UNIVERSIDAD TECNOLÓGICA NACIONAL

FACULTAD REGIONAL CÓRDOBA

INGENIERÍA EN SISTEMAS DE INFORMACIÓN

4K1

INGENIERÍA DE SOFTWARE

TRABAJO CONCEPTUAL N°1

GESTIÓN DEL SOFTWARE COMO PRODUCTO

DOCENTES :

- ING. JUDITH MELES
- ING. ROBLES JOAQUÍN LEONEL

INTEGRANTES:

GRUPO N° 6

- AMATO, ANTONELLA (72062)
- MOISES , LUIS (59647)
- RODRIGUEZ , RAUL (57852)
- VACA, MACARENA (67401)
- VINCENTI, ENRICO (59747)

FECHA DE ENTREGA: 30/08/2019

ÍNDICE

RESUMEN

La integración continua es un procedimiento de desarrollo enmarcado en la metodología ágil en la actualidad. Aunque este término no es nuevo, hace mucho tiempo comenzó a desarrollarse y con el pasar de los años se fue modificando y actualizando.

Aunque el nombre lo dice todo, la integración contínua se trata de que los trabajadores que están desarrollando un producto de software integren su trabajo de manera automática y sistematizada, para garantizar que se mantenga funcional en todo momento.

El hecho de que se componga el trabajo como un todo es lo que se conoce como integración, es continua porque se trata de que el trabajador adicione su trabajo en cada integración.

Como dice uno de los manifiestos ágiles, se debe entregar software funcionando, por sobre documentación detallada, y el principio básico de la integración contínua es entregar software funcionando y testeado, lo más rápido posible. La integración contínua permite mantener una aplicación siempre lista y funcional en cualquier etapa de desarrollo.

INTRODUCCIÓN

La Integración, Entrega y Despliegue continuos, son prácticas de desarrollo relativamente nuevas que han ganado bastante popularidad en los últimos años. La Integración Continua se trata acerca de validar el software, tan pronto como sea verificado a nivel de código fuente, intentando garantizar que funcione y continúe funcionando cuando nuevos cambios sean implementados en el mismo. La Entrega Continua es una propuesta superadora respecto a la Integración Continua ya que hace del software un elemento listo para despliegue a tan solo un click. Por último, el Despliegue Continuo automatiza el proceso completo de despliegue de software en nuestros servidores o en los de nuestros clientes.

Las tres prácticas tratan de automatizar el proceso de prueba y despliegue, minimizar (o eliminar completamente) la necesidad de intervención humana, minimizando el riesgo de errores, y la construcción y el despliegue de software más fácil hasta el punto en que cada desarrollador en el equipo puede hacerlo.

El problema con la Integración Continua, Entrega Contínua y Despliegue Continuo es que no es fácil de configurar y requiere de mucho tiempo, especialmente cuando nunca lo ha hecho antes o se desea integrar un proyecto existente. Sin embargo, cuando se lleva a cabo de manera correcta, se amortiza mediante la reducción de errores, por lo que es más fácil arreglar los errores que se detectan, y la producción de software mejora su calidad lo que conduce a clientes más satisfechos.

La Integración Continua es el primer paso para entregar software consistente y de alta calidad . Se trata de asegurar que el software está en un estado de despliegue en todo momento. Es decir, el código compila y puede suponerse que es de una calidad razonablemente buena.

Esta práctica permite al equipo de trabajo, ir manteniendo de manera integrada, todas las porciones de desarrollo que realizan individualmente cada uno de los miembros, de manera que asegura que el código compila y ejecuta de manera correcta.

En el ámbito de la cátedra de Ingeniería de Software, se desarrollará a continuación una investigación respecto a integración contínua, con el objetivo de profundizarnos en procedimientos que ayuden a mejorar la calidad del producto de software así como también, la forma de trabajar de los equipos. Se podrá comprender qué es la integración contínua, cuales son sus beneficios, requisitos y complicaciones, así como también herramientas que ayudan a la realización de la actividad.

DESARROLLO

Anteriormente, era común que los desarrolladores de un equipo trabajen aislados durante un largo periodo de tiempo y que solo intentasen combinar los cambios una vez completado el trabajo. Como consecuencia, la integración conjunta de dichos cambios en el código resultaba difícil y ardua, además de dar lugar a la acumulación de errores durante mucho tiempo que no se corregían. Estos factores hacían que resultase más difícil proporcionar actualizaciones a los clientes con rapidez. Con el paso del tiempo, el avance del desarrollo simultáneo en equipo, el surgimiento de las metodologías ágiles y con la necesidad de mejorar esta forma de "unificación", surgió una práctica que ayuda a los diferentes equipos de trabajo a combinar los cambios en el código fuente, llamada Continuous Integration, en español, integración contínua.

Entonces, ¿qué es la integración contínua?. Según Martin Fowler, un autor reconocido en el ámbito de prácticas ágiles, define la integración continua como una "**Práctica de desarrollo de software** donde los miembros del equipo **integran su trabajo frecuentemente**, al menos una vez al día. **Cada integración se verifica con un build [1] automático** (que incluye la **ejecución de pruebas**) para **detectar errores de integración** tan pronto como sea posible." ¿Qué quiere decir todo esto? - Comenzaremos por desglosar estas ideas a continuación:

1. *"Práctica de desarrollo de software"*

La *"Integración Contínua"* afecta a nuestro proceso de desarrollo de software, incorporando nuevas prácticas o interconectando las que ya había de manera de agilizar la tarea de construcción del mismo, pudiendo tener ciclos de entrega de funcionalidades ya implementadas en menor tiempo.

2. *"los miembros del equipo integran su trabajo frecuentemente"*

En un equipo de trabajo, el código que implementa cada desarrollador no suele actuar aislado. Generalmente se distribuye el trabajo entre los miembros del equipo, y cada uno comienza a trabajar, pero hasta que no se une todo el código, no es posible detectar los errores cometidos. En cambio, la integración continua se incorpora con el objetivo de evitar esto. La idea es que en vez de dejar dicha integración para el final, se vayan haciendo pequeñas integraciones de código frecuentemente, entonces la cantidad de errores disminuye, y así es más fácil corregirlos.

3. *"cada integración se verifica con un build automático"*

Para poder integrar frecuentemente, debemos primero que nada, realizar varias comprobaciones. Una de estas es garantizar que el código que se ha integrado compila

correctamente. Siendo la mejor forma de integrar, realizarla al final de la implementación de la funcionalidad. Debemos tener en cuenta que cada funcionalidad puede compilar de manera correcta individualmente pero no así al momento de integrar dos o más funcionalidades. Puesto que a la hora de hacer esto en mayor medida se producen errores de integración que debemos captar y resolver lo más pronto posible.

4. *"que incluye la ejecución de pruebas"*

La integración continua no es solo integrar el código frecuentemente y que éste se pueda compilar correctamente sino que también implica definir una estrategia de pruebas, diferentes tipos, en distinta profundidad, para comprobar que las integraciones sean correctas. La ejecución de las pruebas conlleva tiempo, y tenemos que tener un Feedback[2] relativamente rápido para conocer si la integración es correcta o no y poder continuar desarrollando.

Se definen distintas etapas, con distintos grados de prueba. Podemos tomar como ejemplo el Smoke Test[3], que son pruebas simples y básicas que nos sirven para comprobar que la integración es correcta y tener un feedback rápido.

5. *"para detectar errores de integración tan pronto como sea posible"*

Cuanto más tiempo pasa desde que se introduce un error hasta que se detecta y se resuelve, es más costoso solucionar ese error. El objetivo de la integración continua es automatizar las cosas más simples, para detectar los fallos más básicos lo antes posible y luego tener tiempo para otras comprobaciones que detecten fallos más complejos.

La integración continua consiste en "resolver problemas rápidamente". Dado que el software es integrado frecuentemente, cuando se encuentra un error por lo general no es necesario retroceder mucho para descubrir donde se introdujo la falla. En comparación, cuando un equipo no sigue una estrategia de integración continua los periodos entre integraciones son largos y la base de código es muy distinta por lo que cuando se encuentran errores hay que revisar mucho más código, y por ende requiere de un mayor tiempo y esfuerzo.

Fowler dice que "la integración continua no elimina los defectos, pero si hace que sea mucho más fácil encontrarlos y corregirlos".

Aunque esta práctica nos otorga buenos resultados, requiere bastante disciplina y un esfuerzo significativo de introducirlo a un proyecto, y que el equipo de trabajo se adapte a este procedimiento, sin embargo, una vez habilitado es sencillo de mantener y brinda grandes beneficios. Pero, ¿cómo funciona? ¿Se decide que se va a incorporar la metodología y listo? No, para lograrlo se deben cumplir ciertos requisitos como:

- Tener un repositorio maestro donde esté disponible todo el código fuente y que cualquier integrante del equipo pueda obtenerlo.
- Automatizar el proceso de integración para que cualquier persona pueda generar una versión ejecutable del sistema a partir del código fuente.
- Automatizar las pruebas para que sea posible ejecutar la *matriz (suite)*[4] de pruebas en cualquier momento con un solo comando.
- Asegurar que cualquiera puede obtener el ejecutable más reciente y tener la confianza de que es la mejor versión hasta el momento.

Implantar una estrategia de integración continua no es algo trivial, sí es algo que está al alcance de cualquier organización que desarrolla software sin importar su tamaño, presupuesto o tecnología utilizada. Más que nada es cuestión de que el equipo entienda las prácticas y sea disciplinado. Entonces, ¿cómo se implementa la integración continua?

1) Concientizar a las personas y dar formación sobre el tema: este paso es primordial a la hora de enfrentar cambios en la organización. Los cambios afectan a las personas, y normalmente se niegan aceptarlos, aunque se conozcan los problemas. Se debe explicar qué se va a hacer, y por qué se va a implementar la integración continua. Es importante informarle al equipo sobre las nuevas tecnologías que se van a implementar.

2) Tener claro el proceso de desarrollo de una empresa: se refiere a tener bien definido los entornos que hay, cuales son los criterios para las versiones de software, o qué se hace en cada entorno, también qué tipo de pruebas se hacen y en qué momento.

3) Tener en claro la política de gestión de configuración y control de versiones: sin el control de versiones la integración continua no funcionaria, vigila constantemente y monitoriza los cambios que se hagan en el sistema de control de sistema. Por eso antes de implementar este proceso, se debe definir cómo se combinan las estrategias de control de versiones con la integración continua.

4) Gestión de tareas y trazabilidad: debido que el objetivo de la integración continua es llevar a cabo integraciones de código pequeñas y frecuentes, para ello, es una buena idea dividir el trabajo en partes. Algo muy útil son las herramientas de gestión de tareas como Jira y Redmine.

5) Automatización de build: automatización de la compilación. Es el paso de la integración continua antes de ejecutar pruebas.

6) Definir el pipeline[5] de integración continua: Se debe pensar que pasos o fases debería pasar el código. Es una serie de pasos de cómo se va a realizar la integración para luego programarlo en el servidor.

7) Elegir e instalar el servidor de integración continua: serán útiles las herramientas Jenkis, Travis, CruiseControl, entre otras que se detallarán más adelante.

Y ¿Cómo se hace esto? Existe un flujo de actividades posible que sirve para guiar la implementación de esta práctica.

1. Cada desarrollador obtiene una copia del código en su espacio de trabajo privado.
2. Cuando un desarrollador termina su trabajo, envía sus cambios al repositorio.
3. El servidor de integración continua monitorea el repositorio y detecta los cambios automáticamente.
4. El servidor (de integración continua) integra el sistema y corre las pruebas unitarias y de integración.
5. El servidor publica los artefactos ejecutables.
6. El servidor asigna una etiqueta a la nueva versión y su ejecutable.
7. El servidor informa al equipo el resultado de la integración.
8. Si la integración o pruebas fallan, el servidor alerta al equipo y el equipo lo resuelve lo antes posible.
9. Se continúa integrando frecuentemente a lo largo del proyecto.

Para que todo esto funcione se pueden establecer políticas que los integrantes del equipo deban cumplir, tales como:

- Registrar (commit[5]) su código al menos una vez al día.
- No registrar código con errores.
- No registrar código sin probar.
- No generar nuevas versiones con código que no funciona.

- No pueden terminar sus actividades del día hasta haber registrado su código y que el sistema se integre exitosamente.

La mayoría de los equipos de trabajo fijan sus propias políticas y autocontrolan que se cumplan sin necesidad de que haya una supervisión. En metodologías ágiles uno de los manifiestos dice que "Las mejores arquitecturas, diseños y requerimientos emergen de equipos auto organizados", y que " a intervalos regulares, el equipo evalúa su desempeño y ajusta la manera de trabajar". Estos manifiestos, fomentan la autocorrección de los equipos de trabajo. Por ende, como cada equipo de trabajo es diferente, la forma de aplicar la práctica de integración contínua varía de un equipo a otro, aunque todos siempre mantienen el mismo objetivo, combinar los cambios en el código en un repositorio central de forma periódica, tras lo cual se ejecutan versiones y pruebas automáticas.

¿Qué otros beneficios otorga la integración contínua?

- Mejora la productividad de desarrollo: La integración continua mejora la productividad del equipo al liberar a los desarrolladores de las tareas manuales y fomentar comportamientos que ayudan a reducir la cantidad de errores y bugs enviados a los clientes.

- Ayuda a encontrar y arreglar los errores con mayor rapidez: Gracias a la realización de pruebas más frecuentes, el equipo puede descubrir y arreglar los errores antes de que se conviertan en problemas más graves.

- Entrega de actualizaciones de manera ágil y frecuente: La integración continua le permite a su equipo entregar actualizaciones a los clientes con mayor rapidez y frecuencia.

Por lo tanto, la integraciòn contínua permite mejoras en diferentes aspectos de la calidad de software como:

- Calidad del proceso

  La integración continua permite darle más visibilidad al proceso de desarrollo en general, a todos los pasos que se siguen desde que se empieza a programar un requisito del cliente hasta su puesta en producción. Donde, todos tienen la posibilidad de conocer las fases por las que va pasando el código, y el estado del software en cada momento. Además permite tener una mayor claridad sobre gestión de configuración, políticas de control de versiones, entre otras cosas.

- Calidad de producto:

  Debido a que el principal objetivo de la integración contínua es detectar los errores lo más pronto posible, en fases tempranas del desarrollo, para poder solucionarlos rápidamente, permite que a medida que el proyecto avanza, la cantidad de errores disminuye. Introduciendo varios tipos de pruebas y comprobaciones, minimizando los riesgos, y haciendo que el software tenga menos bugs que si no se realizara esta práctica.

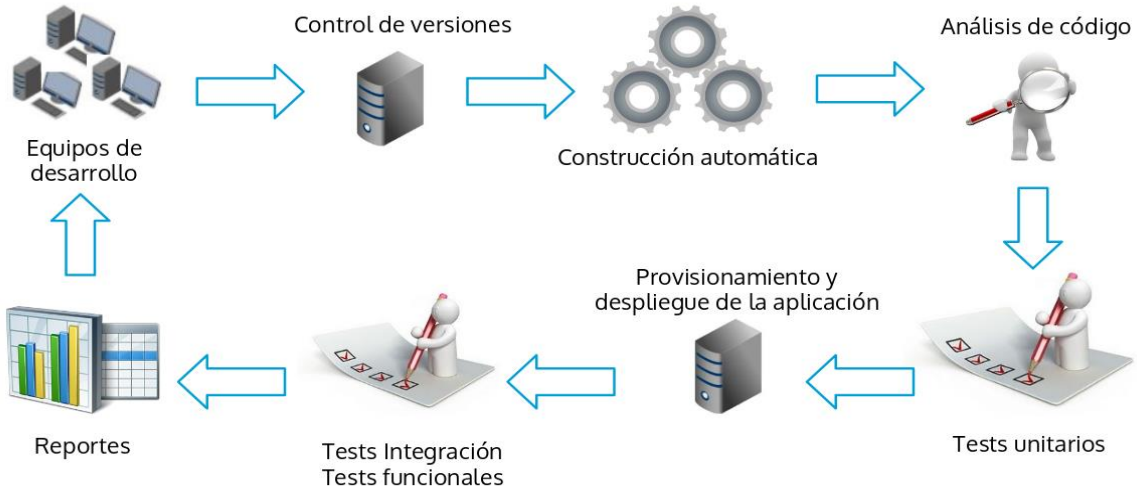
- Calidad del equipo de trabajo:

  El equipo aprende a auto organizarse, y cada miembro profundiza como hacer distintos tipos de pruebas (unitarias, de integración), mejores prácticas de programación y en general a desarrollar código de mayor calidad. Así como también, afrontar nuevos retos, mejoras y cambios y estar más motivado. Y lo más importante, aprende a automatizar procesos repetitivos, lo que le sirve para ahorrar tiempo

Aunque la integración continua aporta muchos beneficios a la hora de trabajar, también presenta algunos inconvenientes, tales como:

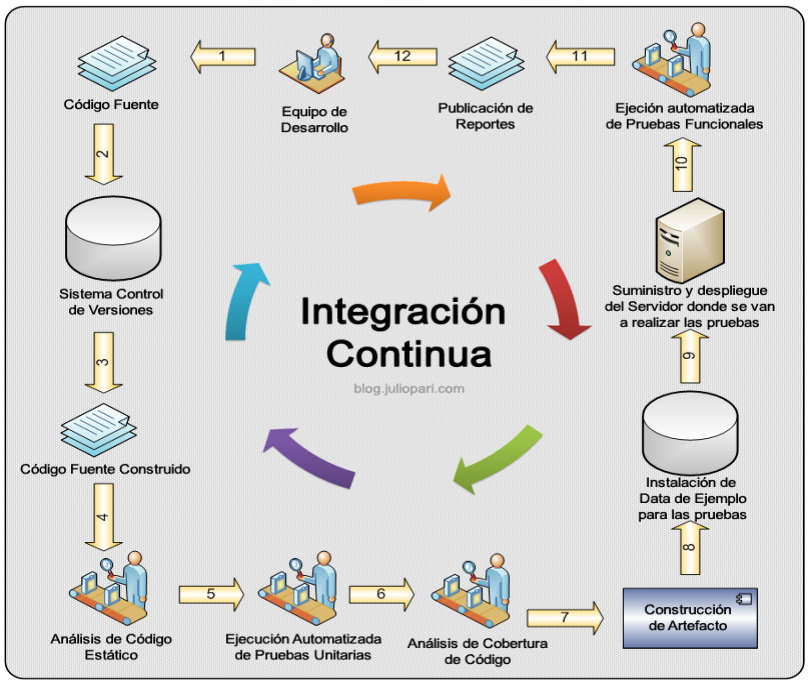
- Sobrecarga en el mantenimiento del sistema: si el servidor de integración continua es detenido durante un tiempo considerable, se acumula código en el control de versiones, el cual podría tener errores que no serían detectados durante este período de tiempo. Además al iniciarse, el servidor de integración podría sobrecargarse por existir un gran número de cambios.
- El impacto inmediato al subir código erróneo provoca que los desarrolladores no suban su código frecuentemente como sería conveniente como copia de seguridad.

Una vez visto qué es la integración continua, se puede identificar ciertos componentes que intervienen en ella como el equipo de trabajo, control de versiones, etc. A continuación, a modo de ejemplo, se podrá observar dos esquemas sencillos y posible de integración continua. Ambos son muy similares, y tienen mínimas variaciones. Ya que ambos buscan cumplir el mismo objetivo, pero la forma de implementarlo siempre va a variar según las políticas que establecer cada equipo.



En este esquema, el equipo de desarrollo sube el código que han hecho al control de versiones frecuentemente. Esto no es tan sencillo como parece, ya que primero se debe elegir qué estrategia de control de versiones se debe seguir, como por ejemplo, desarrollo en distintas ramas, una rama de integración que vigile el servidor de integración continua, etc.

Luego de que el servidor de integración continua se encargue de vigilar el repositorio de control de versiones (la forma de hacerlo depende de cual es la estrategia de control de versiones que haya definido el equipo), los desarrolladores realizan test unitarios, es decir en partes individuales de código. Después de ir integrándose lo testean como un todo, esto es, compilar todo la integración, realizarán las comprobaciones que el equipo haya establecido, esa es la principal función de la integración continua, ir haciendo integraciones frecuentes y pruebas para obtener así errores o fallas, o por el contrario un resultado exitoso. Por último, brinda un feedback a los desarrolladores para saber si el código subido esta bien o hay algún fallo que se debe solucionar, y luego se continúa el ciclo hasta que se finaliza el proyecto



En este otro esquema se plantea otra forma de implementar un flujo de la integración continua haciendo énfasis en un par de aspectos aparte de los mencionados en el anterior esquema, donde tenemos primero a nuestro equipo de desarrollo, quienes trabajan en una funcionalidad que se traduce en código terminado. Luego este es subido a un repositorio el cual lleva el control de las versiones y/o cambios en el mismo. Como se explicó en el anterior esquema. Donde ya en esta etapa se puede decir que ese código ha sido construido, notificando a los interesados de que hay una nueva versión de código y está listo para un análisis del mismo en busca de fallas. Acto seguido se le realizan una serie de pruebas unitarias al código. Luego de que se realicen estas pruebas se lleva a cabo un análisis de cobertura de código, el cual tiene como finalidad conocer el grado de la profundidad de las pruebas realizadas al código. Un programa con una alta cobertura de código ha sido probado más profundamente y tiene una baja probabilidad de contener errores en comparación con un programa con una baja cobertura de código.

De esto surge un artefacto, una pieza de código funcionando, sobre la cual se van a ejecutar las pruebas funcionales, pero para poder realizar dichas pruebas esta pieza de código debe ser desplegada en un servidor para poder realizar las mismas, así también debemos contar datos de ejemplo para que sirvan de entrada en dicha pieza de código para corroborar si lo que debe hacer, efectivamente lo hace. Cabe aclarar que dentro del marco de Integración Continua hablamos mucho de la automatización sino de otra forma la integración frecuente sería imposible.

Por lo tanto, la ejecución de las pruebas funcionales no es la excepción, sino que también son configuradas previamente para realizarse de manera automatizada. A posterior a la realización de las pruebas funcionales, se publican los informes obtenidos de estas pruebas de manera de que el equipo de desarrollo tenga un feedback sobre la funcionalidad implementada, por ejemplo si hay cambios que realizarle o está lista para pasar a producción.

Es redundante decir que la Integración continua necesita de un sistema que nos permita integrar el trabajo de los miembros de nuestro equipo lo más rápido posible, es allí donde interviene un servidor de integración continua, ahora bien ¿Cuáles son las funciones generales de este dispositivo milagroso?

Un servidor de integración continua principalmente debe permitir la realización de las operaciones de *checkout* [7] del administrador de configuración, la compilación del código, la creación del archivo de la aplicación , el despliegue del archivo en la máquina de prueba, la ejecución de una serie de pruebas, la notificación del resultado, la creación de un informe de estadísticas y la integración con otras herramientas.

Por consiguiente, el servidor de integración continua se encarga de ejecutar el código automáticamente según la programación que establece el equipo de trabajo, por ejemplo cada vez que se se realiza un commit/checkin/push al control de código fuente.

Además, es posible establecer distintas configuraciones para un mismo proyecto. Por ejemplo, crear una configuración que realice una compilación “rápida”, ejecutando sólo los

test unitarios cada vez que se realiza un commit , y programar una compilación "completa", que ejecuta también los tests de integración y genera los instaladores todas las noches (las típicas nightly *builds* [4]).

En la actualidad, internet ofrece una gran variedad de herramientas para la integración continua. Todas tienen como objetivo ayudar al desarrollador en la implementación de esta metodología, y lo hacen de diferentes modos y con la ayuda de características distintas. Pero estas herramientas no solo se diferencian unas de otras en cuanto a sus características, sino que también existe una gran variedad en lo que respecta a precios y licencias. Mientras que muchas de ellas son de *código abierto*[8] y se encuentran disponibles de forma gratuita, otros fabricantes ofrecen herramientas comerciales. Seguidamente, se observará un resumen de las más utilizadas con su respectivo análisis de sus características y funciones.

1. **Jenkins**: es probablemente una de las herramientas de integración continua más conocidas del mercado. Cuenta con numerosas funciones que asisten no solo en la integración continua, sino también en el despliegue y la entrega continua. Es gratuito y de código abierto. Permite almacenamiento en la nube y es compatible con diferentes sistemas operativos.

2. **Travis CI:** esta herramienta de integración continua trabaja en estrecha relación con el popular software de control de versiones *GitHub*[9]. Travis puede configurarse con un sencillo *archivo YAML*[10] 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. También es gratuita para código abierto.

3. **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. Es multiplataforma. Gratuita para proyectos de código libre, ONG y centros escolares De lo contrario, pago único de entre 10 y 126 500 dólares, dependiendo del número de servidores utilizados.

4. **GitLab CI:** forma parte del conocido sistema de control de versiones GitLab. Además de integración continua, GitLab[11] 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. Posee alojamiento propio o en la nube, y ofrece una versión gratuita con pocas funciones y el precio para otras versiones, entre 4 y 99 dólares/mes por usuario.

5. **Circle CI:** funciona tanto con GitHub como con Bitbucket[12]. 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. Se configura con archivo YAML, ofrece también alojamiento propio o en la nube. Tiene una versión gratuita para un solo contenedor. Sino cuesta entre 35 y 3150 dólares al mes.

6. **Cruise Control:** es una de 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. Es multiplataforma, gratuita y de código abierto.

7. **Codeship:** 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. Es gratuita para 100 compilaciones al mes en una pipeline de prueba o su precio ronda entre 75 y 1500 dólares/mes.

8. **TeamCity:** a 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.Gratuito para 100 builds con 3 agentes de compilación sino pago único de entre 299 euros y 21 999 euros

En resumen, todas las herramientas de integración continua presentan ventajas e inconvenientes. La siguiente tabla sirve para comparar de manera más sencilla cada una de ellas, y poder identificar cuales son más útiles o se adaptan mejor a la forma de trabajo de cada equipo y organización.

	Entrega Continua	Alojamiento en la nube	Licencia	Precio Versión Comercial	Versión Gratuita	Particulares
Jenkins	Si	Si	MIT	-	Si	Numerosos Plugins
Travis CI	No	Si	MIT	69-489 U$D / Mes	Si	Conexión directa con Github
Bamboo	Si	Si	De Propietario	10 - 126.500 U$D (pago único)	Si	-
GitLab CI	Si	Si	MIT / EE	4 - 99 U$D / Mes	Si	Conexión directa con otros productos de Atlassian
Circle CI	Si	Si	De Propietario	50 - 3150 U$D / Mes	Si	Fácil de Utilizar
Cruise Control	No	No	BSD	-	Si	Completamente Gratuita
Codeship	Si	Si	De Propietario	75 - 1500 U$D / mes	Si	Versión profesional y básica
Team City	Si	No	De Propietario	299 - 21999 € (pago único)	Si	Gated Commit

CONCLUSIÓN

Este trabajo intenta dar respuesta a lo que significa en sí misma la metodología de integración continúa. Haciendo énfasis en sus principales ventajas, como lo son su capacidad de detección temprana de errores, por consiguiente estos se pueden resolver a tiempo, sin seguir perpetuándose. Así como también entregar actualizaciones a los clientes con mayor rapidez y frecuencia. Sin embargo esto no garantiza un software libre de errores, pero si ayuda a mitigar el gran impacto que podrían generar en una etapa más avanzada de desarrollo. Porque como ya se mencionó anteriormente, el objetivo de la integración continua es sin lugar a dudas, detectar errores lo antes posible para ser corregidos y entregar software funcionando, nuevamente tan rápido como sea posible.

En contraparte, uno de los principales problemas o desafíos que surgen de implementar esta metodología, es el desconocimiento muchas veces por parte de los miembros del equipo de su existencia, y cuando se le es presentada, inmediatamente puede generar un rechazo comprensible ya que caen en la creencia de que pierden tiempo teniendo que hacer tantas integraciones, alegando que es mejor realizar una integración una vez cada cierto tiempo. Pero cuando realizan estas integraciones monstruosas y surgen errores por todos lados que los obligan a realizar un análisis para detectar de donde provino el error y porque. Es alli donde la perdida de tiempo que resultaba integrar frecuentemente se vuelve insignificativa comparada con la pérdida de tiempo que resulta buscar un error de integración cuya integración pasada fue hace meses. No estamos diciendo que no cueste incorporar esta práctica a la forma de trabajo del equipo, pero una vez lograda, será realmente beneficioso para todos, puesto que el software es una actividad humano-intensiva y lo más costoso en todo desarrollo es el tiempo, por lo que debemos ser eficientes y responsables en su uso.

Honestamente no tenemos en si una excusa para no aplicar esta metodología puesto que existe una gran variedad de herramientas que dan soporte a la integración continua y muchas de ellas son de uso libre y gratuito, por lo que la falta de capital no es una restricción que nos detenga a implementar esta práctica. Todas tienen como objetivo ayudar al desarrollador en la implementación de esta metodología, y lo hacen de diferentes modos y tienen características distintas, como por ejemplo Jenkins, Travis, Bamboo, Cruise Control, entre otras.

Ninguno de los integrantes de nuestro equipo de trabajo ha implementado alguna vez esta práctica, pero luego de esta investigación, hemos comprendido claramente los beneficios que esta posee, por lo que todos estamos dispuestos a utilizarla de ahora en adelante, y hacerle llegar a otros equipos que también desconozcan esta información para así ayudar a que mas gente pueda incorporarlo en su labor diario.

ANEXO

Glosario

1. Build: es el total de etapas necesarias para la compilación o creación de entregables al momento de ejecutar los test (funcional, unitarios, etc.).

2. Feedback es un mecanismo por el cual una cierta proporción de la salida de un sistema se redirige a la entrada, con señales de controlar su comportamiento

3. Smoke test son aquellas pruebas que pretenden evaluar la calidad de un producto de software previo a una recepción formal, ya sea al equipo de pruebas o al usuario final, es decir, es una revisión rápida del producto de software para comprobar que funciona y no tiene defectos que interrumpan la operación básica del mismo

4. Matriz (suits): es un conjunto de pruebas automatizadas ya predefinidas en un software para que se ejecute y corrobore tu código con la intención de ver si pasa o no la "aceptación". También llamadas pruebas de unitarias. La idea es corroborar la integración, que todo funcione todo junto y no por partes (llámese módulos)

5. Pipeline: Es un conjunto de etapas, de fases por las que va pasando el software y que son automatizadas. Logrando definir una nueva forma de trabajar en el mundo devops con integración continua. Utilizando pipeline podemos definir el ciclo de vida completo de una aplicación (descargar código, compilar, test, desplegar, etc.) mediante código. De esta forma, resulta mucho más sencillo replicar los diferentes pasos con distintas aplicaciones y gestionar mejor los cambios en cada paso.

6. Commit: es la operación que permite la validación de las actualizaciones del código fuente existente en el directorio de trabajo local de la máquina del desarrollador por medio de la herramienta de gestión de configuración (como SVN). El commit se hace desde el directorio de trabajo local hacia el referencial de la herramienta de gestión de configuración.

7. Checkout: es la operación de extracción de una versión de un proyecto que se está desarrollando del referencial del administrador de configuración en un directorio de trabajo local.

8. Código Abierto (Open Source): Hace referencia a todos aquellos programas informáticos que disponen a cualquier usuario el acceso a su código de programación facilitando por parte de otros programadores ajenos la modificación del mismo.No debemos confundir en ningún momento con Software Libre que es software que puede descargarse y distribuirse de manera gratuita.

9. GitHub: es un servicio de almacenamiento de repositorios git, pero este añade muchas más opciones por su cuenta. Mientras Git es una herramienta por comandos, Github provee

una interfaz gráfica web. También provee control de acceso y muchas más opciones de colaboración, como lo son wikis y herramientas para la gestión de tareas para nuestros proyectos.

10. Archivos YAML: son usados por desarrolladores, pues contienen código fuente escrito en el lenguaje de programación YAML (YAML no es un lenguaje de marcas). El código contenido en un archivo YML es legible por el ser humano y generalmente se utiliza para serializar datos. Con la ayuda de los archivos YML, se pueden leer y escribir datos con independencia completa de los lenguajes de programación. Por tanto, los archivos YML pueden utilizarse junto con otros muchos archivos contenedores de código fuente en distintos lenguajes, como C, C#, C++, Java, PHP y muchos más.

11. 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.

12. Bitbucket es un servicio web de almacenamiento de repositorios con control de versión, propiedad de Atlassian, para código abierto y proyectos de desarrollo que usan sistemas de control de versionado como lo son Mercurial (Desde su lanzamiento) o Git (desde Octubre de 2011). Bitbucket ofrece ambos planes, comercial y de cuentas gratis.

BIBLIOGRAFÍA

- ¿Cómo estruran un informe técnico?
    - https://studylib.es/doc/5748976/c%C3%B3mo-estructurar-un-informe-t%C3%A9cnico-como-un
    - https://www.academia.edu/5091078/INFORME_T%C3%89CNICO_LA_ESTRUCTURA_DEL_INFORME_T%C3%89CNICO_EST%C3%81_FORMADA_POR_La_parte_inicial
    - http://www.ingenieria.unam.mx/~especializacion/egreso/Como_redactar_un_informr_tecnico.pdf

- Información
- https://www.martinfowler.com/articles/continuousIntegration.html
- https://sg.com.mx/revista/54/integraci-n-continua
- https://www.javiergarzas.com/2014/08/implantar-integracion-continua.html
- https://www.javiergarzas.com/2014/08/implantar-integracion-continua-2.html
- https://aws.amazon.com/es/devops/continuous-integration/
- http://blog.koalite.com/2013/05/servidor-de-integracion-continua-una-buena-inversion/
- https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/herramientas-de-integracion-continua/
- http://tisten.ir/wp-content/uploads/2019/02/Sander-Rossel-Continuous-Integration-Delivery-and-Deployment-Packt-2017.pdf
- https://www.uv.es/capgeminiuv/documents/Test%20+%20IC.pdf
- https://www.youtube.com/watch?v=LJ6ZLjhDj1s
- https://www.youtube.com/watch?v=mrUfnPqUwKw&t=223s
- https://www.youtube.com/watch?v=58SjUKY6sKk&t=335s