GESTIÓN DEL SOFTWARE COMO PRODUCTO
CONTINUOUS INTEGRATION

Universidad Tecnológica Nacional
Facultad Regional Córdoba

Bruno Conti 72890
bruno.conti1997@gmail.com
Pablo Lucero Schneider 67041
pabloluceroschneider@gmail.com
Franco Migotti 60546
franco.migotti@gmail.com
Agustin Peralta 71759
aperaltam98@gmail.com

RESUMEN: *En el presente paper desarrollaremos el concepto de Continuous Integration, es una fase de un proceso de desarrollo de software automatizado. Se identifica la importancia de implementar Continuous Integration en el desarrollo de software agile, los elementos esenciales, sus beneficios y las actividades necesarias para poder llevar a cabo esta práctica. Finalmente realizamos una breve presentación de las herramientas que permiten implementar esta práctica.*

PALABRAS CLAVE: build, mainline, repositorio, test automatizado.

1 INTRODUCCIÓN

A continuación describiremos cómo se lleva a cabo la práctica de Continuous Integration, para ello nos basaremos en la bibliografía sugerida por la cátedra de Ingeniería de Software de la carrera de Ingeniería en Sistemas de Información. Principalmente, usaremos el artículo de Martin Fowler, quien fue uno de los precursores de esta práctica.

La redacción del paper se hará de acuerdo al estándar IEEE.

Se intentará presentar elementos claves para tener en cuenta a la hora de trabajar en un proyecto automatizado, y finalmente daremos una apreciación personal acerca del uso de esta práctica en el desarrollo de software con metodologías ágiles.

2 Continuous Integration vs Continuous Deployment vs Continuous Delivery

Continuous Integration, Delivery y Deployment son tres fases de un proceso de lanzamiento de software automatizado. La integración es el punto inicial de este proceso, consiste en múltiples desarrolladores uniendo sus cambios en el código fuente en un repositorio del proyecto.

Continuous Delivery es el siguiente paso, implica mantener el código listo para desplegarse en cualquier instante de tiempo.

Continuous Deployment es la etapa final, cuya responsabilidad es iniciar y distribuir automáticamente el artefacto de software a los usuarios finales. En el momento de la implementación, el artefacto ha pasado con éxito las fases de integración y entrega.[5]

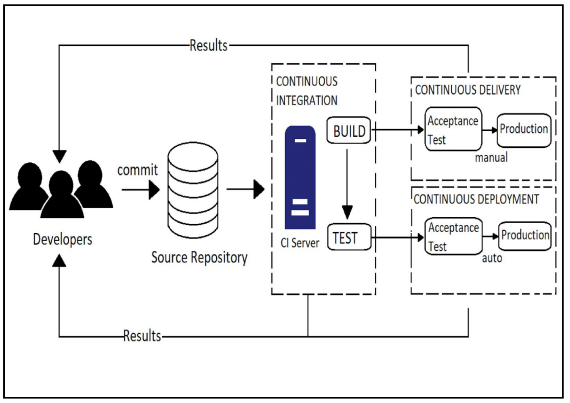

Figura 1. Fases de un proceso de lanzamiento de software automatizado.

3 ¿Qué es Continuous Integration?

Continuous Integration es una práctica de desarrollo de software donde los miembros de equipos integran su trabajo frecuentemente, para esto cada miembro integra al menos una vez al día. Cada una de estas integraciones son verificadas mediante pruebas automatizadas para detectar errores de integración lo más rápido posible. De esta manera, se reducen problemas de integración, permitiéndole al equipo desarrollar software de manera cohesiva y rápida. [1]

Continuous Integration es una práctica valiosa y bien establecida en organizaciones modernas de ingeniería de software de alto rendimiento, que permite a los desarrolladores de software trabajar de forma independiente en funciones.

4 Importancia del Continuous Integration

Para entender la importancia de la Continuous Integration es necesario conocer los problemas que surgen cuando no es aplicada. Uno de estos problemas es cuando la coordinación y la comunicación en el desarrollo del software se hacen de manera manual. Los diferentes equipos que contribuyen al producto se deben coordinar secuencialmente para lanzar nuevas características y mejoras, como también definir qué miembro del equipo es el responsable de realizar estos lanzamientos.

Esto implica un incremento en los costos y en los riesgos del proyecto, debido a que se pueden producir lanzamiento de código más lentos y con mayores tasas de fallas. Estos riesgos crecen exponencialmente cuando el tamaño del equipo o del producto aumentan.

Sin la implementación de Integración Continua se puede generar una desconexión entre el equipo de ingeniería y el resto de la organización. Al equipo de ingeniería le resultará más difícil estimar el tiempo de entrega porque el tiempo para integrar nuevos cambios se convierte en un riesgo desconocido.[2]

5 Factores decisivos para implementar con éxito Continuous Integration

Para poder trabajar con esta práctica es necesario tener en cuenta cuatro factores.

- Acuerdo entre los miembros: Debe existir un consenso en donde los desarrolladores se comprometan a trabajar realizando commits y que cada vez que los realizan el build no debe romperse. Si el build se rompe, deben arreglarlo.
- Build Automático y Auto-Testeable: Los builds que se llevan a cabo después de cada integración deben ejecutarse automáticamente y deben incluir pruebas, a fin de detectar y resolver errores rápidamente.
- Builds erróneos esporádicos: Los builds con errores van a seguir existiendo, pero deberían ser poco frecuentes.
- Solución rápida de builds erróneos: Debido a que los commits se realizan frecuentemente, cuando hay errores se pueden resolver rápidamente.[3]

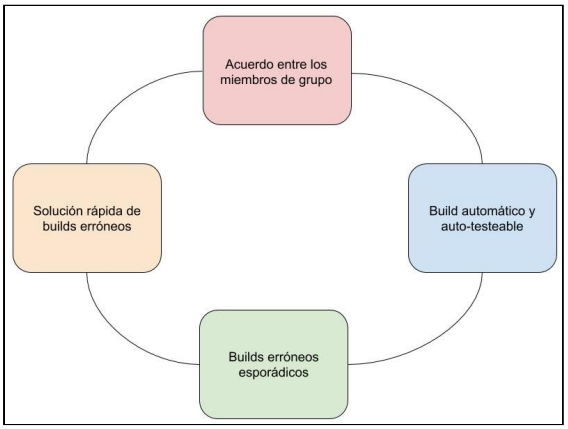

Figura 2. Factores críticos del éxito de Continuous Integration.

6 Construcción de una funcionalidad con Continuous Integration

Tomando una funcionalidad pequeña que puede ser desarrollada en pocas horas, explicaremos el proceso de construir una característica con CI.

En primer lugar, el desarrollador toma una copia del código fuente integrado actual en su máquina local, usando el administrador de código fuente.

Un sistema de control de código fuente mantiene todo el código fuente de un proyecto en un repositorio. El estado actual del sistema es llamado ‘mainline’. La copia en la máquina del desarrollador se denomina ‘working copy’.

El desarrollador realiza su trabajo en esta 'working copy'. Esto implica modificar el código como así también agregar pruebas automatizadas. La integración continua supone un alto grado de pruebas que están automatizadas en el software: una facilidad llamada 'self-testing code'. Frecuentemente es usada una version del popular XUnit testing framework.

Una vez finalizado, se realiza una build automatizada en la máquina del desarrollador, esto implica, que el código fuente de la working copy es compilado y llevado a un ejecutable, además de ser sometido a una serie de pruebas automatizadas. Sólo si dichas pruebas son superadas, el desarrollador está en condiciones de confirmar los cambios en el repositorio. Sin embargo, durante este periodo de tiempo, otros desarrolladores pueden haber realizado cambios en el mainline. Por lo tanto, primero se debe actualizar la working copy. Si llegara a ocurrir algún tipo de error en la reconstrucción, es responsabilidad del desarrollador solucionarlo para realizar una construcción de la working copy sincronizada con el mainline.

Sólo cuando la working copy esté sincronizada correctamente se pueden confirmar los cambios en el repositorio. Esta integración puede ser hecha manual o automáticamente.

La idea es tener múltiples construcciones correctas por día. Si bien las construcciones incorrectas pueden ocurrir, éstas deben solucionarse rápidamente.

El resultado de hacer esto es un software estable que funciona correctamente y contiene pocos errores.

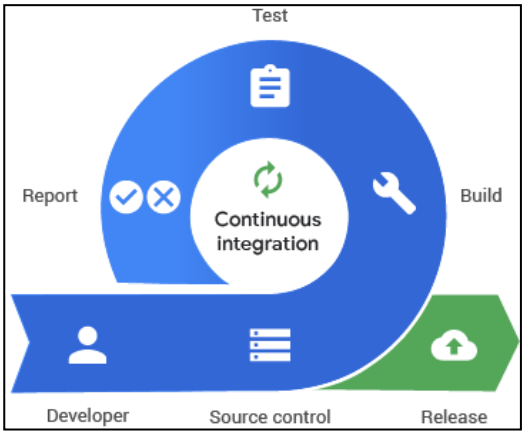

Figura 3. Proceso de construcción de una funcionalidad por Continuous Integration.

7 Beneficios de Continuous Integration

Los beneficios de CI no están sólo limitados al equipo de desarrollo, sino que proveen un beneficio general a la organización. Esta práctica permite mejorar la transparencia y visión dentro del proceso de entrega y despliegue de software.

- Reducir riesgos: CI permite estimar el tiempo que tomará el proceso, como también, conocer en cualquier instante de tiempo que tan avanzado está el proceso, que funciona, que no funciona y los errores pendientes del sistema.
- Reducción de errores: Si bien CI no elimina todos los errores permite encontrarlos y solucionarlos más rápido.
- Mejorar el ciclo de retroalimentación: Los equipos pueden probar ideas e iterar diseños de productos de forma rápida. También los cambios en los productos pueden ser rápidamente impulsados y medidos.
- Mejorar la comunicación: CI mejora la comunicación entre los equipos de desarrollo y, entre estos equipos y el cliente.
- Ciclo de desarrollo colaborativo: Los desarrolladores pueden ver y colaborar en características que están siendo desarrolladas por otros miembros del equipo a medida que se avanza en el flujo de trabajo.
- Agregar funcionalidades en menor tiempo: CI permite añadir piezas funcionales de software en cada integración. [4]

8 Buenas prácticas de Continuous Integration

8.1 Mantener un repositorio de fuente única

Es importante asegurar el uso de un sistema de gestión de código de fuente o también conocido como repositorio. Dentro de este repositorio deberían estar todos los elementos necesarios para realizar una compilación, incluidos scripts de prueba, instalación, esquemas de base de datos, bibliotecas de terceros y configuración de IDE. En general se debería guardar en el repositorio todo lo necesario para realizar la compilación, pero no el resultado de la misma.

8.2 Automatizar la construcción del Build

El objetivo es simplificar y agilizar la construcción de los builds.

Es fundamental contar con entornos automatizados para llevar a cabo las compilaciones y que estas sean ejecutadas con un simple script. Para poder realizar esto hay que asegurar que todos los elementos necesarios para la construcción de un build estén dentro del entorno automatizado. De lo contrario, efectuar una build de manera manual puede ser una tarea tediosa e insumir demasiado tiempo, es especial en proyectos a gran escala.

Un factor relevante a tener en cuenta es que la construcción de un build debe realizarse en un ambiente separado y con las mismas características del entorno en donde va a ser desplegada la aplicación. Para esto se recomienda la utilización de máquinas virtuales.

8.3 Hacer Builds Auto-Testeable

Tradicionalmente, efectuar una build significaba compilar, vincular y agregar todas aquellas librerías necesarias para obtener un programa ejecutable. Sin embargo, un programa funcionando no significa que lo realice correctamente. Por lo tanto, una forma para detectar errores de manera rápida y eficiente es incluir pruebas automatizadas en el proceso de complicación.

El resultado de ejecutar un conjunto de pruebas debe indicar si alguna prueba falló. Un concepto a tener en cuenta es que para que la compilación sea auto-testeable, el fallo de una prueba debe hacer que la compilación falle.

Esta práctica no asegura la ausencia de errores en el código, pero si los reduce sustancialmente.

8.4 Commits diarios en el mainline

Es esencial que cada desarrollador realice al menos una vez al día un commit en el repositorio. Para esto es importante que el ‘working copy’ coincida con el mainline a la hora de hacer el commit, en caso de presentarse algún error el desarrollador debe resolverlo. Cabe destacar que cuanto más frecuente se realizan commits, hay menos lugar a errores y más rápido se pueden resolver.

Un factor clave en la integración, es la comunicación frecuente, esto permite a los desarrolladores entender los cambios a medida que se producen.

8.5 Cada Commit debe construir el mainline en una máquina de integración.

Al hacer commits diariamente se obtienen builds de manera frecuente, los cuales deberían ser testeados y aprobados para poder integrarse al mainline. Para poder lograr esto, es necesario que el desarrollador previo a realizar el commit, actualice su ‘working copy’ y realice las pruebas. En caso de que el resultado de ejecutar los test sea exitoso se integra en el mainline, de lo contrario, el desarrollador debe resolver cualquier conflicto para asegurar la estabilidad del mainline.

Se debe garantizar que las builds se realicen en una máquina integrada, para esto existen dos maneras. Por un lado, el desarrollador puede hacer un build manual, esto implica que el desarrollador ejecute el commit en la maquina de integración de forma tal que sea exitoso para poder considerar la integración finalizada.

Por otro lado, se puede realizar esta práctica utilizando un servidor de integración continua que monitorea el repositorio. Cada vez que se ejecuta un commit, el servidor actualiza el mainline en la máquina de integración, inicia el build y finalmente notifica el resultado al desarrollador que lo realizó.

8.6 Reparar rápidamente errores del mainline

Se debe tener en cuenta que a la hora de trabajar con Continuous Integration pueden surgir errores en el mainline, este inconveniente debería tratar de evitarse tomando los recaudos necesarios al momento de realizar un commit, pero en el caso de que se presente hay que repararlo de inmediato.

La forma más rápida de arreglar el build es revertirlo hasta el último commit realizado con éxito, llevando al sistema a la última build exitosa que se realizó.

8.7 Mantener el buid rápido

La idea central de Continuous Integration es proporcionar una retroalimentación rápida. Para lograr esto, el tiempo ideal de compilación es de diez minutos. El motivo de tener tiempos bajos para la construcción del build es ahorrar tiempo a los desarrolladores. Esto en la práctica es una dificultad, ya que el tiempo de build implica ejecutar las pruebas. La clave es encontrar un equilibrio entre tiempo y capacidad de detección de errores.

En primer lugar se podría pensar en realizar las pruebas de compilación y ejecución que no involucren acceso a la base de datos. De esta forma y debido a que estas pruebas corren rápido es factible mantener el build dentro del tiempo ideal.

En una segunda etapa se ejecuta un conjunto de pruebas que sí involucran el acceso a los datos reales, estas pruebas pueden tardar un par de horas. En caso de que este conjunto secundario de pruebas falle, no es necesario detener todo sino que se debe hacer lo posible por solucionar los errores rápidamente.

A partir de este punto la idea es poder detectar los fallos del segundo conjunto de pruebas, identificarlos y poder construir pruebas aplicables a la primera etapa para evitar que los errores no lleguen a la segunda etapa.

8.8 Probar en un ambiente clonado del entorno de producción

El objetivo de realizar una prueba, es probar el sistema bajo ciertas condiciones, con el objetivo de identificar cualquier problema que pueda surgir con el sistema en funcionamiento. Si se realiza la prueba en un ambiente distinto al entorno de producción se corre el riesgo de que se obtengan ciertos resultados que no se darían en producción. Por lo tanto es necesario que el entorno de prueba sea lo más parecido posible al entorno de producción, es decir que utilice el mismo motor de base de datos, la misma versión del sistema operativo, librerías apropiadas y que se ejecute sobre el mismo hardware.

Para esto una de las opciones más recomendadas es utilizar la virtualización, el uso de máquinas virtuales facilita la sencilla instalación de pruebas de ejecución y compilación, permitiendo ajustar los entornos de diferentes maneras hasta obtener la configuración que más se asemeje al entorno de producción.

8.9 Debe ser fácil para cualquier persona conseguir el último ejecutable

Durante el desarrollo de un producto de software, la mayor dificultad radica en entender lo que un cliente necesita, que muchas veces difiere de lo que el cliente manifiesta que desea. Llevar esta especificación al producto puede ser aún más difícil. Suele ser más fácil para las personas ver algo que no es del todo correcto y decir a partir de esto qué es lo que necesita más precisamente. Los procesos ágiles usan esta característica de los clientes para mejorar el producto bajo desarrollo.

Es por esto que cualquier desarrollador debe ser capaz de obtener la última versión ejecutable y ser capaz de ejecutarla a fines de: demostraciones, pruebas exploratorias, o simplemente para ver lo que ha cambiado esta semana.

Para poder lograr esto, se debe asegurar que los involucrados sean capaces de encontrar la última versión ejecutable. Es muy útil ubicar los ejecutables en un mismo lugar Por último, se debería realizar una prueba de confirmación al último ejecutable para asegurar una version estable.

8.10 Todo el mundo debe poder ver lo que está sucediendo

Como ya hemos destacado anteriormente en el informe, una parte fundamental de Continuous Integration está en la comunicación. Partiendo de este punto la idea central debe ser que todos puedan visualizar en cualquier momento del tiempo el estado del sistema y los cambios que se han realizado.

Resulta de gran importancia establecer dentro del entorno de trabajo y desarrollo indicadores para poder determinar el estado del mainline, la última compilación exitosa y si hay compilaciones en proceso.

El hecho de que todos puedan ver el estado del mainline es importante si trabajamos tanto con manual como con un servidor de Continuous Integration.

8.11 Automatizar el despliegue

Para poner en práctica Continuous Integration es necesario utilizar múltiples entornos, esto lleva a que muchas veces por día los ejecutables cambien de entornos. Por lo que el mainline debería ejecutarse automáticamente en cualquiera de estos entornos. La implementación automatizada ayuda a acelerar el proceso y reducir los errores.

Un factor a destacar es que tener una implementación automatizada, también obliga a desarrollar mecanismos que nos permitan de forma automatizada volver mainline a su último estado estable para reparar rápidamente cualquier error que pueda surgir en la implementación.

9 Herramientas para implementar Continuous Integration

Las siguientes herramientas son las más utilizadas en el mercado para desarrollar aplicando Continuous Integration

- Bitbucket Pipelines: Es una herramienta directamente integrada en Bitbucket, un servicio de alojamiento en la web, para proyectos que utilizan control de versiones. Ofrece también Continuous Delivery. Es fácil y sencillo de configurar, se integra con Jira o Trello y permite mantener protegido el mainline.
- Jenkins: Es utilizada para la automatización, es de código abierto, permite construir build, testear e integrar cambios de forma sencilla. También permite integrar procesos de todo tipo, como por ejemplo, builds, documentación, test, componentes y análisis.
- AWS CodePipeline: Es uno de los más dominantes del mercado, puede integrarse directamente con otros servicios web de amazon, como así también con servicios de terceros (Bitbucket o GitHub).
- Azure Pipeline: Azure es la plataforma de infraestructura en la nube de Microsoft, Azure Pipeline permite la integración con otros servicios en la nube, también con GitHub y permite integrarse en aplicaciones de desarrollo de Microsoft.
- GitLab: Es un servicio web de control de versiones y de desarrollo basado en Git, por lo que ofrece una experiencia de usuario con soporte para contenedores. También ofrece una completa experiencia DevOps que permite implementar además, Continuous Delivery y Continuous Deployment.

10 Conclusión

Continuous Integration resulta una práctica de suma importancia para el desarrollo y mantenimiento de productos de software, debido a que forma parte, junto con Continuous Delivery y Continuous Deployment, de un proceso de desarrollo automatizado.

Continuous Integration propone un cambio de mentalidad y se centra en lograr una entrega rápida de software funcionando a lo largo de todo el ciclo de vida del proyecto. Para esto Continuous Integration adopta la automatización buscando agilizar los procesos manuales y cumplir con la coherencia en las sucesivas entregas.

Podemos ver una fuerte relación entre el manifiesto Ágil y la forma de trabajar en Continuous Integration. Es decir, se centra en los individuos y en la responsabilidad que tengan a la hora de testear su código antes de subir al repositorio; el software funcionando siempre va a ser la métrica más importante; la colaboración y comunicación entre los desarrolladores es fundamental para lograr coordinación en el grupo de trabajo, y por último, ésta práctica nos permite flexibilidad para responder ante cualquier tipo de cambio surgido en el transcurso del desarrollo del proyecto. Por lo expresado en las líneas anteriores, se puede afirmar que el desarrollo automatizado es una clara manifestación de la aplicación de la metodología agile en el desarrollo del software.

Resulta importante comprender que Continuous Integration promueve constantemente la comunicación y la colaboración entre los miembros del equipo. Este factor es el que permite una eficiente aplicación del desarrollo automatizado y garantiza un buen resultado. Además hay que tener en cuenta que la correcta comunicación no solo beneficia al equipo sino a la organización en su conjunto.

La idea que proporciona Continuous Integration acerca de trabajar con entornos automatizados, auto-testeables y en donde los desarrolladores actualicen su trabajo al menos una vez al día está completamente fundamentada en el hecho de entregar valor al cliente en cada una de las iteraciones. Lo anterior, combinado con una correcta y efectiva comunicación con el cliente nos lleva a obtener una retroalimentación activa que nos lleva a una evolución rápida del producto de software.

Podemos afirmar que hoy en día la mayoría de las empresas de desarrollo de software aplican el desarrollo de software automatizado en su negocio, esto es porque las ganancias de velocidad aplicando Continuous Integration, Delivery y Deployment son sustancialmente importantes y necesarias para que las empresas sigan siendo competitivas. Así lo pudo confirmar Amazon cuando en mayo de 2011 publicaron por primera vez la capacidad de implementar software en entornos de producción cada 11 segundos.

Finalmente podemos afirmar que el desarrollo automatizado es el futuro del desarrollo de software, eliminando tareas rutinarias y engorrosas para los desarrolladores y permitiéndoles así centrarse en soluciones creativas que generen un valor de negocio tanto en el cliente como en la organización en sí misma.

11 Referencias

[1] Martin Fowler (2006, Mayo 01) Continuous Integration, *[En línea]* Disponible en: https://www.martinfowler.com/articles/continuousIntegration.html
[2] Max Rehkopf, What is Continuous Integration, [*En línea] Disponible en: https://www.atlassian.com/continuous-delivery/continuous-integration*
[3] Disponible en: http://www.institucional.frc.utn.edu.ar/sistemas/lidicalso/pub/file/Exploraciones/Continuous%20Integration.pdf
[4] Continuous Integration. Disponible en: https://www.thoughtworks.com/continuous-integration
[5] Dev Leaders Compare Continuous Delivery vs. Continuous Deployment vs. Continuous Integration. Disponible en: https://stackify.com/continuous-delivery-vs-continuous-deployment-vs-continuous-integration/