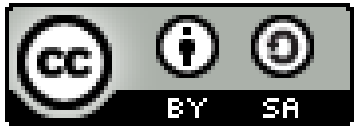
Principios de la Integración Continua

Asignatura: Integración Continua en el Desarrollo Ágil

Máster Universitario en Desarrollo Ágil de Software para la Web

José Ramón Hilera González



Contenido

1. Definición de Integración Continua
2. Entorno de trabajo en Integración Continua
3. Principios de la Integración Continua
4. Beneficios de la Integración Continua
5. Introducir la Integración Continua

Fuente:
https://martinfowler.com/articles/continuousIntegration.html

1. Definición de Integración Continua (original)

- Martin Fowler (2006).
- *Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.*

1. Definición de Integración Continua (traducida)

- La Integración Continua es una práctica de desarrollo de software donde los miembros de un equipo integran su trabajo frecuentemente, normalmente cada persona integra al menos diariamente, obteniéndose multiples integraciones por día.
- Cada integración es verificada por una construcción automatizada del software (incluyendo pruebas) para detectar errores de integración lo antes possible.
- Muchos equipos encuentran que este enfoque consigue reducir significativamente los problemas de integración y permite a un equipo desarrollar software más rápidamente.

2. Entorno de trabajo en Integración Continua

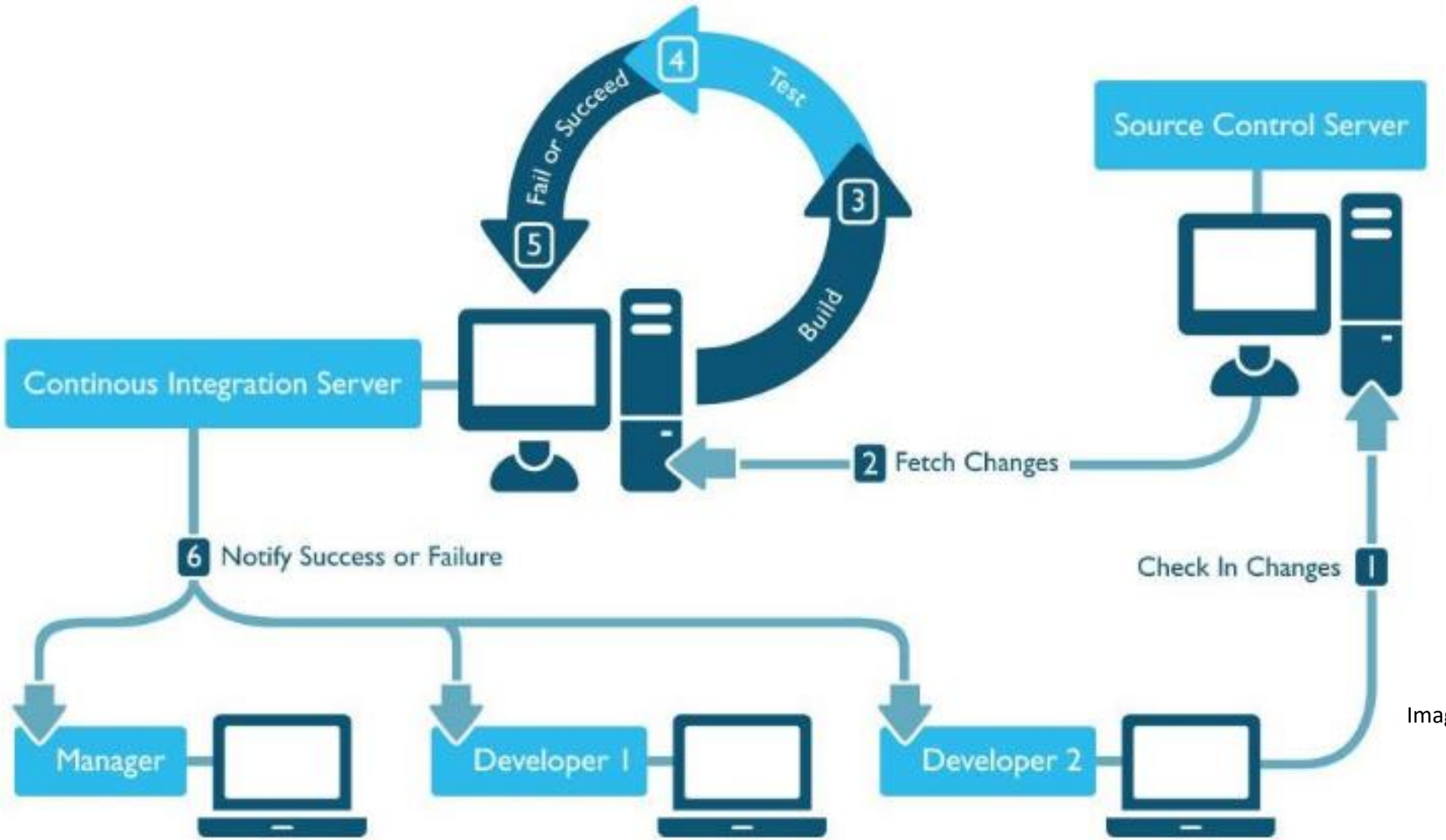


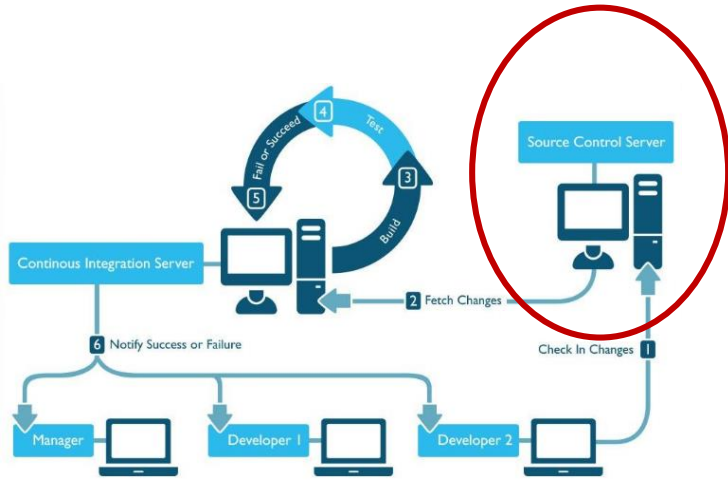
Imagen: insights.sei.cmu.edu

3. Principios de la Integración Continua (Fowler, 2006)

1. Mantener un único repositorio de código fuente compartido
2. Automatizar la construcción (build)
3. Incluir pruebas automatizadas en el proceso de construcción
4. Enviar cada día cambios (hacer commits) al repositorio compartido
5. Utilizar un servidor de integración para volver a construir el software a partir de cada commit
6. Reparar inmediatamente las construcciones fallidas
7. Construir rápidamente
8. Probar en un entorno similar al entorno de producción
9. Todo el equipo debe poder obtener el último ejecutable
10. Todo el equipo puede ver lo que está ocurriendo
11. Automatizar el despliegue

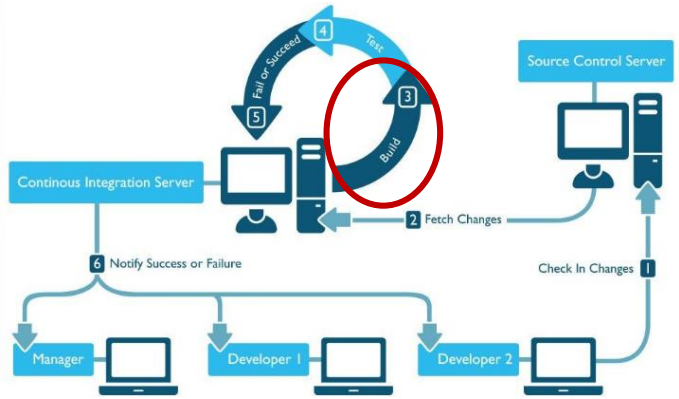
P1. Mantener un único repositorio de código fuente compartido

- Utilizar un repositorio compartido como sistema de gestión del código fuente, que asegure que todo el equipo de desarrollo puede **acceder al código** y localizar cualquier archivo que necesite.
- Todo lo que se necesite para construir (generar) el producto final debe estar en el repositorio: código para pruebas, esquema de base de datos, ficheros de propiedades o configuración, comandos (scripts) de instalación, librerías externas, etc. Pero **no el propio producto final** (jar, war, etc.).
- Cada desarrollador puede estar utilizando en su máquina local un entorno de desarrollo (IDE) diferente al que usan otros desarrolladores (Eclipse, IntelliJ, Visual Studio Code, NetBeans, etc.), pero se puede incluir en el repositorio compartido archivos sobre la configuración de algún entorno de desarrollo (IDE), si se quiere facilitar a los desarrolladores que compartan una misma configuración.
- El repositorio debe estar sometido a un sistema de **control de versiones** que permita crear múltiples **ramas** para manejar diferentes flujos de desarrollo. Se recomienda usar el mínimo de ramas posible. Al menos hay que mantener una rama principal, que suele llamarse rama **master** (o mainline).



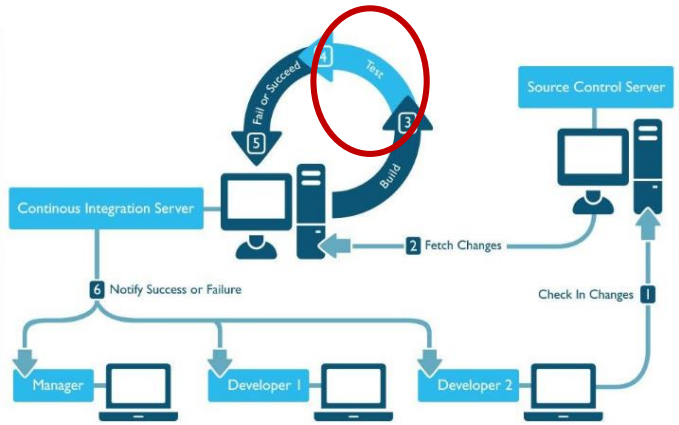
P2. Automatizar la construcción (build)

- La construcción consiste en convertir el código fuente en un producto ejecutable, lo cual puede implicar compilar, descargar librerías, mover archivos, cargar esquemas en bases de datos, etc.
- Hay que hacerlo mediante una secuencia de comandos (script) desde una consola o intérprete de comandos (shell) del sistema operativo.
  - Ej. # mvn package
- No se puede depender de un entorno de desarrollo (IDE), como Eclipse, IntelliJ, NetBeans, VS Code, etc. Cada desarrollador puede trabajar de forma individual con un IDE, pero la construcción a partir del repositorio compartido debe poderse hacer sin ningún entorno de desarrollo, utilizando sólo scripts de comandos.



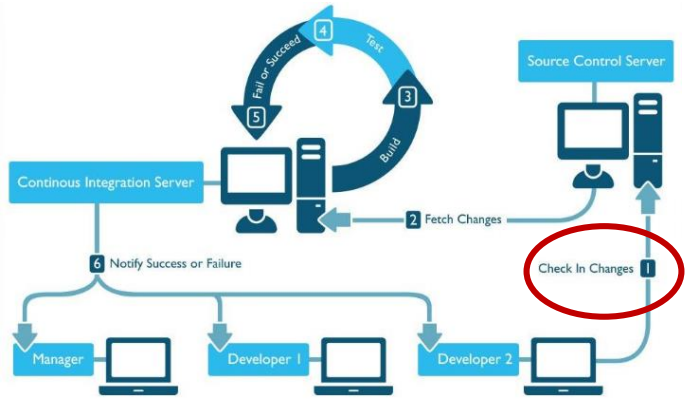
P3. Incluir pruebas automatizadas en el proceso de construcción

- Hay que crear un conjunto (suite) de pruebas (tests) para detectar errores (bugs).
- Las pruebas deben poder ser ejecutadas con un simple comando y el código fuente debe formar parte del código fuente del proyecto (self-testing).
- La suite de pruebas debe fallar si una de las pruebas que incluye falla.
- Las pruebas unitarias (conocidas como Xunit) son un punto de partida para probar partes del producto software, pero hay que complementarlas con otras pruebas que abarquen todo el producto , como las pruebas end-to-end, usando herramientas como Selenium.
- Siempre hay que tener en cuenta que las pruebas demuestran la existencia de errores, pero no demuestran la ausencia de errores.
- Y también que son mejores las pruebas imperfectas que se ejecutan frecuentemente que las pruebas perfectas que no se ejecutan nunca.



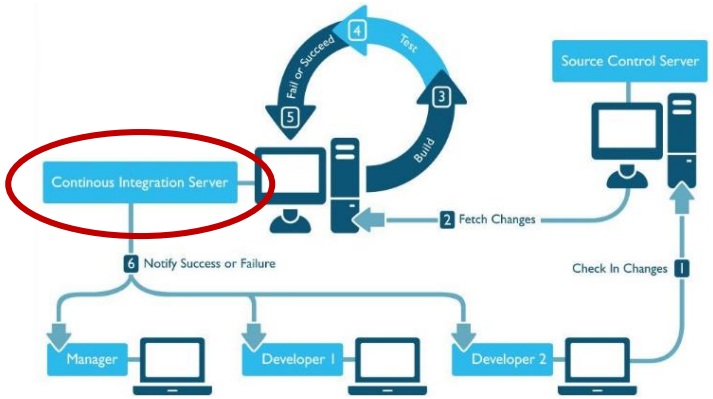
P4. Enviar cada día cambios (hacer commits) al repositorio compartido

- Cada desarrollador debe hacer cambios en un replica del código fuente en su máquina local, compilar y probar en su máquina; y si no hay problemas, entonces puede enviar (subir) los cambios (hacer un commit) al repositorio compartido.
- Haciendo esto frecuentemente, los desarrolladores descubren rápidamente si hay conflictos entre dos desarrolladores. La clave para reparar problemas rápidamente es encontrarlos rápidamente. Si los desarrolladores suben cambios cada mucho tiempo, los futuros conflictos permanecen sin ser detectados durante mucho tiempo, y serán difíciles de resolver.
- Además, los commits frecuentes animan a los desarrolladores a descomponer su trabajo en trozos pequeños, de pocas horas, lo que ayuda a un mejor seguimiento del progreso del desarrollo y proporciona una sensación de progreso.



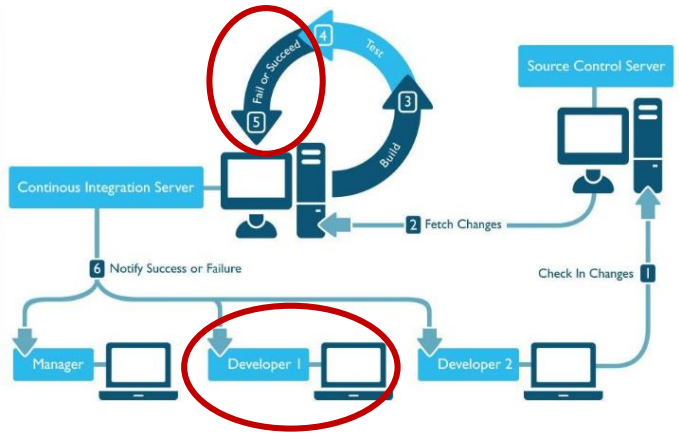
P5. Utilizar un servidor de integración para volver a construir el software a partir de cada commit

- Un servidor de integración actúa como un monitor del repositorio. Cada vez que un desarrollador envía un commit al repositorio, el servidor inicia una nueva construcción automática del producto, y otros trabajos que se hayan definido en el pipeline del proyecto, y al finalizar envía al desarrollador una notificación, normalmente por email, indicando si ha finalizado con éxito o se ha interrumpido por algún error.
- Si una organización lanza construcciones regularmente a unas horas planificadas (por ejemplo, cada noche), no es suficiente para considerarse Integración Continua. El punto clave de la Integración Continua es en encontrar los problemas cuanto antes.



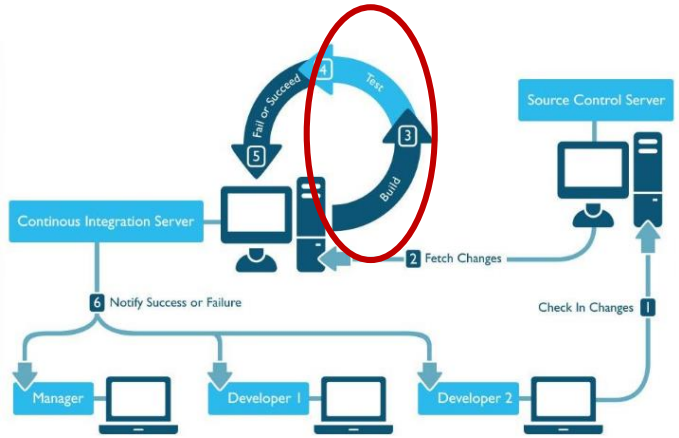
P6. Reparar inmediatamente las construcciones fallidas

- “Nobody has a higher priority task than fixing the build” (Kent Beck).
- A menudo la forma más rápida de reparar una construcción es revertir el último commit enviado al repositorio, volviendo el contenido del repositorio a la situación de la última construcción sin errores.
- El equipo no debe intentar reparar directamente sobre el repositorio, a menos que sea un problema menor, habría que revertir el repositorio, hacer la reparación en una máquina de un desarrollador y volver a enviar un commit.



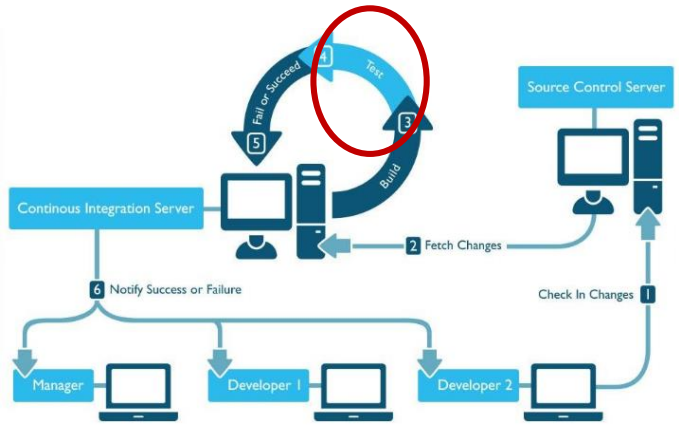
P7. Construir rápidamente

- En la mayoría de proyectos el objetivo de realizar construcciones de hasta diez minutos es razonable. Merece la pena reducir al máximo el tiempo, porque cada minuto que se reduzca la construcción es un minuto ahorrado para cada desarrollador cada vez que hace commits, y la integración continua demanda commits frecuentes, lo cual supone al final un considerable ahorro de tiempo.
- El cuello de botella habitual son las pruebas, especialmente aquellas que necesitan el uso de servicios externos, como una base de datos.
- En estos casos se puede separar la construcción en dos partes, una parte con la compilación y pruebas unitarias, que produce un commit en el repositorio suficientemente estable para que otras personas puedan seguir trabajando. Y una segunda parte para pruebas end-to-end.
- También se pueden lanzar pruebas en paralelo si se dispone de varias máquinas que puedan trabajar simultáneamente.



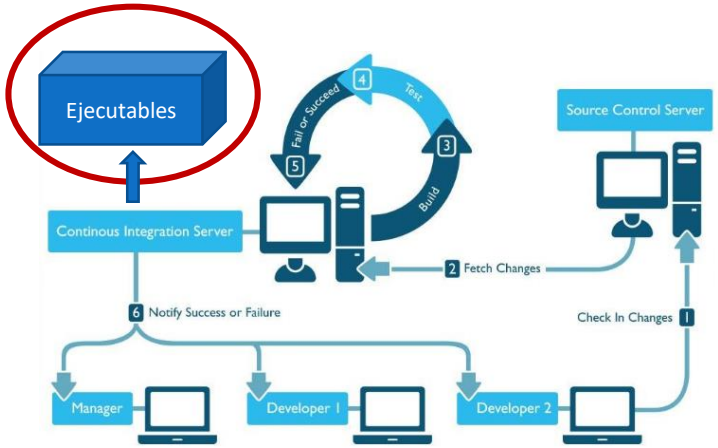
P8. Probar en un entorno similar al entorno de producción

- El objetivo debería ser duplicar exactamente el entorno de producción, y si no se puede, ser conscientes de los riesgos que se están asumiendo por cada diferencia entre el entorno de prueba y el de producción.
- Se trata de utilizar el mismo software de gestión de bases de datos, la misma versión del sistema operativo, las mismas librerías externas, los mismos puertos del servidor, etc.
- Esto se puede conseguir en gran medida mediante el uso de máquinas virtuales y/o contenedores.



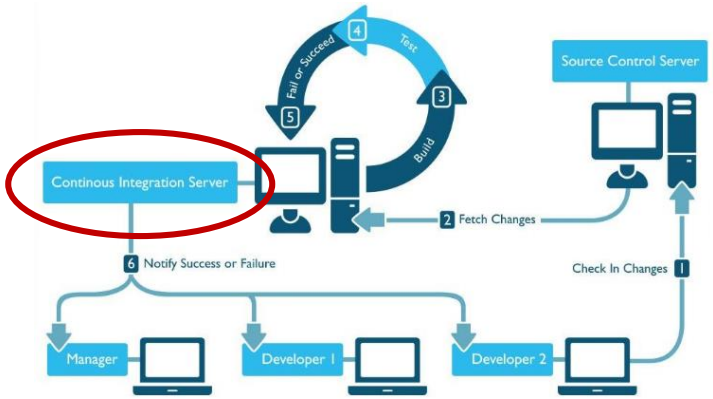
P9. Todo el equipo debe poder acceder al último ejecutable

- Todos los implicados en el proyecto deberían poder conseguir el último ejecutable y ser capaces de ejecutarlo, para demostraciones, para hacer pruebas de funcionamiento, o simplemente para ver lo que ha cambiado desde la semana anterior.
- Es recomendable almacenar los ejecutables en un almacén específico (como un nexus, npm, etc.). No deberían estar en el repositorio del código fuente.
- En desarrollos ágiles (como Scrum), conviene conservar en el almacén los ejecutables obtenidos en las diferentes iteraciones.



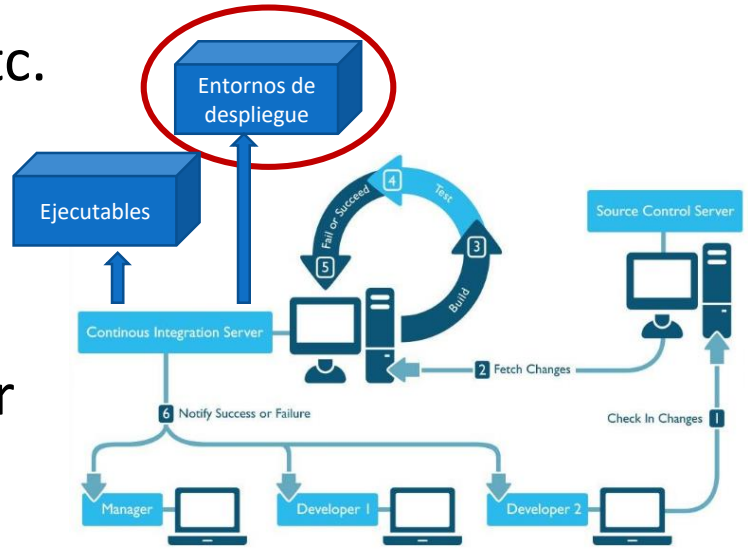
P10. Todo el equipo puede ver lo que está ocurriendo

- La integración continua se basa en la comunicación, hay que asegurar que todos los participantes puedan ver fácilmente el estado del proyecto y los cambios que se han hecho.
- Se recomienda utilizar una página web (normalmente en el servidor de integración continua) que muestre visualmente toda esta información, con indicadores de colores que permitan distinguir cuando un trabajo (compilación, construcción, test, etc.) ha fallado (rojo) o ha tenido éxito (verde). También se debería poder ver el historial de cambios, funcionalidades añadidas, calendario, etc.
- Se trata de que todo el equipo conozca lo que cada miembro ha estado haciendo y cómo ha ido evolucionando el producto que se desarrolla.



P11. Automatizar el despliegue

- Para hacer integración continua se necesitan múltiples entornos, uno para ejecutar pruebas unitarias, otro para pruebas de integración, otro para pruebas de rendimiento, etc. Por tanto, es importante preparar secuencias de comandos (scripts) que permitan desplegar la aplicación en cualquier entorno rápidamente.
- No es necesario desplegar en producción cada día, pero tener preparado el despliegue automático ayuda a acelerar el proceso y a reducir errores.
- Un entorno puede estar basado en clusters de máquinas o nodos, por lo que se deben usar herramientas (ej. Kubernetes) que faciliten el despliegue automático y gradual de lo que corresponda a cada nodo del entorno.



4. Beneficios de la Integración Continua

- Reduce los riesgos
  - El problema con la integración diferida es que es muy difícil predecir cuánto tiempo llevará realizar la integración, y también conocer en qué momento nos encontramos en el proceso de desarrollo, se trabaja a ciegas. Con la integración continua se eliminan situaciones de ceguera (blind spots), porque en cada momento todos los participantes conocen la situación actual, lo que funciona, lo que no funciona, los errores cometidos, etc.
- Reduce los errores, tanto en desarrollo como en producción
  - Esto es cierto siempre que se diseñen buenas pruebas, por lo que es necesario mejorar constantemente el diseño de todas las pruebas.
- Ayuda a eliminar barreras entre clientes y desarrolladores
  - El despliegue frecuente permite a los usuarios obtener nuevas funcionalidades más rápidamente, y pueden dar rápidas opiniones (feeedback) sobre ellas, y generalmente ello redunda en una mayor participación en el desarrollo.

5. Introducir la Integración Continua

1. Uno de los primeros pasos debe ser conseguir automatizar la construcción de la aplicación a partir del contenido del repositorio de código fuente. Para ello hay que conseguir herramientas (como Maven o Gradle), que permitan construir la aplicación completa con un único comando.
2. Hay que introducir el diseño de pruebas automatizadas como parte del proceso de desarrollo. Para ello hay que conseguir herramientas (como JUnit, Selenium, et.) que permitan diseñar diferentes tipos de pruebas.
3. Hay que intentar reducir al máximo el tiempo de construcción de la aplicación, tratando de llegar a un objetivo de diez minutos máximo.