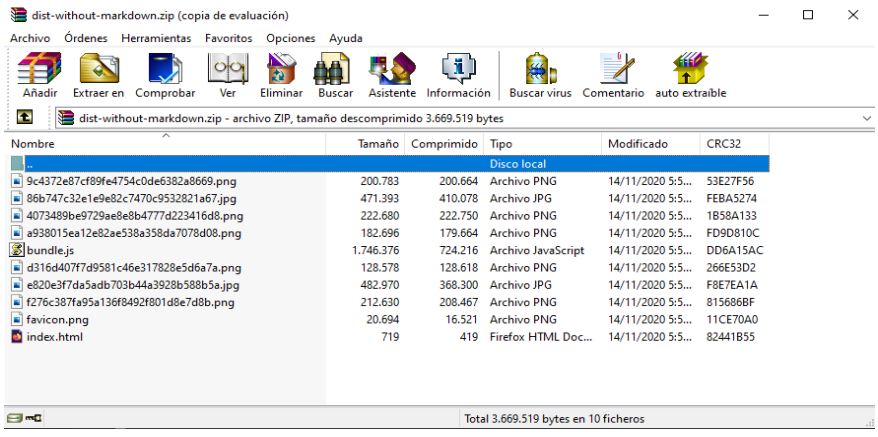


Ilustración 24 Archivo zip con el bundle

7 Conclusión

- Se logro realizar una ardua investigación acerca del proceso de automatización de integración continua, con diferentes implementaciones y aplicaciones, lo cual contribuyo a una mejor toma de decisiones analizando las características de cada uno de los procesos.
- Con el estudio de las herramientas de integración continua se determinó en base a las, ventajas, desventajas y características de las mismas que herramienta se adaptaba a las necesidades de la empresa, eligiendo Git Actions como la herramienta con la cual se trabajó por fu fácil adaptabilidad al entorno en el que operaba el grupo de desarrolladores de BeGo.
- Se diseño un procedimiento de integración continua el cual permitió automatizar, ahorrar tiempo y mejorar el ciclo de vida del proyecto además de que este procedimiento puede ser aplicado en cualquier otro proyecto de la empresa BeGo.
- Se realizo una validación exitosa del proceso de integración continua con el proyecto de prototipo de búsqueda basado en IA, arrojando mejores resultados en cuanto tiempo y precisión de los resultados.

8 Recomendaciones y trabajos futuros

- Se propone que se exploren otras herramientas que ofrece Git, como lo es Git Lab la cual también ofrece grandes ventajas.
- Se recomienda que el proceso de automatización continúe con despliegue continuo y entrega continua, ya que estas funcionalidades pueden aportar mayor eficiencia a la empresa y el proyecto.
- Se propone realizar un estudio para que el proceso de integración continua se ejecute de forma local para evitar costos en repositorios remotos.

9 Bibliografía

.

¿Para qué sirve Jira? | Atlassian. (n.d.). Retrieved November 16, 2020, from https://www.atlassian.com/es/software/jira/guides/use-cases/what-is-jira-used-for#jira-for-task-management

¿Qué es Elasticsearch? | Elastic. (n.d.). Retrieved November 16, 2020, from https://www.elastic.co/es/what-is/elasticsearch

Adrian Hernández Yeja. (2015, June). *(PDF) APLICACIÓN DEL PROCESO DE INTEGRACIÓN CONTINUA EN EL CENTRO DE TELEMÁTICA DE LA UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS*. https://www.researchgate.net/publication/302010909_APLICACION_DEL_PROCESO_DE_INTEGRACION_CONTINUA_EN_EL_CENTRO_DE_TELEMATICA_DE_LA_UNIVERSIDAD_DE_LAS_CIENCIAS_INFORMATICAS

Agustín González, Daniel Vergara C., & Rodrigo Yañez Q. (n.d.). *Sistema de Control de Versiones "CVS."* Retrieved November 16, 2020, from http://profesores.elo.utfsm.cl/~agv/elo330/2s03/projects/CVS/CVS.PDF

Ana María García Orozco. (2015). *LA INTEGRACIÓN CONTINUA Y SU APORTE AL ASEGURAMIENTO DE LA*.

ANA MARÍA GARCÍA OROZCO. (2015). *LA INTEGRACIÓN CONTINUA Y SU APORTE AL ASEGURAMIENTO DE LA*.

Anaraya Albornoz. (2020, October). *Planificación de proyecto con Asana*.

Andrea Garcia. (2019). *Las herramientas de versionamiento de código | by Andrea Garcia |*

Medium. https://medium.com/@andreagarcia_94938/las-herramientas-de-versionamiento-de-código-8a96a88d1702

Belén Soto Lull. (2016). *ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES TRABAJO FIN DE MÁSTER*.

Braulio Diez. (2020, February 12). *Hola Docker CI / CD - GitHub Actions — Lemoncode formacion*. https://lemoncode.net/lemoncode-blog/2020/2/12/hola-docker-ci-cd-github-actions

Claudia Márquez. (2020, May 7). *Construyendo un flujo CI/CD para Laravel con Github Actions – Styde.net*. https://styde.net/construyendo-un-flujo-ci-cd-para-laravel-con-github-actions/

Cody Arsenault. (2017, October 12). *Jenkins vs Travis - Comparing Two Popular CI Tools - KeyCDN*. https://www.keycdn.com/blog/jenkins-vs-travis

Cole Thienes. (2019, December 4). *How we built an AI-powered search engine (without being Google) | by Cole Thienes | Towards Data Science*. https://towardsdatascience.com/how-we-built-an-ai-powered-search-engine-without-being-google-5ad93e5a8591

Damián Cervantes Rodón. (2010, September). *(PDF) Propuesta de entorno de integración continua en el Centro de Informatización Universitaria.* https://www.researchgate.net/publication/268445965_Propuesta_de_entorno_de_integracion_continua_en_el_Centro_de_Informatizacion_Universitaria

Federico Toledo. (2018, June 21). *Travis-CI para integración continua - Federico Toledo*. https://www.federico-toledo.com/travis-ci-para-integracion-continua/

Gabriel Fernando Chiriboga Rogel. (2013, October 25). *Sistemas de Control de Versiones*.

https://portfoliogabrielfcr.wordpress.com/2013/10/25/62/

Git vs. SVN: una comparativa del control de versiones - IONOS. (2020). https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/git-vs-svn-una-comparativa-del-control-de-versiones/

GitHub - microsoft / MSMARCO-Question-Answering: MS MARCO (Microsoft Machine Reading Comprehension) es un conjunto de datos a gran escala centrado en la comprensión de lectura de la máquina y la respuesta a preguntas. (2020, March 27). https://github.com/microsoft/MSMARCO-Question-Answering

GitHub vs. GitLab | GitLab. (n.d.). Retrieved November 16, 2020, from https://about.gitlab.com/devops-tools/github-vs-gitlab/

GitHub vs. GitLab vs. Bitbucket: GitLab. (n.d.). Retrieved November 16, 2020, from http://www.veprof.com/top-three-repositories-gitlab.html

GitLAb Flujo de trabajo dinámicos en un solo lugar. (2020). https://www.grupodot.com/images/Ebook-Interactivo-GitLab.pdf

Gustavo B. (2020, July 30). *Comandos Básicos De GIT - Guía Completa*. https://www.hostinger.co/tutoriales/comandos-de-git

Integración continua | Definición | Ventajas e inconvenientes - IONOS. (2019, March 19). https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/integracion-continua/

Integración continua del software | Pruebas automatizadas | AWS. (n.d.). Retrieved November 16, 2020, from https://aws.amazon.com/es/devops/continuous-integration/

Introduction to GitHub Actions - GitHub Docs. (n.d.). Retrieved November 16, 2020, from

https://docs.github.com/en/free-pro-team@latest/actions/learn-github-actions/introduction-to-github-actions

Javier Gobea. (2018). *19 herramientas de gestión de proyectos para emprendedores digitales*. https://hormigasenlanube.com/herramientas-de-gestion-de-proyectos/

Javier Guillot. (2019, October). *Uso de Trello como herramienta para la gestión de tareas | by Equipo de Innovación Pública (EiP) | Medium*.

Jenkins. (n.d.). Retrieved November 16, 2020, from https://www.jenkins.io/

Jenkins vs. GitLab | GitLab. (n.d.). Retrieved November 16, 2020, from https://about.gitlab.com/devops-tools/jenkins-vs-gitlab/

Jesús Angulo. (2019, January 14). *Cómo crear un proyecto personalizado desde cero en JIRA Cloud con un Workflow Clásico - Adictos al trabajo*. https://www.adictosaltrabajo.com/2019/01/14/como-crear-un-proyecto-personalizado-desde-cero-en-jira-cloud-con-un-workflow-clasico/

José Manuel Alarcón. (2020, June 1). *Qué es Git, ventajas e inconvenientes y por qué deberías aprenderlo (bien) | campusMVP.es*. https://www.campusmvp.es/recursos/post/que-es-git-ventajas-e-inconvenientes-y-por-que-deberias-aprenderlo-bien.aspx

La importancia de una barra de búsqueda inteligente - IONOS. (2017, November 17). https://www.ionos.es/digitalguide/paginas-web/creacion-de-paginas-web/la-importancia-de-una-barra-de-busqueda-inteligente/

Las mejores herramientas de integración continua - IONOS. (2019, March 20). https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/herramientas-de-integracion-

continua/

Las mejores herramientas de integración continua - IONOS. (2020, March 19). https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/herramientas-de-integracion-continua/

Mello Teggia, M., & Tula, P. (2015). *Introducción Actividad I: Bitbucket Comandos básicos de hg Actividad II: clonar repositorio y agregar archivo Actividad III: Pullea Mercurial, sistema de control de versiones LABI Cursos*.

Mercurial (sistema de control de versiones) - EcuRed. (n.d.). Retrieved November 16, 2020, from https://www.ecured.cu/Mercurial_(sistema_de_control_de_versiones)#Ventajas

Prefacio. (n.d.). Retrieved November 16, 2020, from https://tortoisesvn.net/docs/release/TortoiseSVN_es/tsvn-preface.html

Qué es Git: conviértete en todo un experto en Git con esta guía. (n.d.). Retrieved November 16, 2020, from https://www.atlassian.com/es/git/tutorials/what-is-git

Rosa MaDurante Lerate, Pablo Recio Quijano, Leandro Pastrana González, & Noelia Sales Montes. (n.d.). *Sistemas para el Control de Versiones - PDF Free Download*. Retrieved November 16, 2020, from https://docplayer.es/9581276-Sistemas-para-el-control-de-versiones.html

Salamon, A., Maller, P., Boggio, A., Mira, N., Perez, S., & Coenda, F. (n.d.). *La Integración Continua Aplicada en el Desarrollo de Software en el Ámbito Científico-Técnico*.

Soluciones: Buscadores Inteligentes: 3.14. (n.d.). Retrieved November 16, 2020, from http://www.3.14financialcontents.com/buscadores-inteligentes/

Sparck Jones, K., Walker, S., & Robertson, S. E. (2000). Probabilistic model of information retrieval: Development and comparative experiments. Part 1. *Information Processing and Management*, *36*(6), 779–808. https://doi.org/10.1016/S0306-4573(00)00015-7

Thomas Boop. (2020). *GitHub - actions/runner: The Runner for GitHub Actions*. https://github.com/actions/runner

Travis CI - Test and Deploy with Confidence. (n.d.). Retrieved November 16, 2020, from https://travis-ci.com/

Tutorial de Git para principiantes - IONOS. (2020, July 22). https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/tutorial-de-git/

What is Jenkins? Continuous Integration (CI) Tool. (n.d.). Retrieved November 16, 2020, from https://www.guru99.com/jenkin-continuous-integration.html

YANA GUSTI. (2019, July). *JIRA moderno caso de prueba y herramienta de gestión de proyectos*.

Zhao, Y., Serebrenik, A., Zhou, Y., Filkov, V., & Vasilescu, B. (2017). The impact of continuous integration on other software development practices: A large-scale empirical study. *ASE 2017 - Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, 60–71. https://doi.org/10.1109/ASE.2017.8115619