Integración Continua

Universidad Tecnológica Nacional – Facultad Regional Córdoba – Ingeniería de Software – Grupo 01 – 4K1 - Año 2019

Belloni, Matías
Universidad Tecnológica Nacional,
Facultad Regional Córdoba
Córdoba, Argentina
matibellonii@gmail.com

Fernandez, Javier Andres
Universidad Tecnológica Nacional,
Facultad Regional Córdoba
Córdoba, Argentina
cefe1114@gmail.com

Herrera Barón, Gastón
Universidad Tecnológica Nacional,
Facultad Regional Córdoba
Córdoba, Argentina
gaston.herrerabaron@gmail.com

Loza, Leonardo
Universidad Tecnológica Nacional,
Facultad Regional Córdoba
Córdoba, Argentina
leonardoloza98@gmail.com

Maldonado, Paula
Universidad Tecnológica Nacional,
Facultad Regional Córdoba
Córdoba, Argentina
maldonadopaula541@gmail.com

Resumen: ***Debido a la competencia del mercado de desarrollo de software han surgido diferentes prácticas que ayudan a las organizaciones a entregar productos de calidad rápidamente. Una de ellas es la Integración Continua. A continuación, abordaremos temas claves para su comprensión, tales como su definición, la importancia de implementarla, y en qué consiste su implementación, como así también las diversas prácticas que conforman el uso de la técnica de forma efectiva. A su vez mencionaremos los principales beneficios que conlleva aplicar Integración Continua, los desafíos para su implementación y realizaremos una comparativa con la Implementación Continua y Entrega continua, detallando de forma breve en qué consiste cada técnica y sus diferencias. Para concluir enunciaremos algunas de las herramientas más utilizadas para hacer uso de la integración continua con sus principales características.***

Palabras clave: integración continua, herramientas, prácticas.

I. INTRODUCCIÓN

La integración continua se la puede considerar como una técnica que combina continuamente actualizaciones de código fuente de todos los desarrolladores de un equipo en una línea principal compartida.

Esta fusión continua evita que la copia local de un desarrollador de un proyecto de software se aleje demasiado a medida que otros agregan un nuevo código, evitando conflictos de fusión catastróficos**.**

Sus objetivos consisten en encontrar y arreglar errores con mayor rapidez, mejorar la calidad del software y reducir el tiempo que se tarda en validar y publicar nuevas actualizaciones de software.

Sin la Integración Continua, los desarrolladores deben coordinar y comunicarse manualmente cuando están contribuyendo con código al producto final. Además, la sobrecarga de comunicación de un entorno sin Integración Continua puede convertirse en una tarea de sincronización compleja y enredada, que agrega un costo innecesario a los proyectos. Esto provoca lanzamientos de código más lentos con mayores tasas de falla, ya que requiere que los desarrolladores sean sensibles y reflexivos sobre las integraciones.

II. ¿POR QUÉ ES NECESARIA LA INTEGRACIÓN CONTINUA?

Anteriormente, era común que los desarrolladores de un equipo trabajen aislados durante un largo periodo de tiempo y solo intentarán combinar los cambios en la versión maestra una vez que habían completado el trabajo. Como consecuencia, la combinación de los cambios en el código resultaba difícil y ardua, además de dar lugar a la acumulación de errores durante mucho tiempo que no eran corregidos. Estos factores hacían que resultase más difícil proporcionar las actualizaciones a los clientes con rapidez.

Sin una sólida canalización de Integración Continua se puede formar una desconexión entre el equipo de desarrollo y el resto de la organización. La comunicación entre producto e ingeniería puede ser engorrosa debido a que la misma se convierte en una caja negra en la que el resto del equipo ingresa los requisitos y características y tal vez recupera los resultados esperados. A la ingeniería le resultará más difícil estimar el tiempo de entrega de las solicitudes porque el tiempo para integrar nuevos cambios se convierte en un riesgo desconocido.

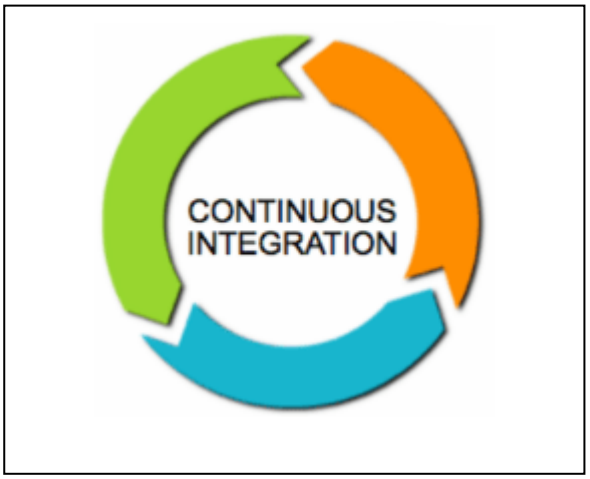


Fig. 1 Integración continua

III. ¿EN QUÉ CONSISTE LA INTEGRACIÓN CONTINUA?

La integración continua es una práctica en la que los miembros de un equipo integran y fusionan el trabajo de desarrollo (por ejemplo, código) con frecuencia, por ejemplo, varias veces al día. En esta práctica se debe:

1) Construir el sistema: Esto incluye tanto compilar su código como reconstruir sus bases de datos de prueba. Aunque esto parezca una tarea fácil, para sistemas de gran tamaño, que se encuentran formados por muchos subsistemas, se necesita una estrategia sobre cómo se van a construir tanto los subsistemas como el sistema en general.

2) Ejecutar conjunto(s) de prueba de regresión: Cuando la compilación es exitosa, el siguiente paso es ejecutar conjuntos de prueba de regresión. Los conjuntos que ejecute estarán determinados por el alcance de la compilación y el tamaño del sistema. Para sistemas pequeños, es probable que tenga un único conjunto de pruebas, pero en situaciones más complejas tendrá varios conjuntos de pruebas por razones de rendimiento. Se pueden simular partes del sistema, desde un punto de vista de rendimiento, como una base de datos o un subsistema externo. Si está utilizando simulacros, o si está probando una parte de la funcionalidad, necesitará uno o más conjuntos de pruebas de mayor alcance. Como mínimo, necesitará un conjunto de pruebas que se ejecute contra los componentes reales del sistema, no simulacros, y que ejecute todas sus pruebas. Si este conjunto de pruebas se ejecuta en unas pocas horas, entonces puede ejecutarse todas las noches, si lleva más tiempo, entonces querrá reorganizarlo nuevamente para que las pruebas de larga duración se encuentren en otro conjunto de pruebas que se ejecuta de manera irregular.

3) Realizar análisis estático: El análisis estático es una técnica de calidad en la que una herramienta automatizada busca defectos en el código, a menudo buscando tipos de problemas como defectos de seguridad o problemas de estilo de codificación (por nombrar algunos).

Su trabajo de integración podría ejecutarse en momentos específicos, tal vez una vez por hora, o cada vez que alguien registre una nueva versión de un componente (como el código fuente) que forma parte de la compilación.

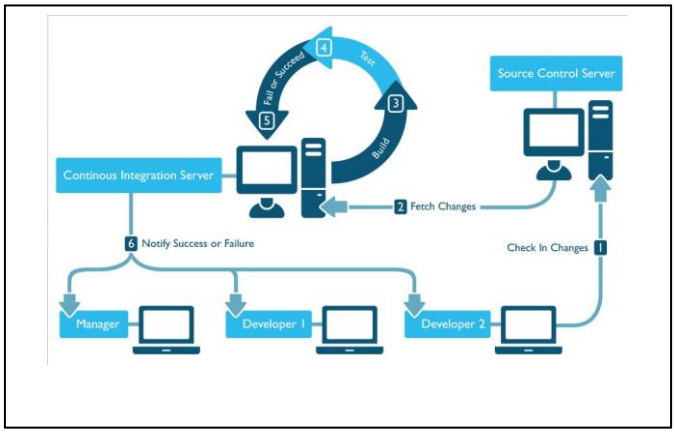


Fig. 2. Pasos de la integración continua

IV. PRÁCTICAS QUE CONFORMAN UNA INTEGRACIÓN CONTINUA EFECTIVA

A. *Mantener un repositorio de fuente única*

Los proyectos de software implican muchos archivos que deben organizarse juntos para crear un producto. Hacer un seguimiento de todo esto es un gran esfuerzo, especialmente cuando hay varias personas involucradas. Por lo tanto, no es sorprendente que a lo largo de los años los equipos de desarrollo de software han creado herramientas para gestionar todo esto. Estas herramientas, llamadas herramientas de administración de código fuente, administración de configuración, sistemas de control de versiones, repositorios u otros nombres, son una parte integral de la mayoría de los proyectos de desarrollo.

Resulta esencial obtener un sistema de gestión de código fuente decente. El costo no es un problema ya que hay herramientas de código abierto de buena calidad disponibles. El repositorio más utilizado de código abierto actual es "Subversion", mientras que "Perforce" forma parte de los repositorios pagos más utilizados.

Aunque muchos equipos usan repositorios, un error común es que no ponen todo en el repositorio. Si las personas usan uno, pondrán código allí, pero todo lo que necesita para hacer una compilación debe estar allí, incluidos: scripts de prueba, archivos de propiedades, esquema de base de datos, scripts de instalación y bibliotecas de terceros. La regla básica es que debe poder caminar hasta el proyecto con una máquina virgen, realizar un pago y poder construir el sistema por completo. Solo una cantidad mínima de cosas debe estar en la máquina virgen, generalmente cosas que son grandes, complicadas de instalar y estables.

Debe colocar todo lo necesario para una compilación en el sistema de control de código fuente; sin embargo, también puede incluir otras cosas con las que la gente generalmente trabaja allí. Las configuraciones IDE son buenas porque allí es fácil para las personas compartir las mismas configuraciones IDE.

Una de las características de los sistemas de control de versiones es que le permiten crear múltiples ramas, para manejar diferentes flujos de desarrollo. Esta es una característica útil, no esencial, pero con frecuencia se usa en exceso y causa problemas a las personas. Se debe mantener el uso de ramas al mínimo. En particular, tienen una línea principal: una sola rama del proyecto actualmente en desarrollo. Casi todos deberían trabajar fuera de esta línea principal la mayor parte del tiempo.

En general, se debe almacenar en el control de origen todo lo que necesita para construir cualquier cosa, pero nada de lo que realmente construimos. Algunas personas mantienen los productos de compilación en el control de la fuente, pero ellos es una indicación de un problema más profundo, generalmente una incapacidad para recrear de manera confiable las compilaciones.

B. Automatizar la compilación.

Convertir las fuentes en un sistema en ejecución a menudo puede ser un proceso complicado que implica compilación, mover archivos, cargar esquemas en las bases de datos, etc. Sin embargo, como la mayoría de las tareas en esta parte del desarrollo de software, puede automatizarse da como resultado que debería automatizarse.

Los entornos automatizados para compilaciones son una característica común de los sistemas. El mundo Unix ha tenido éxito durante décadas, la comunidad Java desarrolló Ant, la comunidad .NET ha tenido Nant y ahora tiene MSBuild.

Un error recurrente es no incluir todo en la compilación automatizada. La compilación debe incluir sacar el esquema de la base de datos del repositorio y activarlo en el entorno de ejecución.

Hay varios scripts de compilación y, a menudo, son particulares de una plataforma o comunidad, aunque es posible utilizar otro. Por ejemplo, la mayoría de los proyectos Java usan Ant, aunque hay otros que usan Ruby (el sistema Ruby Rake es una herramienta de script de compilación).

Una construcción grande a menudo lleva tiempo, no se desea realizar una compilación completa si solo se ha realizado un pequeño cambio. Entonces, una buena herramienta de compilación analiza lo que debe cambiarse como parte del proceso. La forma común de hacer esto es verificar las fechas de los archivos fuente y objeto y solo compilar si la fecha fuente es posterior. Las dependencias se vuelven complicadas: si un archivo de objeto cambia, los que dependen de él también deben reconstruirse. Los compiladores pueden manejar este tipo de cosas, o no.

Dependiendo de lo que necesite, puede necesitar diferentes tipos de cosas para construir. Puede construir un sistema con o sin código de prueba, o con diferentes conjuntos de pruebas. Algunos componentes se pueden construir de forma independiente. Un script de compilación debería permitirle construir objetivos alternativos para diferentes casos.

C. Haga su autocomprobación de construcción.

Una compilación significa compilar, vincular y todas las cosas adicionales necesarias para ejecutar un programa. Aunque un programa puede ejecutarse, no significa que haga lo correcto. Los lenguajes modernos de tipo estático pueden detectar muchos errores, pero muchos más se escapan de esa red.

Una buena forma de detectar errores de forma más rápida y eficiente es incluir pruebas automatizadas en el proceso de compilación. Las pruebas no son perfectas, por supuesto, pero pueden detectar muchos errores, suficientes para ser útiles.

Para el código de autocomprobación, se necesita un conjunto de pruebas automatizadas que puedan verificar una gran parte del código en busca de errores. Estás pruebas deben poder iniciarse desde un comando simple y autocomprobarse. El resultado de ejecutar el conjunto de pruebas debe indicar si alguna prueba falló. Y además para que una compilación sea autocomprobada, el fallo de una prueba debe hacer que falle la compilación.

La familia XUnit de herramientas de código abierto, son ideales para este tipo de pruebas ya que facilitan la configuración de un entorno de autocomprobación.

Las herramientas de XUnit son sin duda el punto de partida para realizar la autocomprobación del código. También se pueden buscar otras herramientas que se centren en las pruebas de extremo a extremo, hay una gran variedad de estas como FIT, Selenium, Sahi , Watir , FITnesse

Las pruebas no aseguran la ausencia de errores, sin embargo, las pruebas imperfectas, ejecutadas con frecuencia, son mucho mejores que las perfectas que nunca se escriben en absoluto.

D. Todos se comprometen con la línea principal todos los días.

La integración se trata principalmente de comunicación, permite a los desarrolladores informar a otros desarrolladores sobre los cambios que han realizado. La comunicación frecuente permite a las personas saber rápidamente a medida que se desarrollan los cambios.

El único requisito previo para un desarrollador que se compromete con la línea principal es que puede construir correctamente su código. Esto, por supuesto, incluye pasar las pruebas de compilación. Al igual que con cualquier ciclo de confirmación, el desarrollador primero actualiza su copia de trabajo para que coincida con la línea principal, resuelve cualquier conflicto con la línea principal y luego construye en su máquina local. Si la compilación pasa, entonces son libres de comprometerse con la línea principal.

Al hacer esto con frecuencia, los desarrolladores descubren rápidamente si hay un conflicto entre dos o más integrantes del equipo. La clave para solucionar problemas rápidamente es encontrarlos rápidamente. Con los desarrolladores comprometiéndose cada poco tiempo, se puede detectar un conflicto a las pocas horas de que ocurra, en ese momento no ha sucedido mucho y es fácil de resolver. Los problemas que permanecen sin ser detectados durante semanas pueden ser muy difíciles de solucionar.

El hecho de que compila cuando actualiza su copia de trabajo significa que detecta conflictos de compilación, así como conflictos de texto. Dado que la compilación se prueba automáticamente, también detecta conflictos en la ejecución del código. Los últimos conflictos son errores particularmente incómodos de encontrar si permanecen durante mucho tiempo sin ser detectados en el código. Dado que solo hay unas pocas horas de cambios entre las confirmaciones, solo hay algunos lugares donde el problema podría estar escondido. Además, dado que no ha cambiado mucho, puede usar la depuración de diferencias para ayudarlo a encontrar el error.

Cada desarrollador debe comprometerse con el repositorio todos los días. En la práctica, a menudo es útil si los desarrolladores se comprometen con más frecuencia que eso. Cuanto más frecuentemente se compromete, menos lugares tendrá que buscar errores de conflicto, y más rápidamente se solucionarán los problemas.

E. Cada commit debe construir la línea principal en una máquina de integración.

Usando confirmaciones diarias, un equipo obtiene compilaciones probadas con frecuencia. Esto debería significar que la línea principal se mantiene en un estado saludable. En la práctica, sin embargo, las cosas todavía salen mal. Una razón es la disciplina, las personas que no realizan una actualización y compilación antes de comprometerse. Otra es la diferencia ambiental entre las máquinas de los desarrolladores.

Como resultado, debe asegurarse de que las compilaciones regulares sucedan en una máquina de integración y solo si esta compilación de integración tiene éxito, se debe considerar que se realiza la confirmación. Dado que el desarrollador que se compromete es responsable de esto, necesita monitorear la construcción de la línea principal para que pueda arreglarla si se rompe.

Hay dos formas principales que he visto para garantizar esto: usando una compilación manual o un servidor de integración continua.

El enfoque de construcción manual es el más simple de describir. Esencialmente es algo similar a la compilación local que hace un desarrollador antes de comprometerse en el repositorio. El desarrollador va a la máquina de integración, comprueba el jefe de la línea principal (que ahora alberga su última confirmación) y comienza la compilación de integración. Él vigila su progreso, y si la construcción tiene éxito, ha terminado con su compromiso.

Un servidor de integración continua actúa como monitor del repositorio. Cada vez que finaliza una confirmación contra el repositorio, el servidor comprueba automáticamente las fuentes en la máquina de integración, inicia una compilación y notifica al confirmador el resultado de la compilación. El confirmador no termina hasta que recibe la notificación, generalmente un correo electrónico.

No todos prefieren usar un servidor de integración continua. Por un lado, es mucho más que simplemente instalar algún software. Pero todas las prácticas aquí deben estar en juego para que la integración sea efectiva.

El objetivo de la integración continua es encontrar problemas lo antes posible ya que una vez que están en el sistema durante tanto tiempo, lleva mucho tiempo encontrarlos y eliminarlos.

F. Arreglar construcciones rotas inmediatamente.

Una parte clave de hacer una compilación continua es que si la compilación de la línea principal falla, debe corregirse de inmediato. El objetivo de trabajar con integración continua es que siempre se está desarrollando sobre una base estable conocida. No es malo que la construcción de la línea principal se rompa, aunque si sucede todo el tiempo sugiere que las personas no están siendo lo suficientemente cuidadosas para actualizar y construir localmente antes de una confirmación. Sin embargo, cuando la construcción de la línea principal se rompe, es importante que se repare rápidamente.

Esto no significa que todos en el equipo tengan que detener lo que están haciendo para arreglar la construcción, por lo general solo se necesita un par de personas para que las cosas funcionen nuevamente. Significa una priorización consciente de un arreglo de compilación como una tarea urgente de alta prioridad.

A menudo, la forma más rápida de arreglar la compilación es revertir la última confirmación de la línea principal, llevando el sistema a la última compilación válida conocida. Ciertamente, el equipo no debe intentar realizar ninguna depuración en una línea principal rota. A menos que la causa de la rotura sea inmediatamente obvia, simplemente revierta la línea principal y depure el problema en una estación de trabajo de desarrollo.

Cuando los equipos están introduciendo integración continua, a menudo esta es una de las cosas más difíciles de resolver. Al principio, un equipo puede tener dificultades para adquirir el hábito habitual de trabajar en las construcciones de la línea principal, especialmente si están trabajando en una base de código existente.

G. Mantenga la compilación rápida.

El objetivo de la integración continua es proporcionar una retroalimentación rápida. Vale la pena hacer un esfuerzo concentrado para que esto suceda, porque cada minuto que se reduce el tiempo de compilación es un minuto ganado para cada desarrollador cada vez que se compromete. Dado que la integración continua exige compromisos frecuentes, esto suma mucho tiempo.

Puede ser desalentador trabajar en un nuevo proyecto y pensar en cómo obtener las cosas rápidas. Al menos para las aplicaciones empresariales, se ha determinado que el cuello de botella habitual es la prueba, particularmente las pruebas que involucran servicios externos como una base de datos.

La idea detrás de una canalización de implementación (también conocida como canalización de compilación o compilación por etapas) es que, de hecho, hay varias compilaciones realizadas en secuencia. El compromiso con la línea principal desencadena la primera compilación, en donde se intenta equilibrar las necesidades de búsqueda de errores y velocidad para que una buena compilación de confirmación sea lo suficientemente estable como para que otras personas puedan trabajar.

Una vez que la compilación de confirmación es buena, otras personas pueden trabajar en el código con confianza. Sin embargo, hay otras pruebas más lentas que puede comenzar a hacer. Las máquinas adicionales pueden ejecutar más rutinas de prueba en la compilación que tardan más en realizarse.

H. Prueba en un clon del entorno de producción

El punto de prueba es eliminar, bajo condiciones controladas, cualquier problema que el sistema tenga en producción. Una parte importante de esto es el entorno dentro del cual se ejecutará el sistema de producción. Si realiza la prueba en un entorno diferente, cada diferencia resulta en un riesgo de que lo que sucede bajo prueba no suceda en la producción.

Bueno, en realidad hay límites. Si está escribiendo software de escritorio, no es factible probar en un clon de cada escritorio posible con todo el software de terceros que ejecutan diferentes personas. De manera similar, algunos

entornos de producción pueden ser prohibitivamente costosos de duplicar (aunque a menudo me encuentro con economías falsas al no duplicar entornos moderadamente costosos). A pesar de estos límites, su objetivo debería ser duplicar el entorno de producción tanto como sea posible y comprender los riesgos que está aceptando para cada diferencia entre prueba y producción.

Si tiene una configuración bastante simple sin muchas comunicaciones incómodas, es posible que pueda ejecutar su compilación de confirmación en un entorno simulado. Sin embargo, a menudo es necesario usar dobles de prueba porque los sistemas responden lenta o intermitentemente. Como resultado, es común tener un entorno muy artificial para las pruebas de confirmación de velocidad, y usar un clon de producción para pruebas secundarias.

I. Haga que sea fácil para cualquier persona obtener el último ejecutable.

Una de las partes más difíciles del desarrollo de software es asegurarse de crear el software adecuado. Hemos descubierto que es muy difícil especificar de antemano lo que desea y ser correcto; A las personas les resulta mucho más fácil ver algo que no está del todo bien y decir cómo debe cambiarse. Los procesos de desarrollo ágil esperan y aprovechan explícitamente esta parte del comportamiento humano.

Para ayudar a que esto funcione, cualquier persona involucrada en un proyecto de software debería poder obtener el último ejecutable y poder ejecutarlo: para demostraciones, pruebas exploratorias o simplemente para ver qué cambió esta semana.

Hacer esto es bastante sencillo: asegúrese de que haya un lugar bien conocido donde las personas puedan encontrar el último ejecutable. Puede ser útil poner varios ejecutables en dicha tienda. Para lo último, debe colocar el último ejecutable para pasar las pruebas de confirmación; dicho ejecutable debe ser bastante estable siempre que el conjunto de confirmación sea razonablemente sólido.

Si está siguiendo un proceso con iteraciones bien definidas, generalmente es aconsejable colocar también el final de las compilaciones de cada iteración. Las demostraciones, en particular, necesitan un software cuyas características sean familiares, por lo que generalmente vale la pena sacrificar lo último por algo que el demostrador sabe cómo operar.

J. Todos pueden ver lo que pasa

La integración continua tiene que ver con la comunicación, por lo que debe asegurarse de que todos puedan ver fácilmente el estado del sistema y los cambios que se le han realizado.

Si está utilizando un proceso de integración continua manual, esta visibilidad sigue siendo esencial. El monitor de la máquina de compilación física puede mostrar el estado de la compilación de la línea principal.

Las páginas web de los servidores de integración continua pueden llevar más información que la manual. A los líderes de equipo les gusta hacer uso de esto para tener una idea de lo que la gente ha estado haciendo y de ver los cambios en el sistema.

Otra ventaja de usar un sitio web es que aquellos que no están ubicados conjuntamente pueden tener una idea del estado del proyecto. También es útil para que los grupos agreguen información de construcción de múltiples proyectos, proporcionando un estado simple y automatizado de diferentes proyectos.

K. Implementación automatizada.

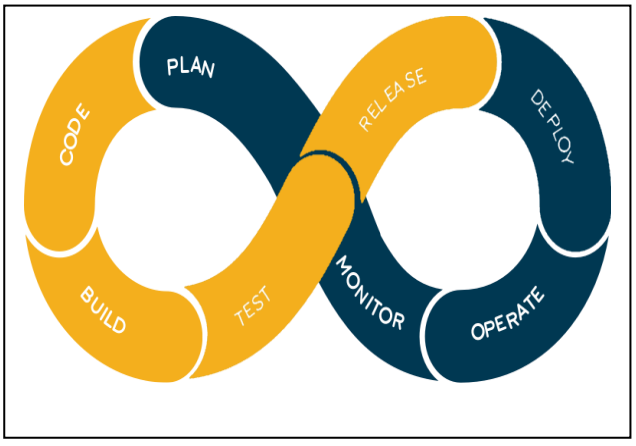
Para realizar la integración continua, necesita múltiples entornos, uno para ejecutar pruebas de confirmación, uno o más para ejecutar pruebas secundarias. Dado que está moviendo ejecutables entre estos entornos varias veces al día, querrá hacerlo automáticamente. Por lo tanto, es importante tener scripts que le permitan implementar la aplicación en cualquier entorno fácilmente.

Una consecuencia natural de esto es que también debe tener scripts que le permitan desplegarse en producción con similar facilidad. Es posible que no esté implementando en producción todos los días, pero la implementación automática ayuda a acelerar el proceso y reducir los errores. También es una opción económica, ya que solo usa las mismas capacidades que usted usa para implementar en entornos de prueba.

Si implementa en producción una capacidad automatizada adicional que debe considerar es la reversión automatizada.

Una variación particularmente interesante de esto que encontramos con la aplicación web pública es la idea de implementar una compilación de prueba en un subconjunto de usuarios. Luego, el equipo ve cómo se usa la compilación de prueba antes de decidir si se implementará en la población de usuarios completa. Esto le permite probar nuevas características e interfaces de usuario antes de comprometerse con una elección final. El despliegue automatizado, vinculado a una buena disciplina de CI, es esencial para que esto funcione.

Fig. 3. En las secciones amarilla se encuentran las fases



de la integración continua.

V. BENEFICIOS DE LA INTEGRACIÓN CONTINUA

- Detectar y arreglar los errores con mayor rapidez: Aunque la integración continua no elimine errores

facilita a que se encuentren y reparen los errores con rapidez debido a la realización periódica de pruebas.

- Reducción de tareas manuales: Al automatizar las actividades, libera a los desarrolladores de tareas manuales, fomentando comportamientos que reducen los bugs y errores y mejorando la producción del equipo
- Entregas más rápidas: Las prácticas de la integración continua permite que las actualizaciones sean entregadas con mayor rapidez y frecuencias.
- Completa visibilidad del proyecto: Se tiene información concreta del avance del proyecto y métricas de la calidad del mismo, permitiendo tomar mejores decisiones al realizar modificaciones.
- Mayor confianza y seguridad del equipo de trabajo: Si se tiene un conjunto de pruebas robusto y todas estas pruebas pasan, se está seguro de obtener en todo momento un software probado y funcional. No se necesita verificar al final que todo se ha realizado de forma correcta.
- Creación de versiones de prueba en cualquier momento: Se puede liberar el software en cualquier momento, ya que se van detectando los errores y solucionando los mismos.
- Mejorar la comunicación: Los desarrolladores pueden ver, comentar y colaborar con el código de otros miembros del equipo.
- Trabajo en paralelo: Permite que los trabajadores trabajen en partes diferentes del código, y que luego estas partes se integren.

Fig. 4 Dibujo de los beneficios de la integración continua

VI. DESAFÍOS PARA ADOPTAR LA PRÁCTICA DE INTEGRACIÓN CONTINUA

Encontramos 2 tipos de desafíos a los que se enfrentan las empresas dedicadas al desarrollo del software, los desafíos que afectan a la implementación de todas las prácticas continuas (integración, entrega e implementación) y aquellos que afectan principalmente a la integración continua.

A. *Desafíos comunes para la adopción de prácticas continuas*

- Desafíos de coordinación y colaboración: una implementación exitosa de prácticas continuas necesita más colaboración y coordinación entre todos los participantes del equipo. Los beneficios reales de las prácticas continuas se obtienen al tener un entendimiento común y responsabilidades colectivas entre los interesados.
- Costo: Para realizar de manera eficiente cada práctica continua se asocia un alto costo que incluye actualizaciones en los recursos, capacitación y entrenamiento de los miembros del equipo.
- Falta de experiencia y habilidad: Las prácticas continuas requieren ciertas habilidades y calificaciones técnicas y blandas (como la comunicación y coordinación) que los miembros de los equipos no las presentan.
- Falta de herramientas y tecnologías adecuadas: Las limitaciones de las herramientas y tecnologías existentes son inhibidores para lograr los objetivos de las prácticas continuas. Muchas herramientas existentes son ineficientes para revisar el código y proporcionar retroalimentaciones de las actividades de prueba de la integración continua.
- Resistencia general al cambio: La implementación de prácticas continuas puede necesitar la adopción de una nueva forma de trabajo para algunos miembros del equipo, y aunque muchos de estos empleados se resisten al cambio, las personas pueden aceptar los cambios siempre que haya razones convincentes para esos cambios.
- Dificultad para cambiar las políticas y culturas organizacionales establecidas: La cultura organizacional es un conjunto de hábitos, comportamientos, actitudes, valores y prácticas de gestión adoptados por una organización. Muchas veces resulta muy complicado cambiar las culturas organizacionales para alinearse con los principios de las prácticas continuas.

B. *Desafíos para la adopción de la integración continua*

- Falta de una estrategia de prueba adecuada: Uno de los principales desafíos al querer adoptar la integración continua se encuentran en los obstáculos de la fase de prueba. La automatización de las pruebas es unas partes más importantes para lograr una implementación de integración continua exitosa, muchas organizaciones dedicadas al desarrollo de software no encontraron forma de automatizar todo tipo de pruebas. Las razones de esto pueden ser, una infraestructura deficiente para automatizar las pruebas, un proceso complejo y laborioso para la automatización de pruebas manuales y las dependencias de software y hardware.

- Mala calidad de la prueba: Otro desafío puede ser la baja calidad de la prueba. Esto incluye tener pruebas pocos confiables (es decir, que las pruebas pueden fallar con frecuencia), un gran número de casos de prueba y pruebas de mucha duración. Estos problemas pueden impedir el proceso de implementación, y reducir la confianza de las organizaciones de desarrollo de software para implementar automáticamente el software de forma continua.
- *Fusión de conflictos:* El último desafío surge durante la integración del código que causan cuellos de botella para practicar la Integración Continua. Estos conflictos pueden darse por la incompatibilidad entre los componentes dependientes y la falta de conocimientos sobre los componentes modificados que puede causar que los equipos realicen un esfuerzo adicional para reescribir sus soluciones. Los conflictos de fusión se atribuyen principalmente a un diseño altamente acoplado.

VII. DIFERENCIAS ENTRE INTEGRACIÓN CONTINUA, IMPLEMENTACIÓN CONTINUA Y ENTREGA CONTINUA

La integración, la implementación y la entrega continuas son tres fases de una "tubería" de lanzamiento de software automatizado. Estas tres fases llevan el software desde la idea hasta la entrega al usuario final. La fase de integración es el primer paso en el proceso. La integración cubre el proceso de múltiples desarrolladores que intentan fusionar sus cambios de código con el repositorio de código maestro de un proyecto.

La entrega continua es la próxima extensión de la integración continua. La fase de entrega es responsable de empaquetar un artefacto para entregarlo a los usuarios finales. La misma ejecuta herramientas de construcción automatizadas para generar este artefacto. En esta fase el artefacto debería estar listo para desplegarse en los usuarios en cualquier momento.

El despliegue continuo es la fase final de la tubería, el artefacto ha pasado con éxito las anteriores de integración y entrega. La fase de implementación es responsable de iniciar y distribuir automáticamente el artefacto de software a los usuarios finales a través de scripts o herramientas que mueven automáticamente el artefacto a servidores públicos u otro mecanismo de distribución, como una tienda de aplicaciones.

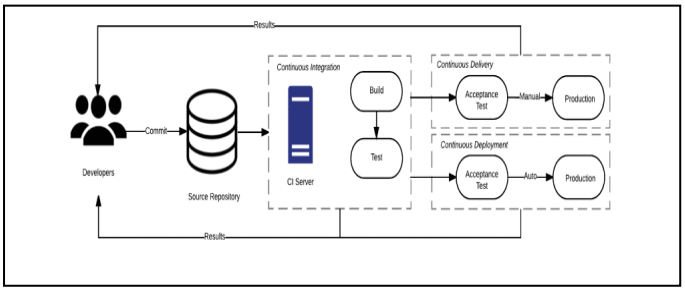


Fig. 5. Relación entre las distintas prácticas continuas

VIII. COMPARACIÓN DE HERRAMIENTAS DE INTEGRACIÓN CONTINUA

A. *Tuberías de Bitbucket*

Es una herramienta de integración continua directamente integrada en Bitbucket, un sistema de control de versiones en la nube ofrecido por Atlassian. Las canalizaciones de Bitbucket se administran como código para que pueda confirmar fácilmente las definiciones de canalización y comenzar las compilaciones. Bitbucket Pipelines, además ofrece CD. Esto significa que los proyectos construidos con Bitbucket Pipelines también se pueden implementar en la infraestructura de producción. Características:

- Fácil instalación y configuración.
- Experiencia unificada de Bitbucket.
- Nube por tercero.

B. *Jenkins*

Es una herramienta de integración continua con una larga trayectoria, es de código abierto e impulsado por actualizaciones de la comunidad. Jenkins es una excelente opción cuando la organización necesita soporte local para manejar clientes confidenciales. Características:

- En la premisa.
- Fuente abierta.
- Robusto ecosistema de complementos.

C. *AWS CodePipeline*

Es uno de los proveedores de infraestructura de nube más dominantes en el mercado. Ofrecen herramientas y servicios para todo tipo de infraestructura y tareas de desarrollo de código. Características:

- Completamente en la nube.
- Integrado con los servicios web de Amazon.
- Soporte de complemento personalizado.
- Robusto control de acceso.

D. *CircleCI*

Es una herramienta de alojamiento en la nube del sistema de control de versiones, es una de las más flexibles, ya que admite una matriz de sistemas de control de versiones, sistemas de contenedores y mecanismos de entrega. Características:

- La mejor integración con el conjunto de productos Atlassian.
- Integración de Github.
- Notificaciones desencadenantes de eventos de integración continua.
- Rendimiento optimizado para compilaciones rápidas.
- Depuración fácil a través de SSH y compilaciones locales.
- Análisis para medir el rendimiento de la compilación.

E. *Azure*

Es la plataforma de infraestructura en la nube de Microsoft, el equivalente de Microsoft a Amazon Web Services. Características:

- Integración de la plataforma Azure.
- Soporte de la plataforma de Windows.
- Soporte de contenedores.
- Integración de Github.

F. *GitLab*

Es una nueva herramienta de integración continua que ofrece una experiencia completa de DevOps, fue creado con la intención de mejorar la experiencia general de Github. Características:

- La mejor integración con el conjunto de productos Atlassian.
- Alojamiento local o en la nube.
- Pruebas continuas de seguridad.
- UX fácil de aprender.

G. *Atlassian Bamboo*

Otra oferta de integración continua de Atlassian. Mientras que Bitbucket Pipelines es puramente una opción alojada en la nube, Bamboo ofrece una alternativa de alojamiento propio. Características:

- La mejor integración con el conjunto de productos Atlassian.
- Un amplio mercado de complementos y complementos.
- Soporte de contenedores con agentes Docker.
- API de activación para la funcionalidad IFTTT.

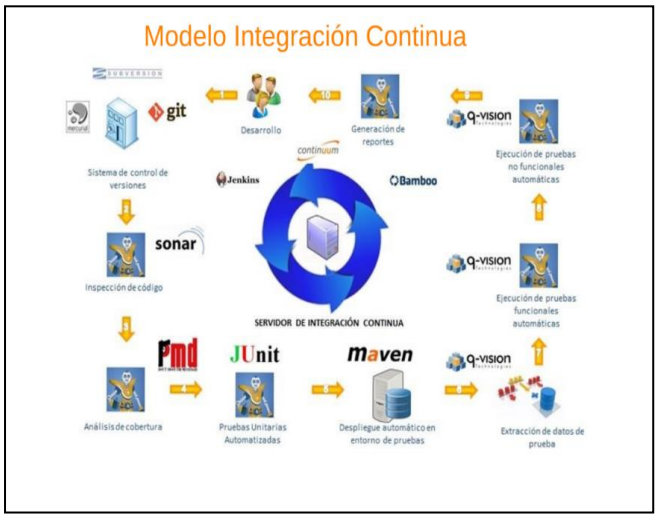


Fig.6. Herramientas en la fases de integración continua

IX. CONCLUSIÓN

Llegamos a la conclusión de que la Integración Continua es una técnica altamente necesaria a la hora del desarrollo de software, debido a que la misma nos brinda diferentes tipos de beneficios tales como la capacidad de mantener desarrolladores de un mismo equipo o diferentes trabajando en paralelo sin conflictos, la cual consideramos una característica clave en la construcción de software en contrapartida a cómo se trabajaba anteriormente con los desarrolladores aislados. Una práctica muy recomendable para mejorar la rapidez y eficiencia de la fusión de las características del producto final con la menor cantidad de conflictos posibles. También podemos decir que la Integración Continua requiere que los equipos que participen en ella están totalmente comprometidos en aplicar la técnica de forma correcta, para de esta forma no dar lugar a situaciones problemáticas.

X. REFERENCIAS

A. *Fuentes*

[1] Fowler Martín, "Integración continua", mayo de 2006, https://www.martinfowler.com/articles/continuousIntegration.html.

[2] Rehkopf Max, "¿Qué es la integración continua?", https://www.atlassian.com/continuous-delivery/continuous-integration

[3] "Práctica del Desarrollo de Software, Integración Continua", https://sites.google.com/site/practicadesarrollosoft/temario/continuous-integration.

[4] Chávez Gisell, "Integración Continua", https://www.smartnodus.cl/integracion-continua/.

[5] Taylor Andrew, "5 Advantages of Continuous Integration", mayo 2017, https://pantheon.io/blog/5-advantages-continuous-integration.

[6] Amazon, "¿Qué es la integración continua?", https://aws.amazon.com/es/devops/continuous-integration/.

[7] Rehkopf, "Continuous Integration Tools", https://www.atlassian.com/continuous-delivery/continuous-integration/tools.

[8] Mojtaba Shahin, Muhammad Ali Babar, Liming Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices", Marzo 2017, https://ieeexplore.ieee.org/abstract/document/7884954.

[9] Ambler Scott, http://www.ambysoft.com/essays/agileTesting.html#ContinuousIntegration.

B. *Links de imágenes:*

[10] Hugo Hendriks, "Ejecuccion de script", diciembre de 2015, https://www.redrock-it.nl/tag/continuous-integration/

[11] Gustavo Delgado, "Integración continua: Lo previo" enero de 2017, https://medium.com/@gustavonecore/integraci%C3%B3n-continua-lo-previo-ff8000516217

[12] Gisell Chavez, "6 Beneficios de la Integración Continua CI", octubre de 2011, https://www.smartnodus.cl/integracion-continua/

[13] Graham Wright, "Integración continua (CI)" , enero de 2017, https://medium.com/@gwright_60924/continuous-integration-ci-e81032bb8502

[14] Mojtaba Shahin, Muhammad Ali Babar, Liming Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices", Marzo 2017, https://ieeexplore.ieee.org/abstract/document/7884954.

[15] Q-Vision Technologies, "Integración Continua y Herramientas de Apoyo", https://www.qvision.com.co/integracion-continua-y-herramientas-de-apoyo/

C. *Modelo de informe técnico:*

[16] IEEE, "Microsoft Word A4", Mayo de 2018, https://www.ieee.org/conferences/publishing/templates.html