
# Análisis y Diseño de Sistemas 2

2015

Ing. Luis Alberto Arias Solórzano

## Unidad 1

# Continuous Integration

- Integración continua es la práctica de fusionar (merge) una labor de desarrollo con un Master/Trunk/Mainline Branch constantemente para que se puedan probar los cambios. La idea aquí es para probar el código con tanta frecuencia como sea posible para detectar problemas de manera temprana. La mayor parte del trabajo es realizado por pruebas automatizadas, y esta técnica requiere el uso de un framework de pruebas unitarias. Normalmente hay un servidor de compilación (build) para realizar estas pruebas, por lo que los desarrolladores pueden seguir trabajando mientras se realizan estas las pruebas.

- Esta práctica de desarrollo de software es donde los miembros de un equipo integran su trabajo frecuentemente, usualmente cada persona integra diariamente, esto conduce a múltiples integraciones al día dependiendo el número de individuos en el equipo.

# Continuous Integration – Principios

1. **Control de Versiones**

Esta práctica aboga por el uso de un sistema de control de versiones para el control del código fuente del proyecto. Todos los artefactos necesarios para construir (build) el proyecto deben ser colocados en el repositorio. Una copia nueva y no requiere dependencias adicionales.  Es preferible que los cambios se mantenga integrados en lugar de mantener varias versiones del software de forma simultánea. La  mainline (trunk) debe ser el lugar para la versión de trabajo del software (working copy).

2. **Automatizar la construcción del proyecto**

Se debe procurar que un solo comando sea capaz de construir (build) el sistema. Han existido muchas herramientas para estos propósitos, algunas mas actuales utilizadas en ambientes de integración continua. Debemos tomar en cuanta la integración continua dentro de la construcción que muchas veces nos lleva a la entrega del producto a producción. En muchos casos no solamente construye los binarios, sino también genera la documentación, websites,  estadísticas y realiza la distribución (Debian DEB, Red Hat RPM o Windows MSI files).

# Continuous Integration – Principios

3. **Realizar que la construcción se pruebe así misma.**

Una vez el código este construido, todas las pruebas deben correrse para confirmar que se comporta como el desarrollador espera que se comporte.

4. **Todos deben de subir (check in, commit) sus cambios todos los días**

Hacerlo regularmente se puede reducir el numero de conflictos debido a los cambios. Realizarlo en una ventana de tiempo mayor ya sea semanalmente o mas, se corre el riesgo de tener conflictos mayor sin descubrir. Conflictos que suceden de manera temprana en el área de desarrollo inician la comunicación entre los miembros del equipo para compartir sus cambios en el sistema. Hacer “Commit” al menos una vez al día es parte de la definición de la integración continua.

# Continuous Integration – Principios

5. **Cada "Commit" se debe construir**

El sistema se debe construir e integrar con la actual versión de trabajo (working version) y verificar que funcione correctamente. Una practica común es que se utilice la Integración Continua Automática, sin embargo esta puede ser manual. Para muchos, Integración continua es sinónimo de integración automática donde esta es realizada por el servidor de integración o daemons que revisan el control de los cambios y entonces construyen el sistema.

6. **Mantener la construcción rápida**

La construcción debe completarse rápidamente por si sucede algún problema en la integración este pueda ser identificado rápidamente.

# Continuous Integration – Principios

7. **Pruebas en un ambiente idéntico a producción (clone)**

Tener un ambiente de pruebas pueden llevarnos a tener fallas una vez probamos en este ambiente y luego al entregarlo se encuentra que algo es diferente en producción. Sin embargo, tener una replica de producción podría ser de costo elevado haciéndolo poco rentable; en lugar de eso, se puede tener un ambiente de pre-producción que se realiza en una medida escalable derivado de producción para disminuir los costos y cumplir con las pruebas.

8. **Debe ser sencillo conseguir los últimos entregables y/o versiones del producto**

Hacer que el producto este disponible para los stakeholders y testers puede reducir la cantidad de trabajo necesaria para reconstruir el software cada vez que se encuentra algo que no cumple con los requerimientos. Adicionalmente reduce la probabilidad que los defectos sean omitidos en las pruebas. Encontrando los errores antes y en algunos casos hasta reduciendo el esfuerzo necesario para resolverlos.

# Continuous Integration – Principios

9. **Todos deben poder ver los resultados de la ultima construcción al software**

Esto debe ser fácil y sencillo para encontrar donde esto puede llegara romperse y quien fue quien hizo el cambio.

10. **Entrega Automática**

Muchos de los sistemas de integración permiten correr una serie de scripts después de finalizada la construcción. En muchos casos es posible crear un script de entrega y realizar pruebas en un servidor de pruebas donde todos puedan verificarlas, aquí se apoya de la Entrega Continua. Un avance mas allá es la Entrega Continua, cuando automatizamos el pasaje a distintos ambientes e incluso se automatiza los procesos de regresión y rollback.

# Continuous Integration – Practicas

- **Repara las fallas de construcción inmediatamente, nunca dejes una construcción rota.** Si lo haces, puedes causar conflictos y seguramente se desecharan tus cambios.

- **Cada check-in debe ser una mejora al anterior.** Cada check-in debe agregar valor, de otra manera ¿Para qué hacerlo? Una mejora se defina como una nueva pieza de funcionalidad, una reparación a un bug, un incremento en el alcance del código. Debe ir especificado en el log si es posible.

- **Nunca hagas check-in en una construcción rota** a menos que sea para repararla. En primera, si los commits son rápidos y sin sentido, quiere decir que no estas reaccionando de manera correcta y estas dejando que otro se encargue de la integración. Segundo, debido a las fallas iniciales muchas veces es mas difícil detectar nuevas fallas o en que punto se arruino el código si continúan sobrescribiendo con un build roto o incorrecto.

# Continuous Integration – Practicas

- **El codigo debe  se construido y probado antes de ir al control de versiones**. Esto se puede hacer al compilarlo localmente haciendo las pruebas pertinentes  o invertir en un sistema de integración continua que pueda hacer esto (Pulse, TeamCity, ElectricCommander). En todo caso realizar un Pre-Commit que permita el acceso del código.

- **Todos deben prestar interés a la salida del sistema de integración**. Esto incluye a project managers y testers. Trabajar con el principio de que cada commit agrega funcionalidad nos lleva a que debemos mostrar esos resultados, probarlos y que todos en el equipo estén enterados. Además debemos reaccionar rápidamente si por alguna razón este ultimo check-in no contenía una mejora al software.

- **Pruebas automáticas funcionales deben ser parte del proceso de integración.**  Si cuentas con una aplicación para realizar pruebas de regresión, muchas veces debido a su alcance es bueno integrarlo en corridas mas largas para abarcar mas código por corrida. Debe existir un balance.

# Continuous Integration – Practicas

- **Se debe procurar hacer builds ligeros y optimos.** Todos detestamos esperar demasiado para simplemente ver si compilo o si alguna dependencia esta rota. Debemos hacer que estos de manera ligera dependiendo que estamos modificando, es necesario revisar si todas nuestras herramientas de revision deben correr para cada linea de codigo que escribimos cada vez que guardamos. Las probabilidades de encontrr errores son altas pero en muchas ocasiones es mejor dejar el trabajo de las construcciones pesadsa y tardadas a procesos paralelos que no afecten nuestro ritmo de trabajo.

- **Reaccionar ante el feedback.** Los sistemas de integración continua nos llevan a obtener mejoras e incremento en la calidad como ninguna otra, por lo tanto no debemos ignorarlas. Sin embargo, es fácil de ignorar cuando las alertas no son importantes (warning). La solución es analizar estas alertas con el equipo de trabajo y determinar el impacto en el sistema.

# Continuous Integration – Hechos

- La integración continua se vale muchas veces del uso de sistemas control de cambios
- Asume un alto grado de ejecución de pruebas en el código previo a cometer cambios en el repositorio de versiones

*El trabajo de un desarrollador no finaliza al momento de hacer cambios al código, probarlos y cometerlos, sino hasta el momento en el que el proyecto completo esta integrado y funciona correctamente aún con los cambios que el desarrollador halla agregado al proyecto.*

# CI – Componentes

- **Los entornos de integración continua construyen el software desde el repositorio de fuentes y lo despliegan en un entorno de integración sobre el que realizar pruebas unitarias o de aceptación.**

- Implantar procesos de este tipo conlleva una inversión en tiempo que será recuperada conforme avance el proyecto. No obstante, esta inversión es cada vez más reducida gracias a la disponibilidad de herramientas Open Source que nos ofrecen soluciones de Integración Continua cada vez más sencillas de implantar.

# CI – Características

- La integración continua es también un buen ejemplo de una regla básica de un programador pragmático: "Intentar automatizar todo el trabajo repetitivo que realiza".

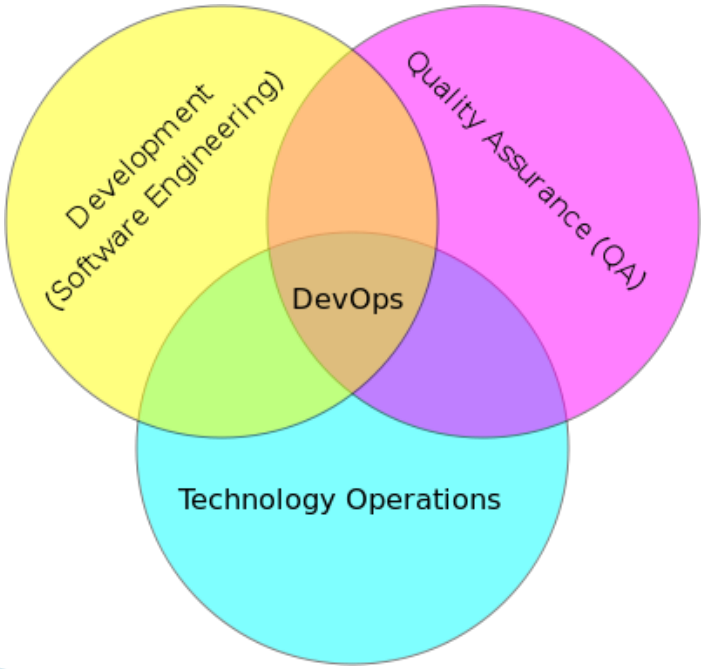
- Integración continua consiste en disponer de un proceso automatizado que permita la construcción de nuestro software desde las fuentes, que despliegue nuestro software en un entorno similar al entorno final y que lleve a cabo el conjunto de pruebas que validan su correcto funcionamiento.

# CI – Beneficios

- Minimiza las sesiones de búsqueda de fallos a la hora de integrar el código. Fallos realmente complicados de encontrar dado que suelen ser efectos colaterales de código que ha sido desarrollado de manera independiente.
- Permite identificar fallos en el entorno de producción en etapas tempranas. Esto permite ser eficiente en los pasos a producción y evitar los periodos de integración finales en los entornos de producción.
- Minimización del tiempo de realimentación con el cliente, al minimizar el paso de desarrollo a un entorno de integración.
- Eficiencia del equipo de desarrollo: no es necesario todo el conjunto de pruebas en el entorno local, minimiza el tiempo de paso a producción.
- Aumenta la confianza en el código subido al control de versiones.

# DEVOPS

- Es una metodología de desarrollo de software que se centra en la comunicación, colaboración e integración entre desarrolladores de software y los profesionales de operaciones en las tecnologías de la información (IT). DevOps es una respuesta a la interdependencia del desarrollo de software y las operaciones IT. Su objetivo es ayudar a una organización a producir productos y servicios software rápidamente.



# DEVOPS – Características

- La meta de DevOps es maximizar la predictibilidad, eficiencia, seguridad y mantenimiento de las operaciones. Esto muchas veces es soportado por procesos automáticos.

- DevOps se integra con la entrega del producto, pruebas de calidad, desarrollo y tareas de mantenimiento, para incrementar la confiabilidad en el sistema en cada uno de los ciclos de desarrollo. Muchas de las ideas y practicas empleadas en Devops vienen la administración de sistemas empresariales y practicas de desarrollo ágil.

- DevOps ayuda en el desarrollo de software a estandarizar los ambientes y procesos de entrega. Los eventos pueden ser mas fácilmente trazados y auditados. Los desarrolladores reciben mas control de los ambientes por medio de accesos especiales y controlados.

- Las compañías que cuentan con problemas de entrega automática, usualmente ya cuentan con algún proceso automático pero necesitan mas flexibilidad en ese proceso sin necesidad de ejecutar complicadas líneas de comando.

# Tarea – Investigar

Conceptos:

- Amazon Machine Image
- CFEngine
- Chef (software)
- Puppet (software)
- Salt (software)
- Ansible (software)
- GigaSpaces
- Cloudify
- Plutora

# Bibliografia

**Continuous Integration**

Martin Fowler

article

http://www.martinfowler.com/articles/continuousIntegration.html

# Gracias