FUNDAMENTOS DE LA ENTREGA CONTINUA

Facundo Mallia – 67781
facumallia56@gmail.com
Gastón Cabrera – 67051
gasty065@gmail.com
Lautaro Diharce Paz–70693
lautarodiharce@gmail.com
Tamara Nicole Soria – 69684
tamarasoria35@gmail.com
Priscila Anahi Palacio – 63018
prii.palacio@gmail.com
Gastón Luis Vottero – 68753
gastonlvottero@gmail.com

RESUMEN: *En el presente paper describimos qué es la entrega continua, cuán importante es implementarla en una empresa de software y cómo realizar los cambios culturales y tecnológicos que se necesitan para poner en práctica ésta técnica.*

PALABRAS CLAVE: Entrega continua, ciclo de vida, software, product owner , automatización, trunk o master.

1 INTRODUCCIÓN

En la década de los 80s y 90s, las metodologías de desarrollo dominantes utilizaban un proceso denominado “cascada”. Esto significaba que se seguía una estricta estructura secuencial en la que los equipos se embarcan en la siguiente fase, si y sólo si, la fase anterior se había concluido por completo. Esto hacía que los equipos trabajasen aislados y generaba conflictos entre las distintas áreas, siendo la lucha más común entre desarrolladores y testers.

Por sobre todas las desventajas que provocaba el proceso en “cascada”, se comenzó a notar que los tiempos se tornaban largos para poder entregar un release y para cuando esto se hacía, el programa ya no cumplía con las expectativas del cliente o el modelo de negocio había cambiado.

Integración continua, Entrega continua y Despliegue continuo son técnicas relativamente nuevas que fueron ganando popularidad en los últimos años. Para poder avanzar en los temas, primero se debe entender que integración continua se trata de validar todo software que haya sido desarrollado y subido al repositorio. Entrega continua es el paso siguiente y es, en pocas palabras, hacer que el programa esté a un solo “click” de despliegue. Por último, el despliegue continuo es la técnica que automatiza por completo el proceso de despliegue de la aplicación al cliente (o nuestros propios servidores).

Las tres prácticas tratan de automatizar los procesos de testing y despliegue, intentando eliminar la necesidad de intervención humana, minimizando el riesgo de errores y facilitando los releases al punto que cualquiera del equipo pueda hacerlo.

Para una empresa que se dedica a la producción de software, la entrega continua se ha convertido en una práctica crucial para mantenerse competitivos.

2 ENTREGA CONTINUA

“Continuous Delivery is a software development discipline where you build software in such a way that the software can be released to production at any time.” - Martin Fowler

La entrega continua puede entenderse como la práctica dentro del desarrollo de software mediante la cual se busca asegurar que el código se encuentre en un estado el cual permita desplegar el mismo en cualquier momento.

Puestos a elegir un concepto clave a resaltar dentro de este proceso, sería ‘automatización’. La automatización permite al cuerpo de desarrollo volver procesos exitosos en procesos exitosos y repetibles, eliminando el error humano, característica inherente de la repetición de un proceso manual.

El beneficio directo de la automatización de procesos es la reducción en el tiempo que conllevan algunas de las fases de trabajo de forma tal que puedan dirigirse los esfuerzos de los trabajadores hacia la completitud y calidad de entrega de otras tareas.

Según Fowler uno realiza entrega continua cuando:

- El software es desplegable a lo largo de su ciclo de vida.
- El equipo prioriza mantener la capacidad de despliegue por sobre el desarrollo de nuevas características.
- Alguien puede obtener de forma veloz, feedback acerca del estado de completitud de sus sistemas cada vez que alguien más realiza cambios en los mismos.
- Se puede realizar despliegue de de cualquier versión del software a demanda, con solo presionar un botón.

Para lograr la entrega continua se necesita:

- Una relación cercana y colaborativa entre todos los involucrados en la entrega (a menudo llamado cultura DevOps).
- Automatización extensiva de todas las partes que sea posible dentro del proceso de entrega, a menudo haciendo uso de Tuberías de Despliegue (*Deployment Pipeline*).

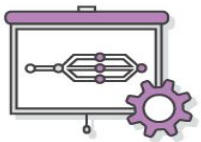
3 ¿POR QUÉ ENTREGA CONTINUA?

La entrega continua plantea realizar release frecuentes, planteando una solución a los procesos tradicionales, como el denominado proceso “Cascada”, donde se realizan release con tiempos largos, casi a final del desarrollo, lo cual tiene como consecuencia que no es el producto que espera el cliente, por motivos de que tiene errores detectados al final, o funcionalidades con poca calidad.

La entrega contínua el software se prueba frecuentemente logrando la detección de errores, calidad de performance en el tiempo justo para poder realizar las correcciones o refactorizaciones necesarios para lograr software de calidad y disponibles para producción.

Algunos de los beneficios de la entrega contínua:

Automatizar el proceso de publicación de software: Permite al equipo crear, probar y preparar automáticamente los cambios del código para poder ponerlo en producción, teniendo como resultado un proceso exitoso.



Mejora la productividad de desarrollo: Mejora la productividad del equipo al liberar los desarrolladores de las tareas manuales y reducir la cantidad de errores.

Automatizar el testing: Permite detectar errores al principio del proceso, antes de que se los errores se conviertan en problemas graves, esto se logra gracias a las pruebas frecuentes y exhaustivas. La entrega continua permite realizar pruebas adicionales en el código con factibilidad, ya que todo el proceso es automatizado.

Entregas frecuentes y rapidez: Permite al equipo entregar actualizaciones a los clientes con mayor rapidez y con frecuencia. Cuando las entregas se hacen adecuadamente se va a poder disponer de un producto listo para la implementación.

4 BASES DE LA ENTREGA CONTINUA

La entrega continua se asienta sobre tres pilares base: gestión de configuración, integración continua y testeo continuo.

4.1 GESTIÓN DE CONFIGURACIÓN

La automatización juega un papel vital cuando se busca la capacidad de entregar software de manera confiable. Una de las claves para lograr esto es tomar tareas manuales repetitivas y automatizarla, para lo cual será imperativo realizar un control de versiones de todos los ítems relacionados a estos procesos.

Tendremos dos metas primordiales:

- Reproducibilidad: El conjunto de acciones llevadas a cabo para conseguir la automaticidad deberían poder ser copiadas de manera idéntica en otros entornos.
- Trazabilidad: Deberíamos ser capaces de elegir cualquier entorno y determinar el conjunto de dependencias y sus versiones. Habilitando de esta forma una comparativa entre versiones para identificar los cambios realizados en ella.

Estas prácticas ofrecen múltiples beneficios, tales como:

- Recuperación en el caso que alguno de los entornos falle, como podría ser una brecha en la seguridad.
- Una mejora en la calidad general del producto debido a una respuesta mas rapida al feedback recibido. Esta velocidad se consigue al no tener que realizar las tareas repetitivas tratadas anteriormente.
- Control de defectos debido a la estructura altamente trazable, que permite identificar los componente en los cuales un cambio tiene impacto, de la mano con un control de versiones que permite volver a una versión anterior, sin errores del proyecto.

A medida que los sistemas se vuelven más complejos lograr los objetivos primordiales se vuelve progresivamente más difícil, y alcanzar ambas metas hasta el último byte es virtualmente imposible. Por tanto la clave de la gestión de configuración es tratar de simplificar la arquitectura, entornos y procesos en pos de reducir el esfuerzo requerido para alcanzar el objetivo.

4.2 INTEGRACIÓN CONTINUA

La integración continua (CI por sus siglas en inglés) es la práctica en la cual los desarrolladores integran el código con el cual trabajan, en un repositorio compartido al menos una vez al dia. Cada integración es verificada de forma automática antes y despues de ser integrada, permitiendo a los equipos detectar errores de forma veloz.

Lo que se busca lograr es mantener al software en un estado de trabajo permanente y que los trabajos de los desarrolladores en sus respectivas ramas no difieran significativamente del master. Estudios sugieren que esta práctica a sistemas más estables y mejora la calidad general del software.

4.3 TESTEO CONTINUO

La automatización de ciertos aspectos programables con respecto a las pruebas ayuda a la entrega de software de forma masiva. En la mayoría de los proyectos existen algunos aspectos de las pruebas que podrían ser automatizadas, y algunos que no, permitiendo centrarse más en aquellas que no podrían ser automáticas. Esto requiere habilidades tanto de los evaluadores como de los desarrolladores y su trabajo continuo en equipo, pero también esa calidad está integrada en la arquitectura del software en sí. Pueden abarcar pruebas de unidad, integración, funcional, de rendimiento y de seguridad. Además, las pruebas de automatización, ayudan en gran medida a satisfacer demandas de clientes por cambios rápidos, y posteriormente, llegar a una etapa en la que cada proyecto de construcción, incluso los que tienen cambios muy pequeños, se entregan probados y verificados.

A medida que la complejidad del software crece, la cantidad de esfuerzo de verificar los cambios y las características incorporadas, crecen al menos de forma lineal. Esto significa que el tiempo de prueba es directamente proporcional al número de casos de prueba necesarios para verificar la corrección. Aplicar la automatización de escenarios de prueba donde se puede le permite disminuir el tiempo / dinero que se necesita para verificar si la interacción del usuario con la aplicación funciona como se ha diseñado. Aumentar el número de pruebas automatizadas, así como invertir en conseguir el tiempo de la prueba-deteriorado, su agilidad y capacidad de respuesta, aumenta de forma masiva, mientras que también reduce el costo.

Lo más importante, para que la entrega continua sea un éxito, es que, los equipos de desarrollo, prueba y operaciones de TI deben trabajar juntos como un solo equipo de entrega desde el principio.

5 ENTREGA CONTINUA Y DEVOPS

DevOps proviene de Development + Operations, es decir, lograr una integración entre el desarrollo de software y las operaciones.

DevOps es la práctica de asegurar que los entornos físicos y de desarrollo de una organización se establezcan para ofrecer nuevas construcciones en la producción lo más rápido posible.

La entrega continua está muy relacionada a DevOps ya que, con DevOps los desarrolladores necesitan aprender a construir software de alta calidad, listo para producción y los DevOps necesitan aprender de las metodologías ágiles.

La implementación correcta de DevOps logra una entrega continua exitosa.

6 LA ENTREGA CONTINUA SE BASA EN PRÁCTICAS ÁGILES

La entrega continua surge naturalmente del movimiento ágil. Dicho movimiento trata de solventar el problema de los releases grandes, llenos de errores y tardíos a través de cambios iterativos e incrementales en el código y la colaboración entre los equipos.

Dentro de los beneficios de las metodologías ágiles se incluye la habilidad de adaptarse rápidamente a los cambios y minimizar los riesgos.

7 LA PRÁCTICA DE LA ENTREGA CONTINUA.

Muchas veces se cree que la práctica de la entrega continua se basa en un modelo ideal, que solo puede ser llevado a cabo por grandes empresas ricas en recursos. Lo cierto es que con implementar la practica, asi no se cumple del todo en ellas, se obtiene grandes beneficios.

7.1 LA ENTREGA CONTINUA ES UN CAMBIO CULTURAL.

En la mayoría de los equipos, el cambio más grande es comenzar a trabajar con un solo grupo, encargado de la implementación exitosa de software de calidad.
Aun así, en lo grupos en donde cada uno tiene una rol y una responsabilidad específica, todos en conjunto se encargan del desarrollo del software y el proceso de lanzamiento.

El objetivo es tratar de cada evento como una oportunidad para aprender y mejorar el proceso.

También se requiere que las personas de pruebas y operaciones de IT se involucren más temprano en el diseño del software, así estarán mejor preparadas para probar e implementar asegurándose que el código sea comprobable y desplegable.

De esta manera, la mayoría de los errores serán resueltos al momento de la implementación.

7.2 ¿CÓMO COMENZAMOS?

Poder comenzar con la entrega continua es un proceso que conlleva cambios en la forma de trabajar, la forma de desempeñarse los equipos y las herramientas que se utilizan.

Lo correcto, es implementar la entrega continua en pequeños pasos:

- ¿Qué pasos toman más tiempo?
- ¿Qué pasos son más propensos a errores o requieren más intervención humana?

Estos son los pasos para primero automatizar, lo que produce una mejora inmediata y devuelve tiempo para implementar en el proceso.

Podemos facilitar la automatización mediante:

- Establecer control de versiones.
- Agregar pruebas para verificar que el código funcione.
- Administrar servidores con herramientas de administración de configuración.
- Monitorear ítems de configuración..
- Mejorar la escritura del código.

Al principio de la implementación de entrega continua, todo puede ser desordenado y presentarse errores que con el tiempo se va mejorando y a partir de ahí, se vuelve a ajustar el proceso.

8 HERRAMIENTAS DE LA ENTREGA CONTINUA.

La entrega continua es un conjunto de prácticas más que de herramientas, sin embargo estas son necesarias para facilitar el proceso y la comunicación entre los integrantes.

Todas las herramientas tienen una cosa en común: se trata todo como código. Lo que hace posible automatizar cada parte y que todos los miembros hablen el mismo idioma y manejen las mismas herramientas.

8.1 INFRAESTRUCTURA DEFINIDA POR SOFTWARE.

La infraestructura debería poder responder automáticamente a cualquier necesidad que requiera una aplicación para ejecutarse correctamente.
Es por eso que implementamos un enfoque definido por software.

Se utilizan herramientas que pueden implementar todo lo que las aplicaciones necesiten, basándose siempre en la infraestructura como código.

8.2 MONITOREO.

Se debe monitorear continuamente el entorno de prueba para poder implementar un proceso de mejora continua. El monitoreo le da al equipo datos que muestran si el rendimiento está mejorando o se está deteriorando.
Algunas herramientas son: Graphite, logstash, Nagios, Splunk

8.3 INTEGRACIÓN CONTINUA.

Es una parte crítica de la entrega continua, ya que sin ella se corre el riesgo de que una porción de código funcione correctamente por su parte pero presente errores cuando se une en el código general.
Si bien, la integración continua es un práctica, hay muchas herramientas que ayudan a implementarla: Jenkins, Hudson, Atlassian Bamboo, CruiseControl

8.4 CONTROL DE VERSIONES.

El control de versiones es importante para todo el equipo y debe utilizarse para mantener un registro de las pruebas, scripts, documentación y archivos de configuración, todo lo que se debe hacer con un trabajo de desarrollo. Se convierte en una herramientas muy importante, ya que una vez que el equipo unifica la herramienta a utilizar, esta será la única que represente el flujo de trabajo de software completo y todos pueden acceder a ver el estado de la compilación.
Algunas herramientas son: Git, Subversion, Perforce, Mercurial

8.5 REVISIÓN DE CÓDIGO.

Estas herramientas nos permiten ver que cambios son aceptables y cuáles no, antes de fusionar los cambios con la rama principal del proyecto.
Como ejemplo de herramientas tenemos: Git, Alijo, Gerrit

8.6 GESTIÓN DE LA CONFIGURACIÓN.

Los entornos del desarrollo, la prueba y la producción deben estar configurados de la misma manera. En este punto recomendamos que los equipos de trabajo asuman la responsabilidad de configurar las máquinas de desarrollo, de pruebas y de producción con las mismas herramientas y que no se genere una configuración única y manual que genera inconsistencia con la información que manejamos ya que genera una complicación mayor a la hora de detectar una falla en el sistema y por lo tanto es más difícil solucionarlo. Una herramienta de administración de configuración permite mantener entornos consistentes durante todo el proceso de desarrollo de software, desde la computadora portátil del desarrollador hasta la producción.

Lo más importante de las herramientas de gestión de configuración basadas en infraestructuras es cualquier cambio que se realice en un entorno puede ser inmediatamente reflejado en cualquier copia del mismo.

8.7 ORQUESTACIÓN

Podemos definir orquestación como la implementación de cambios y actualizaciones en aplicaciones en un orden específico. Aplicarlo reduce en grandes proporciones la posibilidad de generar errores humanos y permite escalar más allá de lo que las personas pueden hacer manualmente.

8.8 TABLEROS

La entrega continua es algo que todo el equipo debe observar para poder saber el estado del desarrollo de software y tener en cuenta aspectos como, ver que la compilación sea correcta o no.
En estos casos los tableros son de mucha ayuda, ya que, los mismos se encargan de mostrar el estado del entorno de prueba y el entorno de producción.
Bamboo, Jenkins y Go son todos tableros de elección para muchos administradores de sistemas.

9 CONCLUSIÓN

A partir de la consideración del software como producto final de un proyecto surge la necesidad de implementar metodologías y técnicas de desarrollo que nos aseguren que los recursos disponibles serán utilizados de la forma más eficiente y eficaz posible con el fin de obtener resultados satisfactorios. La selección apropiada de una determinada metodología es un proceso complejo, puesto que los proyectos de software son de múltiples naturalezas, tamaños y requerimientos.

Definir el proceso y el ciclo de vida es una etapa fundamental en la planificación de un proyecto. El mismo define: qué trabajo técnico debería realizarse en cada fase, quién debería estar involucrado en cada fase, cómo controlar y aprobar cada fase, cómo deben generarse los entregables y cómo revisar, verificar y validar el producto final obtenido.

Es importante tener en cuenta que la selección de una metodología es un proceso que se debe llevar a cabo por personal capacitado, puesto que no todas ofrecen los mismos resultados bajo idénticas condiciones y en cualquier proyecto se cuenta con recursos limitados cuyo aprovechamiento resulta conveniente maximizar. Por otra parte no es posible generalizar una misma solución a un conjunto de proyectos que a simple vista parecieran ser similares, ya que llevar a cabo esto implica que suponemos que los proyectos que queremos solucionar son resueltos de la misma manera, cuando en realidad cada proyecto es único y por lo tanto se deberán implementar resoluciones diferentes para cada uno de ellos.

Para culminar tenemos que recordar que la entrega continua no es un conjunto de herramientas que vamos a utilizar, sino que es un proceso que prioriza la entrega de software con la menor cantidad de errores posibles, con la finalidad de entregárselo lo más rápido posible al cliente para que lo comience a usar. Claramente en ésta metodología se favorece el trabajo en equipo y que las personas se sientan parte del mismo.

REFERENCIAS

[1] "What is continuous delivery?" [En línea] Disponible en: https://aws.amazon.com/es/devops/continuous-delivery/
[2] Jez Humble on continuous delivery [en linea] disponible en: https://continuousdelivery.com Repositrio de descarga en https://github.com/jezhumble/cdsite
[3] Continuous delivery - Martin Fowler https://martinfowler.com/bliki/ContinuousDelivery.html
[4] Continuous integration, delivery, and deployment: reliable and faster software releases with automating builds, tests, and deployment - Sander Rossel
[5] Continous delivery-Pipelines en el desarrollo de software [en linea] https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/continuous-delivery/
[6] Continuous integration vs. continuous delivery vs. continuous deployment By Sten Pittet [en linea] https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment
[7] Javier Garzas on: "¿DevOps? ¿Continuous Delivery? ¿Continuous Deployment? ¿Integración Continua? Aclarando términos." [en linea] https://www.javiergarzas.com/2016/01/devops-continuous-delivery-continuous-deployment-integracion-continua-aclarando-terminos.html
[8] Continuous Delivery: What It Is and How to Get Started
[9] Continuous integration on: https://www.thoughtworks.com/continuous-integration
[10] Why is test automation the backbone of Continuous Delivery? on: https://www.thoughtworks.com/insights/blog/why-test-automation-backbone-continuous-delivery
[11] El formato del presente paper está basado en el formato de la IEEE: http://www.unisecmexico.com/archivosPDF/Formato_IEEE.pdf