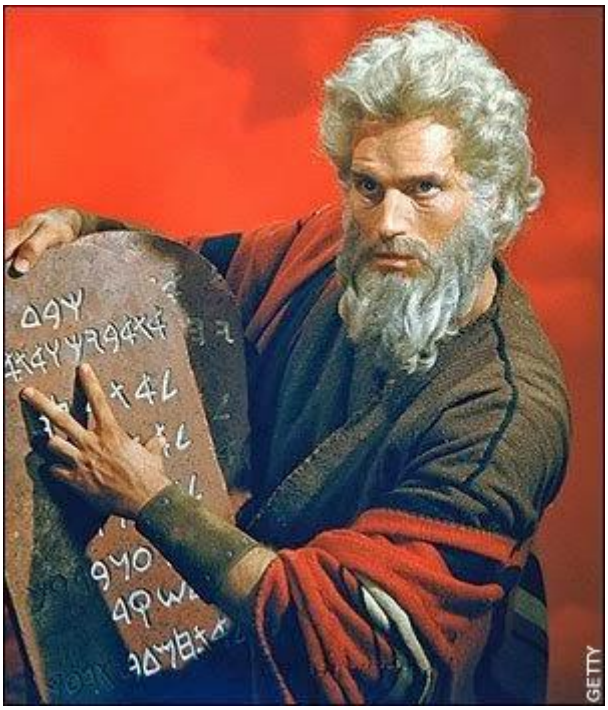
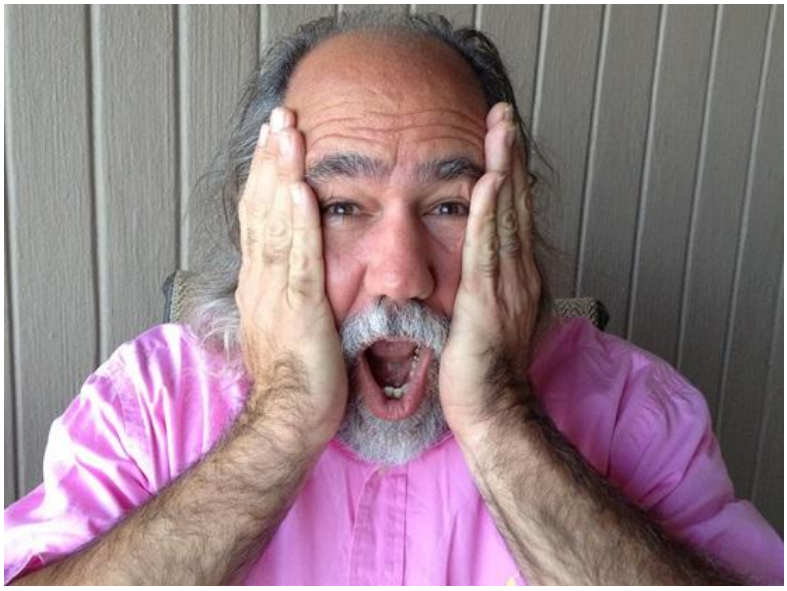
La Integración continua: Netscape, la NASA, XP y dos mayordomos

En el artículo "La Integración Continua reduce el riesgo en proyectos software" comentábamos un poco el contenido del artículo de Martin Fowler que sacó en el 2005 sobre este mismo tema. En este artículo Fowler explicaba que la CI tenía un beneficio directo en los proyectos software reduciendo el riesgo, y tuvo el efecto añadido de ayudar a que se difundiera con aún más fuerza. Haciendo un guiño nostálgico a un anuncio del 2001 que decía "Si no sabes programar no sabes informática", hoy en día "si no practicas la integración continua, no desarrollas software". Volviéndome a ponerme nostálgico, ya lo decían en un anuncio de los 80: "la informática es mucho más que unas palabras". Porque desarrollar software es mucho más que sentarse durante horas a escribir líneas de código.

Los que conozcan Extreme programming me dirán, "integración continua es una de las prácticas de XP". Sí que lo es, es una de las 12 prácticas de XP, pero ¿cómo fue su evolución? y ¿a quién se le ocurrió por primera vez que debían hacerse integraciones automáticas lo más a menudo posible para reducir el riesgo en los proyectos? ¿Un día se levantó Kent Beck de la cama y dijo: "¡¡¡Eureka!!!"? y uno de los doce mandamientos será: integración continua. Nada más lejos de la realidad.



La CI fue propuesta y nombrada por primera vez en 1991 por Grady Booch en su método (Método Booch). En este caso Booch no hablaba en ningún caso de integrar varias veces al día. Grady Booch es conocido por desarrollar UML junto con James Rumbaugh. En 1991/94 desarrolló el método Booch. Se trataba de una técnica usada en ingeniería software para el diseño de objetos (predecesor de UML y RUP). Este método hablaba de uso de objetos, métricas, QA, patrones de diseño, formalismo, madurez de procesos y una notación robusta. Habla de sacar releases de arquitectura hasta llegar al sistema final (evolutivos)…si, es el precursor de RUP también, evidentemente.



En los años 90 del siglo XX se producían dos temas importantes para el surgimiento de la integración continua. Uno fue que contrataron a Kent Beck (uno de los creadores de XP)

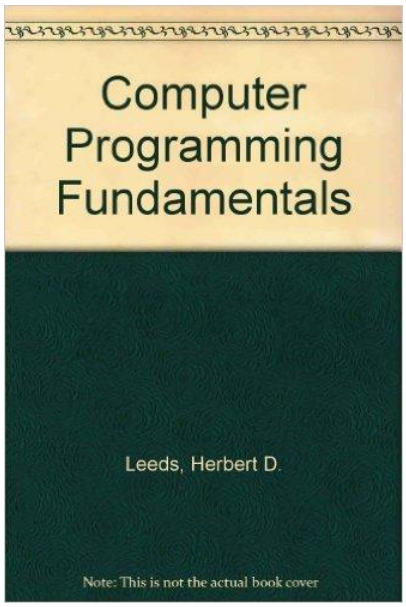
en Chrysler con el objetivo de llevar un proyecto (El C3) de gestión de nominas, en el cual los requisitos eran un gran problema (¡¡¡no me lo esperaba!!!…). Parece ser que el sistema legacy de nóminas era bastante infumable (más de lo habitual supongo), se trataban de varias aplicaciones, cada uno de su padre y de su madre, que hacían lo mismo en algunos casos pero de forma diferente y donde los requisitos, como no, brillaban por su ausencia. El primer año que sacaron la primera release fue muy duro. Tan duro, que cliente, punto de contacto de Chrysler con el proyecto, tuvo que pedir la baja por estrés.

Por otro lado, mientras se cocía la historia de Chrysler y XP, ocurre una historia paralela. Los ingenieros de calidad de Netscape, tuvieron una gran ocurrencia. Estos llegaron a la conclusión de que sería bueno poder realizar construcciones del código automáticamente de forma periódica. Tenían bastantes problemas con CVS (que acaba de surgir en los 90), ya que cada vez que iban a sacar una release, el código no compilaba correctamente: “en mi máquina funciona” decían los desarrolladores.

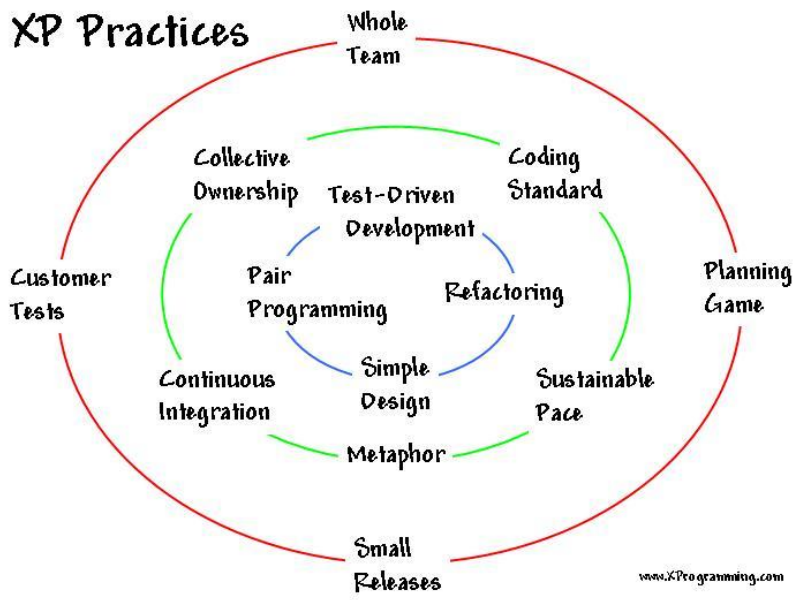
Se pusieron manos a la obra y crearon una herramienta muy sencilla llamada Tinderbox que les permitía realizar una construcción diaria de su software de forma que sabían en el mismo día si había habido un problema con la construcción del mismo (la build). En poco menos de un año, se dieron cuenta que bajando la cadencia de los builds, eran capaces de saber cuál había sido el problema de integración que había roto la build (a menos tiempo, menos commits realizados y más fácil detectar el problema de fallo). Progresivamente bajaron la cadencia hasta realizar builds automáticos cada hora. Cuando en 1998 el código de Netscape es liberado, también es difundida su forma de trabajo a la comunidad de open-source. De esta manera la construcción frecuente de tu código de forma automática pronto se convirtió en una buena práctica en el mundillo del desarrollo software.

XP lleva las mejores prácticas a niveles extremos. Por ejemplo la práctica de hacer test antes de desarrollar fue usada por la NASA en el programa Mercury Space por primera vez para tarjetas perforadas en 1960. XP la llevo al extremo con TDD. Aunque Kent beck no se considera a sí mismo el creador de Test Driven Development, fue el que implemento

SUnit, un framework de tests unitarios para Smalltalk que fueron las bases de la familia xUnit (conjunto de frameworks basados en sus premisas como nUnit, phpUnit, junit, MsUnit, etc). Este framework se pensó para realizar TDD basándose en ideas de Gerard Weinberg y Leeds, Herbert D. en su libro (Computer Programming Fundamentals – 1960). Sí, he de afirmar que es curioso que en la portada original Weingberg no aparezca. Pero eso es otra historia.



Otra de las prácticas que se lleva al extremo es la integración continua. Kent Beck tomó la idea de Booch, las nuevas buenas prácticas de los build automáticos con las ideas de los servidores de builds automáticos emergentes y la inclusión del framework de test automáticos que había creado para dar forma a una de las prácticas de XP: Si aquellos tipos de Netscape podrían hacer construcciones del código cada hora, ¿por qué no íbamos a poder hacerlos en cada commit y además pasar automáticamente baterías de test automáticamente con sUnit? En 1999 Kent beck publica el primer libro sobre XP donde incluye el concepto como una de sus 12 prácticas.

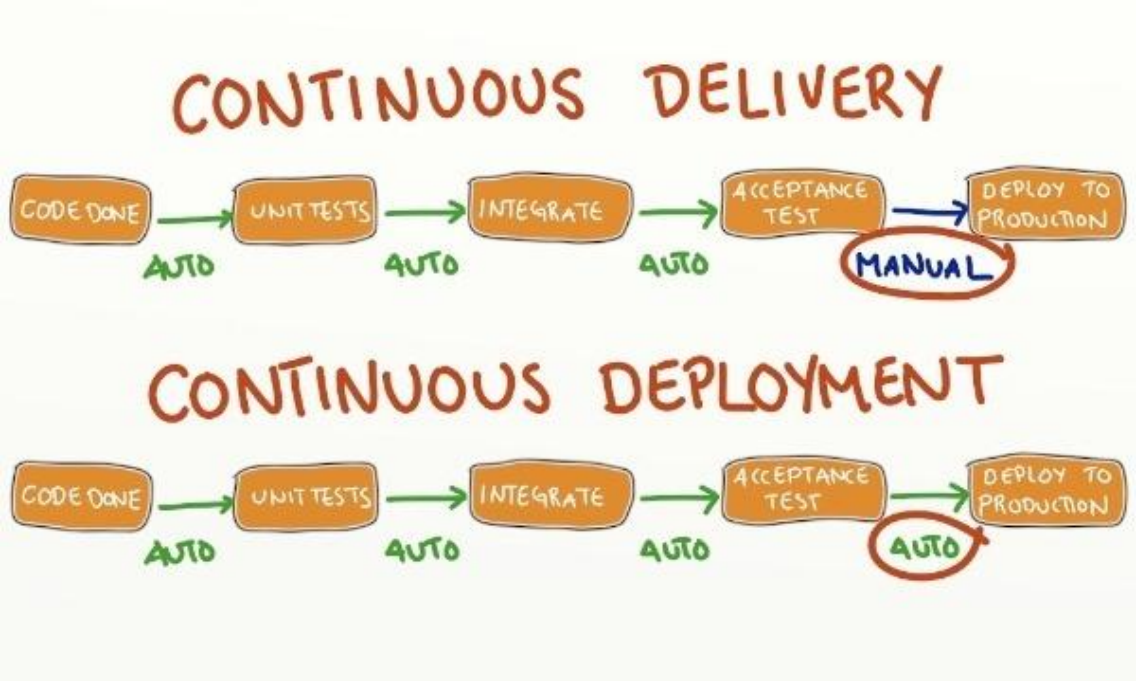


La integración continua ha ido evolucionando a medida de que han ido apareciendo nuevas tecnologías y herramientas. Si tenemos que pensar en cual fue la primera herramienta construida para este propósito, podemos pensar que fue Cruise Control que surgió en el 2001 como servidor de builds extensible. Esta herramienta estaba pensada para realizar builds automáticos usando Ant o Maven. Funcionaba (y funciona) a través de plugins que extendían su funcionalidad. Se trata de una herramienta gratuita y open-source. Existe una herramienta para .NET similar denominada Cruisecontrol.NET que aparece en el 2003 con su versión 0.3.1 pero no es hasta 2005 que aparece la primera versión 1.0 estable. Es decir, que la integración continua basada en herramientas específicas (no el concepto como tal), tampoco es tan lejano. Fue cuando se estrenó "Batman begins".

Hudson es otra herramienta de integración continua escrita en Java que se ejecuta en un Apache Tomcat o en el servidor de aplicaciones GlassFish. Trabaja con CVS, Svn, Git y clearcase, y puede ejecutar Apache Ant y Maven así como procesos Windows. Hudson surgió en 2008 y se convirtió en una alternativa a Cruise control y otros servidores de builds de código abierto. El desarrollador principal de Hudson fue Kohsuke Kawaguchi que actualmente es empleado de Sun Microsystems. Si dejas a un montón de desarrolladores software el código de una herramienta como esta, pasa lo que pasa: crean una comunidad y la llevan “hasta el infinito y más allá”.

Cuando más adelante Oracle compró Sun, este declaró que quería registrar el nombre de Hudson y desarrollar una versión comercial. Esto enfadó mucho a la comunidad de desarrollo de Hudson, la cual decidió, junto con Kawaguchi hacer un fork del código de Hudson y llamarlo Jenkins en 2011 (ambos son nombres típicos de mayordomos). Al final Oracle se da por vencido y en 2012 dona Hudson a la fundación Eclipse. En 2013 llevan muchos commits de Jenkins a Hudson. Lo que decía yo… “hasta el infinito y más allá”.

El caso es que los conceptos han cambiado poco desde final del siglo XX, ya que siguen siendo prácticamente los mismos. Si han aparecido nuevos, es porque han aparecido herramientas que ahora lo permiten y anteriormente no, o no con la misma facilidad. De la integración continúa se pasa a la entrega continua y de ahí al despliegue automático en producción. Hoy en día existen muchas herramientas y plugins desarrollados que están trayendo nuevos conceptos y capacidades de automatización, o quizá ya no tan nuevos en el momento de publicación de este post. Algunas organizaciones, hoy en día, son lo suficientemente maduras para automatizar tanto el desarrollo software y las pruebas, que con cada commit despliegan directamente en los servidores de producción con seguridad (pocas son) (continuous deployment). Pero eso amigos, eso, es otra historia.



danielVillahermosa.wordpress.com

Publicado: 17/01/2016

Artículo extendido

REFERENCIAS:

[1] Continuous Integration – Martin Fowler

[2] Continuous delivery – Martin Fowler

[3] Extreme Programing Explanined:Embrace change – Kent Beck