CONTINUOUS DELIVERY (ENTREGA CONTINUA)

Raúl Mínguez Rodríguez
Eric Almeda Tomás
Xurde García Fernández

Introducción

Se trata de una disciplina de producción de código basada en un desarrollo medido, controlado y actualizado en todo momento, que prioriza la posibilidad de disponer de continuo de un sistema completamente funcional con todos los elementos ya terminados disponibles. Pretende ofrecer la capacidad de integrar cambios de variada índole con la mayor velocidad posible, sin afectar al funcionamiento del programa.

Los desarrolladores siguiendo un sistema de entrega continua buscan programar secciones de código más pequeñas y específicas, que son más fácilmente implementables en las versiones lanzadas del sistema. Esto permite, además, mantener un control de fallos sobre porciones de código no muy grandes, minimizando los tiempos muertos por mantenimiento. De forma adicional, la respuesta frente a cambios solicitados por el cliente o partes interesadas tiene tiempos de consecución por lo general bajos, y la posibilidad de recibir retroalimentación en tiempo real reduce la disconformidad del solicitante, permitiendo suministrar la funcionalidad necesaria de manera gradual y constante.

Todo el proceso tiene una fuerte base de automatización, buscando integrar el desarrollo, pruebas y lanzamiento en una secuencia definida. Dicha automatización reduce los tiempos de trabajo de los desarrolladores sin lastrar su productividad, evitando también que los mismos se encuentren en situaciones de tiempo muerto e incrementando el tiempo útil de desarrollo. Además, permite controlar diferentes tipos de fallos asociados al trabajo manual, al poder automatizar comprobaciones de casos de error típicos, pudiendo trazar con facilidad las excepciones que se diesen.

Etapas de desarrollo

El desarrollo del producto en continous delivery se distribuye en una “pipeline” que puede ser más o menos compleja dependiendo del producto. Hay 4 fases que suelen ser comunes en continous delivery, éstas no tienen por qué ser fases físicas sino más bien lógicas.

- **Component phase:** Primero se construyen componentes, es decir unidades de producto pequeñas y testeables. Estos componentes pueden ser validados con revisiones de código, test unitarios y analizadores de código estáticos. Las **revisiones de código** son importantes en esta fase para que el equipo conozca las necesidades del producto, las diferentes perspectivas pueden ayudarnos a revisar nuestro código y fortalecer las debilidades que encontremos. Los **test unitarios** suelen ser la primera fase en el test de software, estos no tocan la base de datos ni la red. La cobertura de código se conoce como el porcentaje de código que usan los test unitarios, puede ser medido por líneas, clases, métodos… Los **analizadores de código estáticos,** detectan problemas en el código sin necesidad de ejecutarlo, es una manera barata de detectar errores, detectan cosas como fugas de memoria, complejidades temporales altas o duplicidad de código.
- **Subsystem phase:** Los componentes débilmente acoplados crean subsistemas que son la unidad desplegable más pequeña. A diferencia de los componentes, estos se pueden validar con los casos de uso de los clientes. Estos pueden ser comprobados con tests funcionales, de rendimiento y de seguridad.
- **System phase:** Una vez que los subsistemas pasan las pruebas, se empieza a formar un sistema formado por los diferentes subsistemas. En esta parte validaremos el sistema en su conjunto con tests de Integración, rendimiento y seguridad.
- **Production phase:** En esta fase desplegaremos los subsistemas o el sistema entero si así lo requiere el producto. Algo que deberemos llevar a cabo es evitar la pérdida de tiempo de los clientes (ZDD). Una técnica a llevar a cabo es “Blue-green deployment”, donde las nuevas versiones están desplegadas para una pequeña parte de los usuarios (parte verde) mientras que la gran mayoría están contentos con las versiones viejas y funcionales (azul). El objetivo es, si cometemos algún error en la parte verde podemos revertirlo volviendo a la azul, mientras que si las cosas van como deberían en la verde podemos ir moviendo a todos los demás hacia esta. En esta fase son útiles los smoking test.

Comparativa con otros sistemas

Hay algunos términos como DevOps, Continuous integration y Continuous deployment que se pueden confundir con Continuous delivery pero NO son lo mismo.

Continuous delivery: el software se construye de tal manera que siempre está listo para ser llevado a producción. Esto solo ocurre cuando el Product Owner decide pasar a producción y no cuando se puedan llevar nuevas funcionalidades a producción, es decir, cuando haya cambios.

Continuous deployment: cada cambio pasa por un “pipeline” automatizado y se pone en producción. Es decir, permite realizar varios despliegues al día que proporcionan rápido feedback. Se diferencia del Continuous delivery en que sólo implica que somos capaces de hacer entregas cada poco, pero no es necesario hacerlas. Se puede decir que el continuous deployment es un paso más avanzado que el continuous delivery.

Continuous integration: práctica que se basa en integrar cambios en el código de la forma más rápida posible al proceso de desarrollo. Asegurando que estos cambios no provocan fallos en la aplicación. Integración, construcción y prueba frecuente automatizada. El continuous delivery y continuous deployment se basan en esta práctica.

DevOps: viene de Development + Operations, es decir, conseguir la máxima colaboración e integración entre desarrollo de software y operaciones (producción). Se caracteriza por una cultura de colaboración entre los equipos, que en otras prácticas están separados por departamentos, hay un único equipo autónomo.

Bibliografía

https://www.atlassian.com/continuous-delivery/pipeline
https://inlogiq.com/continuous-integrations-continuous-delivery-continuous-deployment-devops-lo/#:~:text=Continuous%20integration%3A%20com%C3%BAnmente%20nombrado%20como,posible%2C%20al%20proceso%20de%20desarrollo
https://www.javiergarzas.com/2016/01/devops-continuous-delivery-continuous-deployment-integracion-continua-aclarando-terminos.html