DELIVERY CONTINUO (30 de Agosto de 2019)

Acosta Nicolás, Amaya Estefania, Jofré Gastón Martín,
Doffo Tamara, Moré Milagros, Pereyra Evangelina

Resumen— La meta del delivery continuo es encontrar maneras de entregar software con valor y de alta calidad de manera eficiente, rápida y confiable.

Palabras clave—Delivery continuo (continuous delivery),* tubería de entrega *(Delivery pipeline),* ingeniería de Software continua *(continuous software engineering)

I. Nomenclatura

CI Continuous Implement.
DC Delivery Continuo.

II. Introducción

Las metodologías modernas que se aplican en la tecnología de la información dependen en gran medida en la capacidad de las personas para aceptar los cambios en entornos cada vez más ágiles. Las prácticas de desarrollo de software en cascada se fueron sustituyendo por metodologías ágiles. Este cambio atrajo a los equipos de desarrollo, usuarios, clientes y directivos cada vez más cerca, incluso modificando y creando nuevos roles, como el del Dueño del Producto.

A raíz de esto surgen nuevas prácticas conocidas como prácticas continuas: el presente escrito permite al lector interiorizarse en el delivery continuo, metodología que se ha consolidado fuertemente en el área de desarrollo de software.

Los siguientes apartados expondrán principios específicos que sientan las bases para los flujos de trabajo de delivery continuo, resumiendo el estado actual del delivery continuo.

III. Desarrollo del artículo

A. Introducción en la práctica del delivery continuo

Entre las prácticas continuas que aplican actualmente las organizaciones para poder lanzar nuevos productos y características de estos con mayor frecuencia y fiabilidad con el objetivo de acelerar el ritmo de desarrollo y entrega, pero conservando la calidad del mismo, encontramos el Delivery continuo.

Hace referencia a una serie de metodologías utilizadas en el desarrollo de software para ejecutar el desarrollo, la entrega, el feedback y la gestión de calidad de forma simultánea y cada poco tiempo, de acuerdo con un proceso repetitivo. Se centra en la capacidad de liberar de manera rápida y confiable elementos de valor hacia los clientes, utilizando la automatización siempre que sea posible [1].

El delivery continuo es una práctica de desarrollo de software donde los cambios de código se crean, prueban y preparan automáticamente para su lanzamiento en producción. Eso se expande en la integración continua mediante la implementación de todos los cambios de código en un entorno de prueba, un entorno de producción o ambos después de que se haya completado la etapa de compilación.

Puede ser totalmente automatizado con un proceso de flujo de trabajo o parcialmente automatizado con pasos manuales en puntos críticos. Cuando el delivery continuo se implementa correctamente, los desarrolladores siempre tienen un artefacto de compilación listo para la implementación que ha pasado por un proceso de prueba estandarizado.

El delivery continuo se enfoca en automatizar el proceso de entrega de software para que los equipos puedan implementar su código de manera fácil y segura en la producción en cualquier momento. Al garantizar que la base de código esté siempre en un estado desplegable, la liberación del software se convierte en un evento sin complicaciones sin un ritual complicado. Los equipos pueden estar seguros de que pueden liberar cuando lo necesiten sin una coordinación compleja o pruebas de última etapa.

Esta práctica es atractiva porque automatiza los pasos entre la verificación del código en el repositorio y la decisión de lanzar versiones funcionales bien probadas en su infraestructura de producción. Los pasos que ayudan a afirmar la calidad y la corrección del código están automatizados, pero la decisión final sobre qué publicar se deja en manos de la organización para una máxima flexibilidad.

El delivery continuo genera un feedback constante para el desarrollador procedente de pruebas automatizadas que sirven, en general, para comprobar la estructura después de cada cambio introducido en el código fuente [2].

B. Principios del delivery continuo

Existen una serie de principios específicos que sientan las bases para los flujos de trabajo del Delivery Continuo (DC).

- **Proceso confiable y repetible:**
  Los procesos organizacionales tienen su propio ciclo de vida de desarrollo.
  Comienzan generalmente como “playbooks”, que son tareas que se realizan manualmente. Luego,

pueden automatizarse con distintas herramientas de software y scripts. Al hacer esto, se garantiza que sean repetibles.

La confiabilidad se obtiene cuando dichos scripts de playbooks se ejecutan constantemente entre entornos.

- **Automatiza todo:**

La automatización es un valor clave del DC. El tiempo humano es costoso y debe usarse de manera conservadora, en lugar de ocuparlo ejecutando tareas de playbooks.

- **Control de versiones:**

El control de versiones es una necesidad absoluta para cualquier proyecto de software serio.

Esto permite a un equipo de desarrollo colaborar eficientemente en una base de código compartida. Git es el sistema de control de versiones más usado.

Habilita la funcionalidad "deshacer" al permitir retrocesos a versiones anteriores.

- **Construir con calidad:**

En DC, la calidad no es una ocurrencia tardía que se delega al equipo de control de calidad.

El ciclo de retroalimentación central del DC es un re-examen constante de la calidad entregada a los usuarios finales. Las nuevas funciones se entregan con un conjunto de pruebas automatizadas, las cuales garantizan que dicho código esté libre de errores y cumpla con las expectativas de calidad.

- **Hacer las partes más difíciles primero:**

Las tareas más lentas o propensas a errores deben abordarse lo antes posible para evitar una pérdida de energía.

Este principio ayuda a identificar las debilidades en el proceso organizacional. Si existe una tarea que se posterga o evita continuamente, es un indicador de que podría ser un área de mejora.

- **Todos son responsables:**

Toda la organización debe centrarse e incentivarse para garantizar que la entrega al usuario final sea de la mayor calidad posible.

Los gerentes del producto deben planificar con atención la implementación y asegurar la calidad. El equipo de seguridad debe participar activamente en el proceso de lanzamiento. El equipo de control de calidad debe probar los entornos de desarrollo como lo harían en la producción, para detectar cualquier falla antes de la liberación final del producto. Los desarrolladores, por su parte, deben planificar activamente el lanzamiento a producción.

- **"Hecho" significa liberado:**

Las compañías de software están en el negocio para entregar software a usuarios finales. No tendría sentido que una aplicación funcione únicamente en la máquina de un desarrollador.

El DC está completamente enfocado en entregar software al cliente final. Además, "hecho" no significa cuando se hace una contribución individual de los miembros del equipo, sino cuando todos los miembros hacen su contribución.

- **Mejora continua:**

Las prácticas de DC no terminan en la implementación, ya que el ciclo de retroalimentación es la clave de esto.

Esto quiere decir que, una vez que se haya implementado, existe una fase adicional de monitoreo y medición de la efectividad de dicha implementación. Esta fase usa herramientas automatizadas con el objetivo de medir el impacto de la implementación en el usuario final.

C. Ventajas y desventajas del delivery continuo

En un principio, el producto final solo se entregaba si todas las funcionalidades estaban totalmente desarrolladas y no se detectaban fallos importantes a la hora de realizar las pruebas de calidad.

De esta manera, el desarrollador debía entregar actualizaciones que corrijan dichos errores, cada cierto periodo de tiempo.

Gracias al DC, el cliente recibe el producto en una fase temprana del desarrollo, el cual incluye la funcionalidad estructural para que el cliente pueda probarlo en un entorno real.

Gracias al feedback recibido de parte del cliente, el desarrollador puede mejorar las funcionalidades del producto durante la fase de desarrollo. Además, recibe información valiosa que le da una idea clara sobre qué funcionalidad debería desarrollar a continuación.

Antes de que existiera el DC, este proceso solía dejar descontentos a ambas partes:

Por un lado, el cliente normalmente espera que el producto final cumpla con todos sus requisitos, y por otro, el desarrollador todavía no sabe exactamente cuáles son estos requisitos.

Si se entabla una comunicación sobre el estado de desarrollo del producto en una fase más temprana, es más fácil tener en cuenta los requisitos del cliente y no cometer errores [1].

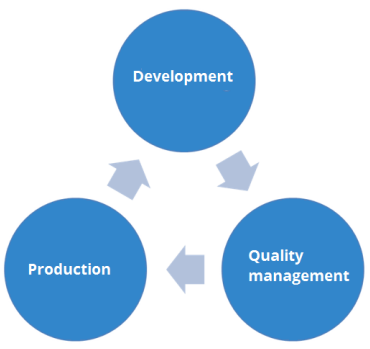

Fig. 1. Automatización del ciclo de proceso Delivery Continuo.

Como muestra la imagen, las 3 áreas que incluyen el desarrollo, control de calidad y producción no se reemplazan por un solo proceso, sino que están interconectadas.

Así, un producto pasa por cada una de estas fases repetidamente y recibe mejoras continuas.

El DC permite comprobar los procesos y mejoras implementados sobre el software en tiempo real, con el fin de conseguir un feedback.

Si un cambio genera efectos secundarios no deseados, será posible detectarlos rápidamente, lo cual permitirá implementar las acciones necesarias en una fase temprana del desarrollo.

A continuación, se presentan una serie de ventajas y desventajas que presentan las prácticas de DC:

Ventajas:

Automatiza el proceso de lanzamiento de software:

Proporciona un mètodo para que el equipo de desarrollo registre el código que se crea, prueba y prepara automáticamente para su lanzamiento a producción, de forma tal que la entrega del software sea eficiente, resistente, rápida y segura.

Mejora la productividad del desarrollador:

Las prácticas de DC ayudan a la productividad del equipo, liberando a los desarolladores de las tareas manuales, eliminando dependencias complejas y centrándose en la entrega de nuevas funciones en el software.

Mejora la calidad del código:

Permite descubrir y abordar errores al principio del proceso de entrega, antes de que estos se conviertan en errores más grandes en etapas posteriores.

El equipo puede realizar fácilmente tipos adicionales de pruebas de código, ya que todo el proceso se ha automatizado. Gracias a esto, los equipos pueden iterar más rápido con comentarios inmediatos sobre el impacto de los cambios.

Los desarrolladores podrán identificar, a través de estos comentarios si el nuevo código introducido funciona, si hubo cambios importantes o errores en este.

Entrega actualizaciones más rápido:

Ayuda al equipo a entregar actualizaciones al cliente de manera más rápida y con mayor frecuencia.

Desventajas:

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.

Implementación manual:

A la hora de implementar innovaciones, mejoras o modificaciones en el producto, sigue siendo necesario hacerlo de forma manual.

Si se quiere automatizar este proceso, es necesario recurrir al despliegue continuo.

Pruebas sin errores:

Las pruebas automatizadas deben funcionar a la perfección, sin presentar errores de código. En caso contrario, pueden llegar a ocasionar graves daños al ser ejecutadas.

Mayor participación del cliente:

El cliente debe estar dispuesto a usar el software cuando todavía se encuentra en fase de desarrollo.

Además, debe poner de su parte mediante el feedback, lo que le permite al equipo mejorar el software de forma inmediata tras cada modificación introducida en el código fuente.

D. Relación entre Delivery continuo y Despliegue continuo.

En la actualidad, las empresas cambian rápidamente ante los escenarios competitivos del mercado. Este desafío se traslada además hacia los productos de Software.

El delivery y el despliegue continuo son prácticas que permiten cambios rápidos de software mientras se mantiene la estabilidad y seguridad del sistema. Aunque suelen utilizarse en conjunto, no necesariamente debe ser así. Por esto, comúnmente se confunden entre sí. La diferencia entre ellos radica en la forma de poner el Software en producción.

En una implementación continua, el código se puede probar, examinar y liberar de manera automática hacia el entorno de producción listo para ser usado por los clientes y usuarios.

Cada cambio pasa por pipelines automatizados para ser puestos en el entorno de producción lo que permite un ciclo continuo y temprano de retroalimentación del cliente en el ciclo de vida del producto.

Una idea errónea acerca de la entrega continua es que significa que cada cambio comprometido se aplica a la producción inmediatamente después de pasar las pruebas automatizadas. Sin embargo, el punto no es aplicar todos los cambios a la producción inmediatamente, sino garantizar que todos los cambios estén listos para pasar a la misma.

El código fluye automáticamente a través de múltiples pasos para prepararlo para el lanzamiento de producción, pero no se activa automáticamente. Los cambios primero deben ser aprobados, y es probable que haya pruebas manuales y control de calidad. La aprobación es responsabilidad del Dueño del Producto, quien decide que está listo para el entorno de producción. La decisión de lanzarse se convierte en una decisión comercial, no técnica.

Para hacer implementación continua, necesariamente se necesita haber realizado entrega continua, y en ambos casos, haber realizado integración continua.

E. Implementación de tuberías

Podemos definir a la Implementación de Tuberías, de una manera abstracta, como “el proceso para llevar el software del control de versiones a las manos de sus usuarios. Cada cambio en su software pasa por un proceso complejo en su camino a ser puesto en producción. Ese proceso implica la construcción del software, seguido del progreso de estas compilaciones a

través de múltiples etapas de prueba e implementación. Esto, a su vez, requiere la colaboración entre muchas personas y quizás varios equipos [3]. Pasa por todos los procesos necesarios para transforma los requerimientos del cliente un producto de software puesto en producción, en una versión estable, listo para los usuarios.

Proceso del Implementación de Tubería

1. El proceso comienza con los desarrolladores que confirman los cambios en su sistema de control de versiones. En este punto, el sistema de gestión de integración continua responde a la confirmación activando una nueva instancia de nuestra cartera. La primera etapa (commit) de la tubería realiza los siguientes pasos básico:

- Compila el código.
- Ejecuta pruebas unitarias.
- Realiza análisis de código.
- Crea instaladores.

Si la unidad prueba todos los pasos y el código es estable, reunimos el código ejecutable (binarios) y los almacenamos en un repositorio de artefactos. Los servidores CI modernos proporcionan una instalación para almacenar artefactos como estos y hacerlos fácilmente accesibles tanto para los usuarios como para las etapas posteriores del proceso de tubería.

2. Pruebas de aceptación automatizadas de mayor duración. Una vez más, su servidor CI debería permitirle dividir estas pruebas en paquetes que se puedan ejecutar en paralelo para aumentar su velocidad. Esta etapa se activará automáticamente al completar con éxito la primera etapa del proceso de tubería. En este punto, se ramificará en:
    1.UAT (prueba de aceptación del usuario),
    2.Prueba de Capacidad.
    3.Producción.

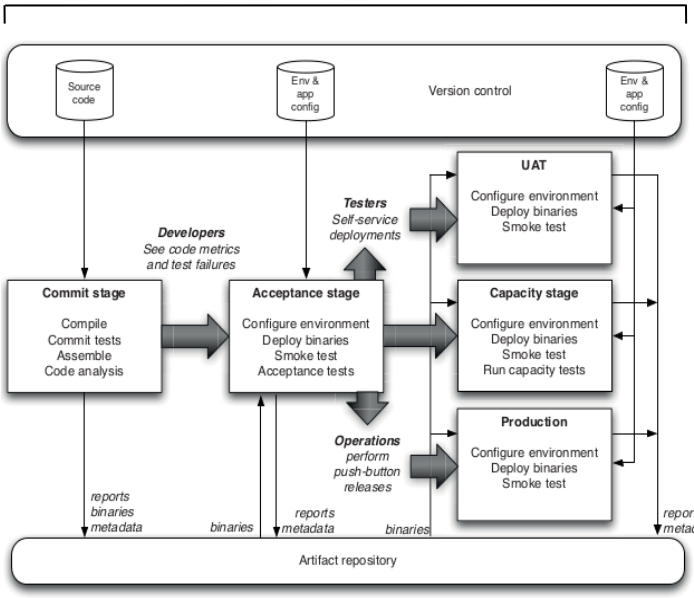

Fig. 2. Estructura de Las Tuberías de Implementación Básica. [3]

1) Buenas Prácticas de Tuberías de Implementación

Prácticas que se deben seguir para una buena implementación de tuberías.

- **Solo construya sus binarios una vez.** Mantenga siempre una única versión del código ejecutable o archivos binarios (jar, .NET, .so, etc). Los archivos binarios que se implementan en la producción deben ser exactamente los mismos que pasaron por el proceso de prueba de aceptación. Cada vez que compila el código, corre el riesgo de introducir alguna diferencia. La versión del compilador instalada en las etapas posteriores puede ser diferente de la versión que utilizó para sus pruebas de confirmación, verifique las versiones o actualizaciones, por ejemplo sus librerías, de los compiladores que utilice.

- **Implemente la misma manera en todos los entornos.** Es esencial utilizar el mismo proceso para implementar en todos los entornos, ya sea la estación de trabajo de un desarrollador o analista, un entorno de prueba o producción, a fin de garantizar que el proceso de construcción e implementación se pruebe de manera efectiva.

- **Prueba de humo de las implementaciones.** La prueba de humo[1] o prueba de implementación, es probablemente la prueba más importante para escribir una vez que tiene un conjunto de pruebas unitarias en funcionamiento, de hecho, podría decirse que es aún más importante. Le da la confianza de que su aplicación realmente se ejecuta. Si no funciona, su prueba de humo debería poder darle algunos diagnósticos básicos sobre si su aplicación está inactiva porque algo de lo que depende no funciona.

- **Implementar en una copia de producción.** Para obtener un buen nivel de confianza de que la puesta en marcha realmente funcionará, debe realizar sus pruebas e integración continua en entornos que sean lo más similares posible a su entorno de producción.

- **Cada cambio debe propagarse a través de la tubería al instante**. La Tubería de implementación adopta un enfoque diferente: la primera etapa debe activarse en cada check-in, y cada etapa debe activar la siguiente inmediatamente después de completar con éxito. Por supuesto, esto no siempre es posible cuando los desarrolladores (especialmente en equipos grandes) se registran con mucha frecuencia, dado que las etapas de su proceso pueden llevar una cantidad de tiempo no insignificante.

[1] "***Subconjunto de todos los casos de prueba definidos/planificados que cubren la funcionalidad principal de un componente o sistema, con el objeto de asegurar que las funciones cruciales de un programa funcionan, pero sin preocuparse por los detalles finos. Una construcción diaria y pruebas de humo pertenecen a las mejores prácticas de la industria*". [4]**

- **Si alguna parte de la tubería falla, pare la línea.** Si falla una implementación en un entorno, todo el equipo es el propietario de esa falla. Deben detenerse y arreglarlo antes de hacer cualquier otra cosa.

2) Implementación de una tubería de implementación

Contiene los siguientes pasos:

1. **Modelando su flujo de valor y creando un esqueleto ambulante.** el primer paso es realizar el mapa de flujo de valor que va desde el requerimiento hasta la puesta de producción. Una vez que tenga un mapa de flujo de valor, puede seguir adelante y modelar su proceso en su herramienta de integración continua y administración de versiones.

2. **Automatizar el proceso de construcción e implementación.** El siguiente paso en la implementación de una tubería es automatizar el proceso de construcción e implementación. El proceso de compilación toma el código fuente como entrada y produce binarios como salida. La característica clave de los archivos binarios(ejecutables) es que deben poderse copiar en una nueva máquina y, dado un entorno adecuadamente y la configuración correcta para la aplicación en ese entorno, se ejecute su aplicación, sin depender de ninguna parte de su cadena de herramientas de desarrollo. instalado en esa máquina. Su servidor CI debe estar configurado para:

   - Vigilar su sistema de control de versiones, verificar o actualizar su código fuente cada vez que se realice un cambio en él, ejecute el proceso de compilación automatizado y almacene los archivos binarios en el sistema de archivos donde están accesibles para todo el equipo a través de la interfaz de usuario del servidor CI.

   - El siguiente paso es automatizar la implementación. En primer lugar, debe obtener una máquina para implementar su aplicación. Para un nuevo proyecto, esta puede ser la máquina en la que se encuentra su servidor de integración continua. Para un proyecto que es más maduro, es posible que necesite encontrar varias máquinas. Dependiendo de las convenciones de su organización, este entorno se puede denominar entorno de pruebas de aceptación de usuario (UAT).

   - Una vez que el proceso de implementación de su aplicación está automatizado el siguiente paso es poder realizar implementaciones de botón en su entorno UAT. Configure su servidor CI para que pueda elegir cualquier compilación de su aplicación y haga clic en un botón para activar un proceso que tome los binarios producidos por esa compilación, ejecute el script que implementa la compilación y ejecute la prueba de implementación. Asegúrese de que al desarrollar su sistema de compilación y despliegue haga uso de los principios que describimos, como construir sus archivos binarios solo una vez y separar la configuración de los archivos binarios, de modo que se puedan usar los mismos archivos binarios en todos los entornos. Esto asegurará que la gestión de la configuración de su proyecto se ponga en pie. Excepto para el software instalado por el usuario, el proceso de producción debe ser el mismo proceso que utiliza para implementar en un entorno de prueba. Las únicas diferencias técnicas deberían estar en la configuración del entorno.

3. **Automatizar pruebas unitarias y análisis de código.** Esto significa ejecutar pruebas unitarias, análisis de código y, en última instancia, una selección de pruebas de aceptación e integración en cada check-in. La ejecución de pruebas unitarias no debería requerir una configuración compleja, porque las pruebas unitarias, por definición, no dependen de la ejecución de su aplicación.

4. **Automatizar las pruebas de aceptación.** La fase de prueba de aceptación puede reutilizar para implementar en su entorno de prueba. La única diferencia es que después de ejecutar las pruebas de humo, el marco de prueba de aceptación debe iniciarse y los informes que genera deben recopilarse al final de la prueba para su análisis. También tiene sentido almacenar los registros creados por su aplicación. Las pruebas de aceptación se dividen en dos tipos: funcionales y no funcionales. Es esencial comenzar a probar parámetros no funcionales como la capacidad y las características de escala desde el principio en cualquier proyecto, para que tenga una idea de si su aplicación cumplirá con sus requisitos no funcionales. Es perfectamente posible ejecutar pruebas de aceptación y pruebas de rendimiento consecutivas como parte de una sola etapa. Luego puede separarlos para poder distinguir fácilmente qué conjunto de pruebas falló. Debe intentar obtener al menos uno o dos de cada tipo de prueba que necesita para ejecutarse de manera automatizada al principio de la vida de su proyecto e incorporarlos al canal de implementación. Por lo tanto, tendrá un marco que facilita agregar pruebas a medida que su proyecto crece.

5. **Automatizar Producción.** Realizar algún proceso (como por ejemplo un script) que automatice la puesta en producción, siempre que los resultados de los pasos anteriores se cumplieron de manera exitosa. [3]

3) Evolucionando el proceso de Tubería

Los pasos que describimos anteriormente se encuentran en casi todos los flujos de valor y, por lo tanto, en la canalización que hemos visto. Suelen ser los primeros objetivos para la automatización. A medida que su proyecto se vuelva más complejo, su flujo de valor evolucionará. Hay otras dos

extensiones potenciales comunes a la tubería: componentes y ramas. Las aplicaciones grandes se construyen mejor como un conjunto de componentes que se ensamblan juntos. En tales proyectos, puede ser necesario dividir el proceso de tubería, en otros procesos más pequeños, para cada componente, y luego una tubería que ensamble todos los componentes y someta a toda la aplicación a pruebas de aceptación, pruebas no funcionales y luego despliegue en entornos de prueba, preparación y producción.

La implementación de la tubería variará enormemente entre proyectos, pero las tareas en sí son consistentes para la mayoría de los proyectos. Usarlos como patrón puede acelerar la creación del proceso de compilación e implementación para cualquier proyecto. Sin embargo, en última instancia, el objetivo de la tubería es modelar su proceso para construir, implementar, probar y liberar su aplicación. Luego, la tubería garantiza que cada cambio pueda pasar por este proceso de forma independiente de la manera más automatizada posible.

A medida que implemente el proceso, se descubre que las conversaciones que mantiene con las personas involucradas y las ganancias en eficiencia que usted percibe, a su vez, tendrán un efecto en su proceso. Por lo tanto, es importante recordar tres cosas.

1. No es necesario implementar toda la tubería de una vez. Debe implementarse de forma incremental. Si hay una parte de su proceso que actualmente es manual, cree un marcador de posición en su flujo de trabajo. Asegúrese de que su implementación registre cuando este proceso manual se inicie y cuando se complete. Esto le permite ver cuánto tiempo se dedica a cada proceso manual y, por lo tanto, calcular en qué medida es un cuello de botella.

2. Su canalización es una rica fuente de datos sobre la eficiencia de su proceso para crear, implementar, probar y liberar aplicaciones. La implementación de la tubería que cree debe registrar cada vez que un proceso comienza y finaliza, y cuáles fueron los cambios exactos que pasaron por cada etapa de su proceso. Estos datos, a su vez, le permiten medir el tiempo del ciclo desde la confirmación de un cambio hasta su implementación en la producción, y el tiempo dedicado a cada etapa del proceso. Por lo tanto, es posible ver exactamente cuáles son los cuellos de botella de su proceso y atacarlos en orden de prioridad.

3. Finalmente, su canal de implementación es un sistema vivo. A medida que trabaja continuamente para mejorar su proceso de entrega, debe continuar ocupándose de su canal de implementación, trabajando para mejorarlo y refactorizarlo de la misma manera que trabaja en las aplicaciones que está utilizando para entregar.

4) Métrica

La retroalimentación es el corazón de cualquier proceso de entrega de software. La mejor manera de mejorar la retroalimentación es hacer que los ciclos de retroalimentación sean cortos y los resultados sean visibles. Debe medir continuamente y transmitir los resultados de las mediciones de una manera sensilla.

¿Qué debe medir? Una buena técnica es centrarse en la reducción del tiempo del ciclo, esto fomenta las prácticas que aumentan la calidad, como el uso de un conjunto completo de pruebas automatizadas que se ejecutan como resultado de cada check-in.

Una creación adecuada de una tubería de implementación debería facilitar el cálculo de la parte del tiempo de ciclo correspondiente a la parte de la secuencia de valor desde el requerimiento hasta la puesta en producción. También debería permitirle ver el tiempo de espera desde el check-in hasta cada etapa de su proceso, para que pueda descubrir sus cuellos de botella.

Una vez que conozca el tiempo de ciclo para su aplicación, puede determinar la mejor manera de reducirlo. Puede usar la Teoría de restricciones para hacer esto aplicando el siguiente proceso.

1. Identifique la restricción limitante en su sistema. Esta es la parte de su proceso de compilación, prueba, implementación y ejecución que es el cuello de botella. Para elegir un ejemplo al azar, quizás sea el proceso de prueba manual.
2. Explotar la restricción. Esto significa asegurarse de que debe maximizar el rendimiento de esa parte del proceso. Un ejemplo (prueba manual), se aseguraría de que siempre haya un búfer de historias en espera de ser probado manualmente, y se aseguraría de que los recursos involucrados en la prueba manual no se usen para nada más.
3. Subordinar todos los demás procesos a la restricción. Esto implica que otros recursos no funcionarán al 100%; por ejemplo, si sus desarrolladores trabajan desarrollando historias a plena capacidad, la acumulación de historias que esperan ser probadas continuará creciendo. En su lugar, haga que sus desarrolladores trabajen lo suficiente para mantener constante el trabajo atrasado y pasen el resto de su tiempo escribiendo pruebas automatizadas para detectar errores, de modo que se dedique menos tiempo a las pruebas manualmente.
4. Elevar la restricción. Si el tiempo de su ciclo aún es demasiado largo, debe aumentar los recursos disponibles: contratar más evaluadores o quizás invertir más esfuerzo en pruebas automatizadas.
5. Enjuague y repita. Encuentre la siguiente restricción en su sistema y regrese al paso 1.

Si bien el tiempo de ciclo es la métrica más importante en la entrega de software, hay una serie de otros diagnósticos que pueden advertirle de problemas. Éstos incluyen:

- Cobertura de prueba automatizada
- Propiedades de la base de código, si cumple con los patrones o principios básicos según el lenguaje que utilice, etc.

- Número de defectos.
- La velocidad a la que su equipo entrega código de trabajo, probado y listo para usar
- Número de confirmaciones al sistema de control de versiones por día
- Número de compilaciones por día
- Número de fallas de compilación por día
- Duración de la compilación, incluidas las pruebas automatizadas

F. Herramientas para la aplicación del delivery continuo

1) Jenkins

Es una aplicación web que facilita la integración continua de los componentes del software. Está escrito en Java, se ejecuta en un contenedor EJB y dispone de varias herramientas de build (Apache Ant, Maven/Gradle, CVS, Subversion, Git, etc.) y, por supuesto, de los procesos de pruebas automáticas tan importantes en el caso del delivery continuo (Junit, Emma). Existen plugins opcionales que sirven para garantizar la compatibilidad con otros compiladores. Gracias a su interfaz basada en REST, otros programas pueden acceder a Jenkins. Es un programa gratuito de código abierto. Está recomendado especialmente para principiantes porque su interfaz y las funcionalidades que ofrece son muy intuitivas y fáciles de usar [1].

2) CircleCi

Es una aplicación web para integración, entrega y despliegue continuo. Funciona preferiblemente con GitHub, GitHub Enterprise y Bitbucket. Además, esta plataforma ofrece muchas funcionalidades prácticas como la gestión de pedidos, gestión de recursos, docker support, soporte de todos los lenguajes de programación conocidos, almacenamiento seguro en caché, análisis de datos con estadísticas y completos conceptos de seguridad. CircleCI recibió el premio "Leader in Continuous Integration" de Forrester en 2017 [1].

3) Microsoft Team Foundation Server (TFS):

Es una herramienta colaborativa para proyectos de software que requieren planificación y desarrollo. Soporta varios procesos de desarrollo, incluido el CMMS, el desarrollo de software ágil y Scrum. TFS divide el proceso completo en varios subprocesos: control de versiones, build, reports y administración de usuario [1].

4) Codeship

Es una plataforma SaaS de integración y entrega continua que se adapta a las necesidades particulares de cada usuario. Soporta GitHub, Bitbucket y GitLab, pudiendo utilizar a los siguientes lenguajes: Ruby; Node; PHP; Python; Java y Go [1].

G. DevOps y entrega continua

DevOps es una metodología para ayudar a las organizaciones a construir equipos y software. DevOps es una cultura, movimiento o práctica que enfatiza la colaboración y la comunicación tanto de los desarrolladores de software como de otros profesionales de TI, a la vez que automatiza el proceso de entrega de software y los cambios en la infraestructura, aumentando la capacidad de una organización para entregar aplicaciones y servicios a alta velocidad.

La entrega continua y los DevOps comparten características comunes. Ambas están orientadas al pensamiento ágil y lean: cada una busca ofrecer cambios pequeños y rápidos, cada una se basa en una estrecha colaboración de negocios de TI, y cada una comparte el objetivo común de acelerar el tiempo de lanzamiento al mercado de nuevos servicios.

Sin embargo, el objetivo de DevOps es fusionar los roles de dev y ops, y los procesos que gestionan, para alcanzar los objetivos de negocio. Se trata de una cultura común y compartida y de una mayor colaboración. Sobre procesos de negocio claramente definidos.

DevOps combina la integración continua (CI) y la entrega continua (CD), y las prácticas de CI / CD mejoran la velocidad para entregar un nuevo servicio y satisfacer las necesidades del cliente. [5]

APÉNDICE: GESTION DEL DELIVERY CONTINUO

TABLA I
MODELO DE MADUREZ PARA EVALUAR EL PROCESO Y GESTIONAR LA EVOLUCIÓN DEL MISMO.

	Gestión de builds e integración continua	**Entornos y despliegue**	**Gestión de entregas y conformidad**	**Testing**	**Gestión de los datos**	**Gestión de la configuración**
Nivel 3- Optimizando Foco en la mejora de proceso	Equipos se reúnen regularmente para discutir problemas de integración y resolverlos con automatización, feedback más temprano y visibilidad.	Todos los entornos gestionados eficazmente. Aprovisionamiento automatizado.	Operaciones y desarrollo colaboran para gestionar riesgos y reducir tiempo de ciclo.	Rollbacks de producción infrecuentes. Los defectos que se encuentran se arreglan de inmediato.	Feeedback sobre el rendimiento de la base de datos y proceso de despliegue entre entregas.	Validación frecuente de que las políticas promueven la colaboración, el desarrollo ágil y un proceso de gestión de cambios auditable.
Nivel 2- Gestionado en gran parte Proceso medido y controlado	Métricas sobre las builds visibles y gestionadas. Las builds no quedan rotas mucho tiempo.	Despliegues a los entornos gestionados. Procesos de entrega y rollback probados.	Salud del entorno y la aplicación monitorizada y gestionada al igual que el tiempo de ciclo.	Métricas de calidad y tendencias medidas. Requerimientos no funcionales definidos y medidos.	Actualizaciones y rollbacks de base de datos probados en cada despliegue. Rendimiento de la base de datos monitorizado y optimizado.	Desarrolladores hacen commit a trunk al menos una vez al día. Las ramas se usan solo para las entregas.
Nivel 1 - Consistente Procesos automatizados en todo el ciclo de vida	Build automática y tests automáticos pasados en cada commit. Dependencias gestionadas. Scripts y artefactos reutilizados.	Despliegue a cualquier entorno totalmente automatizado.	Procesos de gestión de cambio y aprobaciones definido y aplicado. Se cumplen con las condiciones regulatorias de conformidad.	Unit tests. Tests de aceptación escritos con testers. El testing forma parte del desarrollo.	Cambios en la base de datos se aplican como parte del proceso de despliegue.	Librerías y dependencias gestionadas. Políticas de uso del control de versiones en el proceso.
Nivel 0- Repetible Proceso documentado y parcialmente automatizado	Builds automáticas frecuentes. Cualquier build se puede recrear mediante el control de versiones y un proceso automático.	Despliegue automático a algunos entornos. Crear nuevos entornos es trivial. Configuración versionada.	Entregas fiables pero infrecuentes y dolorosas. Trazabilidad limitada de los requisitos a la entrega.	Tests automáticos escritos como parte del desarrollo.	Cambios en la base de datos mediante scripts versionados con la aplicación.	Control de versiones con todo para recrear el software: código, configuración y scripts de build, despliegue y datos.
Nivel -1 - Regresivo Procesos no repetibles, descontrolados y reactivos	La build es un proceso manual. Los artefactos no se gestionan.	Proceso manual de despliegue. Binarios propios de cada entorno. Entornos también manuales.	Entregas no fiables e infrecuentes.	Testing manual después del despliegue.	Cambios en la base de datos manuales y sin versionar.	Sin control de versiones o usado parcialmente.

[7]

ACLARACIÓN

El presente trabajo informe que fue redactado y estructurado teniendo en cuenta los lineamientos de informe técnico planteado por la IEEE.

Disponible:

https://www.academia.edu/37518280/Plantilla_en_Word_para_Trabajos_Tecnicos_IEEE

REFERENCIAS

[1] Continuous Delivery – Pipelines en el desarrollo de Software, Mayo, 2019.Disponible: https://www.ionos.es/digitalguide/paginasweb/desarrollo-web/continuous-delivery/

[2] Amazon Web Services, Inc, "Practicing Continuous Integration and Continuous Delivery on AWS", Junio 2017.

[3] Continuous Delivery- Reliable Software Realeses Through Build, Test And Deployment Automation. Editorial Addison-Wesley – Jez Humble, David Farley -2011.

[4] El Comité Internacional de Cualificación de Pruebas de Software (ISTQB: International Software Testing Qualification Board, www.istqb.org). SSTQB (o Comité Español de Testing) http://www.sstqb.es/

[5] DevOps and Continuous Delivery:Not the Same. Michael Schmidt, Abril,2016. Disponible: https://devops.com/devops-and-continuous-delivery-not-same/

[6] Continuous Delivery Principles. Mark Rehkopf, Atlassian.

[7] Resúmen de Continuous Delivery, Samuel Casanova. Disponible: https://samuelcasanova.com/2017/12/resumen-continuous-delivery/