**Continuous Delivery   
Universidad Tecnológica Nacional  
Facultad Regional Córdoba**

Facundo Gallardo 67100  
[facundo\_gallardo@live.com](mailto:facundo_gallardo@live.com)

Santiago Benavidez 72474

[snbenavidez98@gmail.com](mailto:snbenavidez98@gmail.com)

Gustavo Peralta 67055

[gusta\_2905@hotmail.com](mailto:gusta_2905@hotmail.com)

Lucas Ferreyra 64980

[ferreyralucas18@gmail.com](mailto:ferreyralucas18@gmail.com)

Facundo Murga 67759

fmurga2011@gmail.com

***RESUMEN:***

*En el siguiente informe, nos proponemos desarrollar el concepto de Continuous Delivery, un enfoque de la ingeniería de software en el cual se intenta producir software valioso para el usuario en ciclos cortos, y asegurar que el software puede ser liberado en cualquier momento.*

*También se identificarán algunas ventajas y desafíos que presenta Continuous Delivery.*

**PALABRAS CLAVE:**

**1 Introducción**

A lo largo de los últimos años, los enfoques respecto a el desarrollo de software fueron cambiando, buscando poder asegurar la calidad, o poder agilizar el proceso de desarrollo, siempre persiguiendo aumentar la satisfacción del cliente.  
Gracias a esta evolución es que, derivado de los principios ágiles, surgen algunos conceptos, habitualmente usados en conjunto, que son: Continuous Integration, Continuous Delivery, y Continuous Deployment.

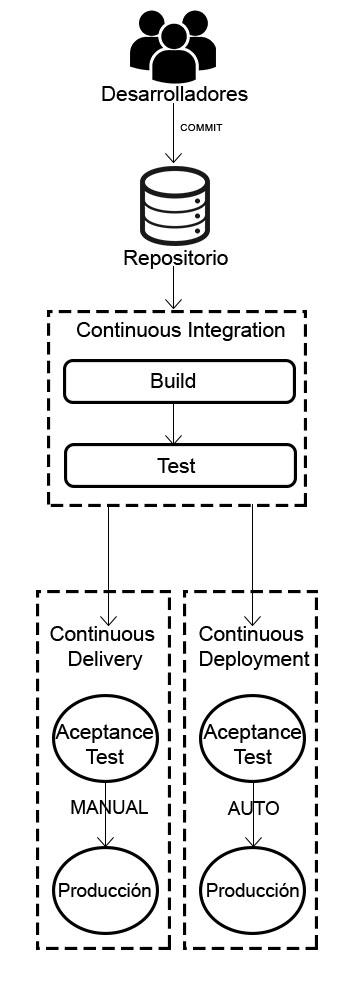
En este informe vamos a tomar solamente el concepto de Continuous Delivery (CDE) para explicar su concepto, beneficios, y los desafíos que se presentan al aplicarlo.

Si bien, el concepto de CDE ya existía hace tiempo en los principios ágiles, gracias a las herramientas y tecnologías actuales, podemos perseguir el objetivo de hacer liberaciones de software hasta varias veces al día si fuera necesario.

**2 ¿Qué es Continuous Delivery?**

Para responder esta pregunta debemos primero explicar el concepto de Continuous Software Engineering (Ingeniería de Software Continua), que es una práctica emergente que busca desarrollar, implementar y obtener un feedback de parte del cliente en ciclos cortos y muy frecuentes. Para lograr esto, se vale de 3 conceptos: Continuous Integration (CI), Continuous Delivery (CDE), y Continuous Deployment (CD).

En la siguiente imagen podemos ver la relación entre estos conceptos.

**

*Figura 1*

Habiendo definido esto, CDE es una práctica que busca asegurar que una aplicación esté siempre lista para ser pasada a producción, después de pasar exitosamente pruebas automatizadas y verificaciones de calidad. Para lograr este objetivo, CDE utiliza CI (Como podemos ver en la fig 1), y automatización de implementación.

Según L. Chen [1], esta práctica tiene muchos beneficios como la reducción en los riesgos de implementación, disminución de costos y la obtención más rápida de feedback.

Para lograr los objetivos de CDE, se utilizan muchas herramientas y tecnologías, para automatizar varios aspectos del proceso que vamos a ver más adelante, pero en CDE la tecnología no lo es todo.

Según Sylvia MacDonald [2], también es importante analizar el factor humano en CDE, hay muchas tareas que no pueden automatizarse y tienen que ser llevadas a cabo por personas de manera manual, es importante analizar estas tareas, obtener datos específicos al respecto para poder tomar decisiones que nos permitan reducir tiempos no productivos en nuestro proceso de liberación, por ejemplo, podemos modificar los medios de comunicación para reducir el tiempo de respuesta dentro del equipo.

**3 Principios de Continuous Delivery**

* **Proceso Repetible:** el proceso de implementación está formado por una serie de pasos a seguir, se debe poder asegurar que cada vez que se ejecute, se realice de igual manera.
* **Automatizar todo:** para lograr que el proceso sea repetible, debe automatizarse la mayor parte posible.
* **Control de versionado:** el control de versiones nos permite “deshacer” cambios volviendo a versiones anteriores. Junto con el código debe controlarse con versionado las configuraciones, documentaciones, bases de datos, etc.
* **Considerar la calidad:** en CDE, la calidad es algo que se controla en todos los pasos, no se deja para ser examinado por un equipo de QA.
* **Hacer la parte difícil primero:** hay que tratar los temas “dolorosos” lo más pronto posibles, para evitar tener que lidiar con ellos por el resto de las iteraciones.
* **Todos son responsables:**la organización entera, debería tener como objetivo asegurar que se entrega un producto de la mayor calidad posible.
* **“Listo” significa liberado:** una tarea puede darse por terminada, cuando el código generado está funcionando en el entorno del cliente.
* **Mejora continua:** el núcleo de CDE es el feedback, después de cada implementación se usan herramientas para medir el impacto del software liberado, permitiendo así mejorar ya sea versiones futuras del software, o la mejora del proceso de liberación en si.

**4 ¿Qué es un Pipeline?**

Anteriormente mencionamos que parte fundamental de CDE es la automatización de la mayor cantidad de tareas posibles. Para llevar a cabo esta automatización se utiliza algo conocido como Pipeline, que es una serie de pasos programados a los que se somete el código. Se incluyen compilaciones, tests automatizados, e implementaciones.

Imagen que contiene captura de pantalla

Descripción generada automáticamente

*Figura 2*

En la figura 2 podemos observar un pipeline genérico con las tareas que se ejecutan.  
Un pipeline de CDE entrega software de calidad frecuentemente y de manera predecible. Es importante destacar que en esta definición queremos dejar claro que, de ninguna manera se cambia la calidad por velocidad, de nada sirve un pipeline que permita liberar código con fallas constantemente, dentro del pipeline deben controlarse la calidad y la seguridad.

**4.1 Fases del pipeline**

El proceso que el producto atraviesa, se divide en 4 etapas, cada etapa tiene sus propias validaciones, si alguna de estas falla se detiene y se informa el error producido.

**4.1.1 Fase de componentes**

En una primera instancia, en el pipeline se compilan las unidades más pequeñas posibles del producto, por ejemplo una DLL. Y se las somete, entre otras cosas, a revisiones de código, test unitarios y análisis estático de código.

**4.1.2 Fase de subsistemas**

Una vez que se terminó la primer fase, se procede a agrupar componentes ligeramente asociados, generando subsistemas. A estos subsistemas se los somete a pruebas funcionales, para comprobar que se cumpla la funcionalidad esperada, pruebas de performance y puede haber controles de seguridad.

**4.1.3 Fase de sistemas**

Una vez que se atravesó la segunda fase, se puede definir que los subsistemas se agrupen para generar el sistema completo, que se va a someter a tests de integración, y se repetirán las pruebas de performance y seguridad.

**4.1.4 Fase de producción**

Una vez que se atravesaron todas las pruebas previas, se implementa el sistema en producción, una vez que está implementado se realizan pruebas propias de este entorno, para conocer el rendimiento que podemos esperar del sistema cuando el cliente final lo use. Entre estas pruebas se encuentran, tests de performance, de stress, etc. Es posible automatizar estos pasos a través de Continous Deployment.  
Cabe aclarar que en algunos sistemas, puede ser necesario que haya una o más validaciones manuales entre la fase de sistemas y la fase de producción para permitir que el sistema sea implementado.

**5 Beneficios de CDE**

Basado en la aplicación de CDE en Paddy Power [1] se observaron los siguientes beneficios:

* **Reducción en el tiempo de entrega:** La frecuencia de liberación se incrementó dramáticamente, hasta alcanzar una liberación semanal en promedio.  
  El tiempo entre la concepción de una historia de usuario hasta su puesta en producción se redujo hasta entre 2 y 5 días.
* **Construcción del producto correcto:** La alta frecuencia con que se producen las liberaciones, permite un feedback más rápido y constante, lo que nos permite concentrarnos en desarrollar las funcionalidades útiles para el usuario.
* **Productividad y eficiencia mejorada:** el pipeline permite eliminar el tiempo dedicado a la preparación de entornos de parte de los desarrolladores y testers, o eliminar el tiempo solucionando problemas relacionados a la implementación manual de los sistemas, todas estas tareas se realizan de manera automatizada en el pipeline.
* **Liberaciones confiables:** Dado que el proceso de liberación se realiza siempre de la misma manera, se eliminan los errores en el proceso.  
  Además, al haber liberaciones cada vez más frecuentes, en cada liberación se incluyen pequeños cambios, lo que reduce el riesgo de fallos.
* **Mejora en la calidad del producto:** Cada vez que se realiza un cambio en el repositorio de código, se realizan una serie de pruebas en el mismo, si alguna falla, el desarrollador puede saberlo inmediatamente y solucionarlo en el momento, antes de seguir adelante con la liberación. Esto permite que casi todos los bugs sean encontrados y tratados antes de llegar al usuario.
* **Satisfacción del cliente incrementada:** Dados los beneficios anteriormente nombrados, la relación con el cliente se vio beneficiada. Dado que utilizan un sistema casi libre de bugs y en caso de encontrar uno, se resuelve con gran velocidad, esto incrementa la confianza del cliente en el sistema y el equipo de desarrollo.

**6 Desventajas de CDE**

* Factor de costo: es necesario contar con un servidor potente y fiable de integración de datos para llevar a cabo las pruebas automatizadas y conseguir una liberación correcta y segura del producto.
* Las pruebas automatizadas tienen que funcionar a la perfección y no presentar errores de código.
* Requiere que haya una buena coordinación dentro del equipo porque los cambios introducidos en el código deben compilarse de forma eficiente y con una determinada frecuencia.
* Requiere una buena y continua comunicación con los clientes y sus sistemas de destino.
* El cliente tiene que estar dispuesto a utilizar el software cuando todavía se encuentra en una fase de desarrollo. Además, tiene que poner su parte y devolvernos su feedback.

**7 Desafíos al momento de aplicar CDE**

Como toda práctica, CDE presenta algunos desafíos con los que se encuentran las organizaciones que quieren aplicarlo.  
Siguiendo con el ejemplo de Paddy Power tomamos los siguientes desafíos que fueron los que ellos se encontraron:

* **Desafíos organizacionales:** El principal desafío es a nivel cultura organizacional. Tradicionalmente el proceso de liberación involucra a varias personas o equipos, cada uno con sus intereses o formas de trabajar. Para poder aplicar CDE, debemos coordinar a todos los equipos involucrados, para que todos lleguen a un acuerdo respecto a lo que se necesita. Lamentablemente este es uno de los temas sobre los cuales hay menos bibliografía disponible, lo que supone un desafío extra ya que no hay muchos lugares donde investigar al respecto.
* **Desafíos de procesos:** Algunos procesos pueden entrar en conflicto con CDE, por ejemplo, si parte del proceso determina que una nueva funcionalidad debe ser aprobada por una junta, y esta aprobación lleva varios días, no tiene sentido para funcionalidades que se podrían llevar a producción en 5 días.  
  En estos casos deberían encontrarse este tipo de procesos y encontrar una alternativa que encaje con CDE.
* **Desafíos técnicos:** A la hora de crear un entorno de CDE, todo debe hacerse a la medida, aplicando distintas herramientas y tecnologías. Este proceso puede ser costoso y requiere aprender conceptos nuevos y considerar aspectos que tradicionalmente no se consideraban.

**8 Continous Delivery versus Continous Integration versus Continous Deployment**

Continous Delivery son pequeños ciclos de construccion con sprints cortos donde el objetivo es mantener el código en un estado desplegable en cualquier momento dado . Esto no significa que el código o proyecto esté 100% completo, pero los conjuntos de características que están disponibles se verifican, prueban, depuran y están listos para implementarse, aunque es posible que no se implemente en ese momento. El enfoque y la clave aquí es mantener la base del código en un estado desplegable. Este es el proyecto cotidiano estándar que sale al público o se enfrenta al consumidor. En el mundo de hoy, no puede permitirse el lujo de lanzar un proyecto plagado de errores, por lo que los sprints más pequeños permiten tiempos de giro más rápidos para identificar errores y, por lo tanto, un tiempo más rápido para corregir esos errores, creando una base de código mucho más estable desde el principio. Este es nuestro método preferido de trabajo.

Con la implementación continua , cada cambio que se realiza se implementa automáticamente en producción. Este enfoque funciona bien en entornos empresariales donde planea utilizar al usuario como probador real y puede ser más rápido de lanzar.

La integración continua está fusionando todo el código de todos los desarrolladores a una rama central del repositorio muchas veces al día tratando de evitar conflictos en el código en el futuro. El concepto aquí es tener múltiples desarrolladores en un proyecto para mantener la rama principal del repositorio en la forma más actual del código fuente, para que cada desarrollador pueda verificar o extraer del último código para evitar conflictos. [3]

**9 Conclusiones**

Concluimos considerando que la entrega continua es una idea particularmente buena para equipos que tengan una experiencia adecuada o miembros con un nivel de seniority aceptable, no es una idea que se pueda aplicar en equipos en los cuales no se tenga experiencia o se está poniendo en practica una nueva plataforma o lenguaje, pero es una tecnica que se puede poner en practica perfectamente para equipos experimentados en los cuales se pueden realizar releases finales de bajo riesgo debido a la cantidad de interacciones que tenemos con el cliente y seran releases de mayor calidad al poder contar con herramientas de integracion continua automatizada como Jenkins, lo que acorta significativamente riesgos y costos, ademas que se ha demostrado que equipos que implementan correctamente entrega continua son equipos que funcionan mas motivados y trabajan mas felices ya que las entregas son menos agotadoras y se reduce el stress generalizado del equipo. Y esto se traslada al cliente, ya que se eliminan tareas molestas y se puede concentrar el equipo en las tareas necesarias para otorgar valor al cliente.

Si esto suena demasiado bueno para ser verdad, tenga en cuenta: la entrega continua no es mágica. Se trata de una mejora continua y diaria: la disciplina constante de lograr un mayor rendimiento siguiendo la heurística "si duele, hágalo con más frecuencia y haga que el dolor avance".

**10 Referencias**

[1] Lianping Chen, “Continuous Delivery: Huge benefits, but challenges too.”. Disponible en: https://www.infoq.com/articles/cd-benefits-challenges/

[2] Sylvia MacDonald, “Continuous Delivery – It’s not all about tech!”. Disponible en: <https://www.infoq.com/articles/continuous-delivery-not-about-tech/>

[3] Dev Leaders Compare Continuous Delivery vs. Continuous Deployment vs. Continuous Integration - Nick Jones, Stackify https://stackify.com/continuous-delivery-vs-continuous-deployment-vs-continuous-integration/