La Integración Continua Aplicada en el Desarrollo de Software en el Ámbito Científico – Técnico

Alicia Salamon, Patricio Maller, Alejandra Boggio, Natalia Mira, Sofia Perez, Francisco Coenda.

Departamento de Informática, Instituto Universitario Aeronáutico.
Av. Fuerza Aérea 6500, Córdoba, Argentina.
{as.salamon, pmaller, alejandra.boggio, ncmira, sofiabeatrizperez, franciscocoenda}@gmail.com

Abstract. El proceso de integración de componentes que se requiere en los proyectos no es una tarea simple. La integración de software es un problema complejo sobre todo en sistemas que involucran código desarrollado por diferentes personas, por esta razón es necesario contar con un entorno que garantice la adecuada integración de las partes de un proyecto y posibilite visualizar los resultados de la integración de una manera fácil y clara. En este marco la Integración Continua ofrece un esquema que permite realizar integraciones a medida que se lleva a cabo el desarrollo generando incrementos pequeños y mostrando los resultados obtenidos. En este sentido el presente trabajo plantea un modelo de referencia cuya finalidad es construir una solución open source que implementa la Integración Continua, y permite evaluar los beneficios que aporta al proceso de desarrollo de software científico – técnico.

Keywords: Automatización, Integración Continua, Control de Versión, Cruise Control, Git.

1 Introducción

La automatización de actividades que se realizan en la industria es básica para mejorar la producción, la calidad y garantizar el correcto cumplimiento de las reglas de negocio. En los equipos de trabajo de desarrollo de software se busca reducir los tiempos de las actividades que se realizan a través de la automatización [1, 2]. Esto ha llevado a que en los últimos años hayan surgido nuevas prácticas y herramientas que tienden a satisfacer esta necesidad en el marco de la mejora continua del proceso de desarrollo de software. En este sentido, la integración continua [3], de la cual devienen los motores de integración, es uno de los temas que está ocupando un lugar cada vez más importante en la construcción de software.

La integración continua es una práctica que comienza con la organización de los proyectos en una estructura de directorios adecuada para establecer el orden de ejecución de los componentes de un proyecto (incluyendo casos de prueba), y de esta manera facilitar la construcción correcta del software cuando se ejecuta el proceso de integración, logrando que el mismo sea transparente para el equipo de desarrollo.

Además permite a los desarrolladores de software visualizar el estado de las construcciones que se generan en el proceso de integración identificando posibles incidencias, y de esta manera evitando que las mismas generen errores futuros en el proyecto y se dificulte su resolución.

Para el desarrollo de esta investigación se ha observado que en el ámbito científico-técnico se desconoce a la ingeniería de software como disciplina, lo que conlleva a que el desarrollo de software científico-técnico carezca del soporte o fundamento necesario para implementar las prácticas más básicas de la ingeniería de software. Al mismo tiempo, los científicos y técnicos argumentan que utilizan al software como una herramienta cuyo fin es permitirles evaluar los resultados de sus investigaciones [4, 5].

Ante lo expuesto anteriormente, este trabajo propone una arquitectura de referencia desarrollada con componentes open source [6] que permiten automatizar el proceso de desarrollo a través de la integración continua y el control de versiones. Esta investigación se ha llevado a cabo en el marco del proyecto PIDDEF 42/11, titulado Metodología y Framework de Gestión de Líneas Base de Integración de Aplicabilidad en el Desarrollo de Software para el Proyecto UAV.

2 Investigación

2.1 Integración Continua

La integración continua es una práctica en la cual los miembros de un grupo de desarrollo integran (compilación y ejecución) los distintos componentes de un proyecto con una frecuencia especificada. Cada integración se realiza de forma automática (incluyendo sus casos de prueba) con el fin de detectar los errores de integración lo antes posible. Según Martin Fowler, muchos equipos de desarrollo han encontrado que este enfoque reduce significativamente los problemas de integración y permite que los equipos desarrollen software cohesivo más rápido [3].

Un escenario típico de Integración Continua [7] se compone de la siguiente manera:

- Primero, un desarrollador realiza un commit de su trabajo al repositorio de control de versión a la vez que el servidor de Integración Continua verifica cambios en el repositorio, por ejemplo cada 5 minutos.
- El servidor de integración continua detecta los cambios en el repositorio de control de versión, extrayendo el último commit que se ha realizado y ejecutando una build script que se encarga de integrar los distintos componentes del software en desarrollo.
- El servidor de integración continua genera feedback con los resultados del proceso de building, el cual es enviado a los miembros que se especifique del proyecto.
- El servidor de integración continúa revisando cambios en el repositorio de control de versiones.

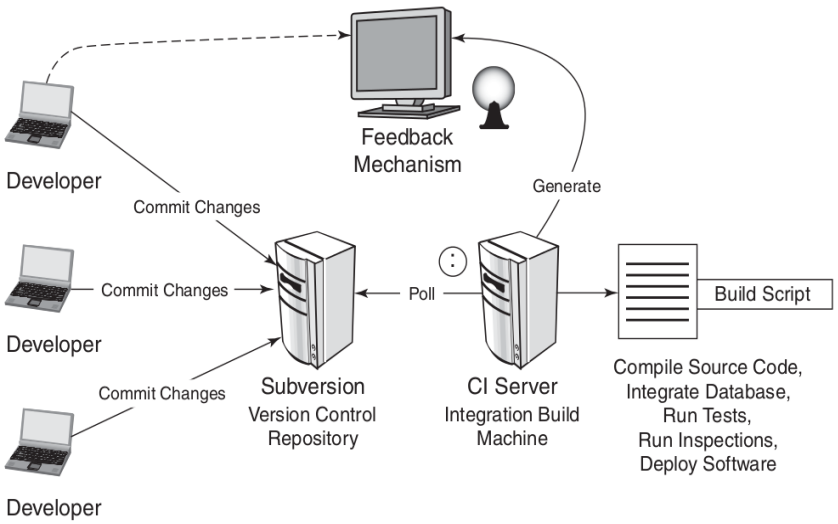


Fig. 1. Escenario típico de Integración Continua.

Como se puede apreciar en la descripción del escenario anterior, no solamente se automatiza el proceso de building de software, sino también requiere un sistema de control de versión. Contar con un sistema de control de versión permite registrar los cambios que se realizan en un proyecto a lo largo del tiempo, posibilitando recuperar versiones específicas más adelante [8].

2.2 Software Científico

El desarrollo de software científico es muy diferente al desarrollo de software comercial, ya que poseen características particulares, hay dos culturas bien definidas [9]. Una de ellas es el dominio en el que se desenvuelven, tomando en cuenta que el ámbito científico es un ambiente muy específico, ya que sólo los científicos poseen un conocimiento acabado acerca de la problemática a solucionar.

Una de las referentes del estudio de la problemática es Segal, quien observó la existencia de dos culturas bien marcadas. La primera observación que menciona es que poseen prácticas de desarrollo distintas dificultando la formación de equipos multidisciplinarios, ya que hay dos perfiles de profesionales. El científico sabe del dominio y posee conocimientos de la física y la ingeniería, mientras que el ingeniero de software sabe hacer software de calidad, flexible, reutilizable. Una segunda diferencia es referida a la actividad de la definición de los requerimientos, la ingeniería de software plantea adoptar un proceso de desarrollo, promueve una definición clara y concisa de los requerimientos de ser posible en documentos formales desde el comienzo del proyecto. Mientras que en el ámbito científico no ocurre así, ya que en varias ocasiones no se sabe exactamente cuáles son los requerimientos que debe satisfacer el software, los mismos se van definiendo a

medida que avanza el proyecto y no necesariamente se documentan formalmente [10]. Como resultado de estas diferencias muchos de los proyectos científicos están conformados por científicos ya que es más fácil que ellos aprendan de software que viceversa.

Los problemas antes expuestos aparecen ya que los desarrolladores científicos rara vez tienen formación acerca de la ingeniería de software [4,5]. Chalmers expresó que los científicos no le dan suficiente valor al software sino más bien a la ciencia y segundo, el software sólo es un instrumento [11].

2.3 Open Source

Esta iniciativa define al software open source como un software que puede ser usado libremente, modificado y compartido por cualquier persona (no solamente desarrolladores de software). El software open source es desarrollado por varias personas (incluso se pueden encontrar en distintas partes del mundo) y distribuido bajo diversas licencias [12] que debe cumplir con la definición que establece open source.

La iniciativa open source fue fundada por Eryc Raymon y Brunce Perens en Febrero de 1998, siendo estas dos personas referentes ampliamente reconocidas en la comunidad del open source [6].

3. Modelo de referencia

En este trabajo se ha llevado a cabo una investigación y diseño de un modelo de referencia de integración continua.

Se realiza una selección de herramientas en función de ciertos parámetros, entre ellas, el soporte que poseen por parte de la comunidad open source, la documentación disponible, la flexibilidad ofrecida para trabajar con diversos lenguajes de programación, las prestaciones que ofrecen y la capacidad para adaptarse al entorno de prueba del modelo de referencia.

Las herramientas que se utilizaron en esta arquitectura de referencia son las siguientes:

1. Cruise Control como servidor de integración continua.
2. Git como software de control de versiones. Sumado a este, Gitolite, el mismo es una herramienta que permite el manejo de usuarios en Git.
3. Ant como herramienta de construcción de software (compilación).

A continuación se brinda una breve descripción de cada una de ellas.

3.1 Cruise Control

Es un framework de integración continua basado en java que permite crear un proceso de building personalizado. Incluye una diversidad de plugins que permiten adaptar la configuración (a través de un archivo xml llamado config.xml) del framework de acuerdo a las necesidades del equipo de desarrollo, incluye además una

interfaz web que provee detalle sobre las builds actuales y anteriores [13]. Es una herramienta open source que se distribuye bajo la licencia BSD. [12]

3.2 Git

Es un sistema de control de versiones distribuido diseñado para trabajar con proyectos de diferentes tamaños, haciendo foco en la eficiencia (velocidad de trabajo). Cada directorio de trabajo de Git es un repositorio en sí mismo con un historial completo de las modificaciones realizadas y las capacidades de tracking completas que no depende del acceso a una red o servidor central [8].

3.3 Gitolite

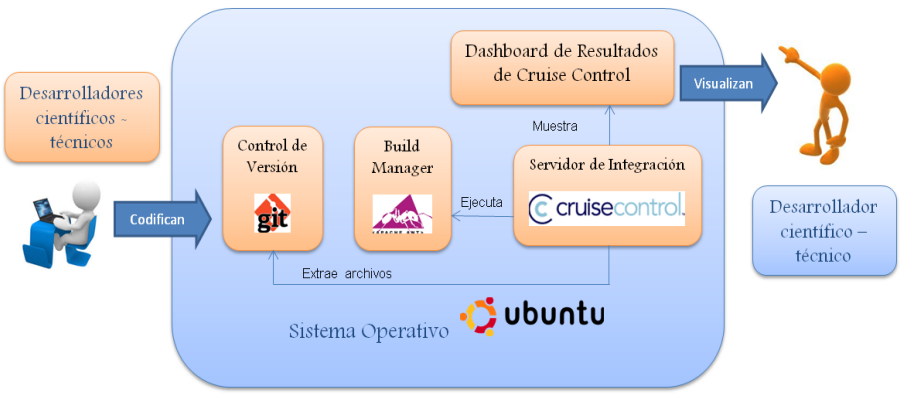
Es una capa de control de acceso para Git. Facilita la administración de los usuarios de los distintos repositorios sin necesidad de que tengan inicio de sesión en el servidor y también facilita la administración de permisos permitiendo granular a nivel de rama / etiqueta / archivo / directorio, incluyendo quién puede retroceder, crear y eliminar las ramas / etiquetas. Configura el ssh para poder acceder al servidor remotamente.

3.4 Ant

Conocido como Apache Ant, es una librería Java que se ocupa de realizar construcciones siguiendo los procesos descritos en su archivo build.xml (donde se configura el proceso de construcción), un xml altamente flexible y legible que no impone restricciones respecto a la codificación y estructura de directorios.

En la Fig. 2 se puede apreciar la descripción del modelo desarrollado, implementando las herramientas open source mencionadas.

Fig. 2. Arquitectura del modelo de referencia



4. Experiencia

La experiencia consiste en observar como trabaja un grupo de desarrolladores de software científico-técnico, antes y después de la aplicación del modelo de referencia desarrollado en este trabajo y mostrar las conclusiones.

Este grupo de desarrolladores construye software de simulación en estructuras de materiales, dicho software está desarrollado en lenguaje C y estructurado en tres componentes.

Además, todos los miembros del equipo de desarrolladores pueden realizar modificaciones en los componentes en función de sus necesidades. El grupo de desarrollo está compuesto por cinco personas: entre ellos ingenieros en diferentes ramas: informáticos, electrónicos y aeronáuticos.

- 4 desarrolladores.
- 1 líder de equipo.

4.1 Situación Actual.

Se relevó la forma de trabajo actual y se observó que utilizan un sistema de control de versiones (Git) para llevar adelante, solamente, el versionado de forma local de los componentes que están desarrollando y de esta manera no perder los avances que van logrando. Por otra parte, en los casos donde dos o más miembros del proyecto desarrollan código nuevo o realizan actualizaciones en componentes que son dependientes entre sí, al no trabajar en espacios sincronizados las modificaciones realizadas sobre escribe la versión estable que era usada por otros componentes haciendo que la ejecución del proyecto no genere los resultados esperados.

El procedimiento que aplican al utilizar Git es:

```
$ git add .

$ git commit –m 'comentario'
```

También se ha constatado que no trabajan con casos de prueba y documentación relacionada a estos, con lo cual, no realizan pruebas de ningún tipo. Es decir, la persona que desarrollo un componente determinado es la única que valida si ese componente funciona y produce los resultados esperados. Recién allí se procede a integrar dicho componente al proyecto general y verificar si este funciona acorde a las necesidades que necesitan satisfacer y que no produzca errores.

Por último, no se visualiza que existan roles en los equipos de trabajo.

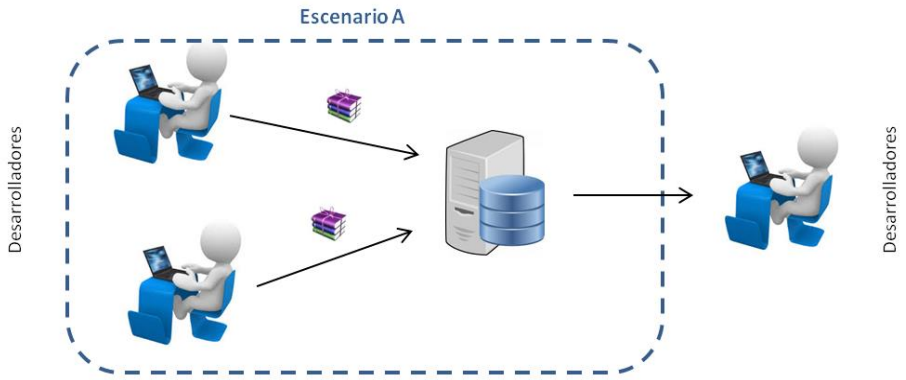


Fig. 3. Descripción del escenario A.

4.2. Propuesta de Mejora.

A continuación se definen un plan de tareas que detalla un procedimiento con las actividades a realizar indicando el responsable que se sugiere desde la Ingeniería de Software. Este proceso abarca actividades de configuración y describe las acciones en el uso de la arquitectura propiciando las buenas prácticas de la Integración Continua. Por ultimo menciona la forma de visualizar los resultados obtenidos en la integración de los componentes del proyecto.

4.3 Procedimiento y Roles

4.3.1 Creación del repositorio vacío que almacenará el proyecto.

El rol involucrado en esta etapa debe conocer la estructura del proyecto y comprender la gestión de versiones. Debe realizar lo siguiente:

1. Ingresar al servidor.
2. Clonar o actualizar el directorio gitolite-admin.
3. Abrir el archivo gitolite.conf (home/nombreUsuario/gitolite-admin/conf/gitolite.conf), con el editor que prefiera. Al final del archivo agregar el nuevo repositorio para el proyecto.
4. Guardar las modificaciones localmente y remitir los cambios al repositorio.

Para facilitar esta etapa, se ha creado un Script que realiza las tareas antes mencionadas.

4.3.2 Configuración del servidor de integración continúa.

La persona involucrada en esta fase debe conocer las tecnologías involucradas en el proyecto, los objetivos del mismo y el manejo del sistema de integración continua. Para ello, debe ser responsable de:

1. Generar el proyecto nuevo en el servidor de integración.
2. Definir la frecuencia con la que se realizara la integración de los componentes.
3. Configurar la herramienta de construcción.

4.3.3 Uso de la Arquitectura

Esto corresponde al core de actividades rutinarias y que van a ser llevada adelante por los desarrolladores. Estos replican el repositorio vacío en su espacio de trabajo local y allí crean y modifican los componentes que tienen asignados. A continuación se detallan los pasos a seguir por el desarrollador.

1. Ubicarse en el espacio de trabajo. Por ejemplo si se trabaja en el IDE de desarrollo Eclipse, seria en el "workspace".

2. En caso de que el proyecto no esté en la máquina del desarrollador debe replicar el proyecto desde el servidor con la siguiente sentencia.

```
$git clone gitadmin@ip_servidor:nombre_proyecto.git
```

En cambio si el proyecto ya está en el servidor, debe traer el proyecto del servidor en el estado actual, ejecutando la siguiente sentencia.

```
$git pull
```

3. Realizar las modificaciones o desarrollos que considere en el proyecto, incluyendo los casos de prueba, los cuales seran ejecutados por Cruise Control a la hora de realizar las builds.

4. Luego realizar un add (agregar cambios), commit (confirmar cambios) y push (enviar cambios) desde la máquina de remota, es decir desde la máquina del desarrollador donde se encuentran los archivos a agregar. Pueden ser diferentes tipos de archivos por ejemplo, .txt, .xml, .java, etc.

```
$git add

$git commit -m 'mensaje'

$git push -u origin master
```

4.2.5 Visualización de los resultados

Los responsables del proyecto van a poder visualizar los resultados de las integraciones a través del dashboard de Cruise Control. Para ello deberan ingresar a un navegador web a la siguiente URL: IP_Servidor/puerto_de_cruise_control.

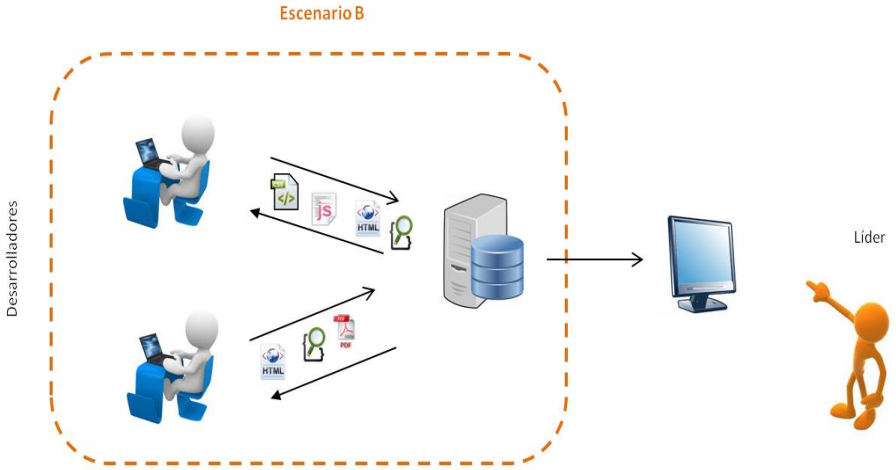


Fig. 4. Descripción de escenario B.

5. Resultado de haber aplicado la propuesta

Este trabajo presenta un modelo de referencia que no solo integra un pool de herramientas open source que permite automatizar y brindar información del proceso de building sino que provee un plan de tareas que detalla las acciones a realizar y los encargados de ejecutarlos.

A través de este modelo se logró que el equipo de desarrollo se desentendiera del proceso de compilación e integración de los componentes y que pudieran visualizar en que momento fallan las integraciones. Por otra parte, la sincronización de los espacios de trabajo ha permitido identificar cuando los desarrolladores están actualizando componentes dependientes entre sí, el sistema de integración continua informa de esta situación para evitar conflictos. Esto propicia el reúso de componentes en los proyectos.

Además, los responsables del equipo de desarrollo pueden estar al tanto en todo momento de cuál es el estado en el que se encuentra el proyecto y de administrar las configuraciones necesarias de las herramientas de integración.

6. Conclusión

La investigación y desarrollo que se ha llevado a cabo en este trabajo ha permitido obtener como resultado la integración exitosa de un sistema de integración continua (Cruise Control), un builder (Ant) y un sistema de control de versiones (Git),

permitiendo obtener como producto final un modelo de referencia que automatiza el proceso de building de software.

Además, se ha planteado un cambio en el proceso de integración de componentes, ya que el modelo diseñado en este trabajo plantea mejoras en la forma de realizar el desarrollo de software al proponer un esquema donde se integra continuamente dependiendo de la configuración del servidor Cruise Control. Esto permite realizar la integración de un proyecto a medida que avanza el desarrollo del mismo.

Por otra parte, este modelo establece una trazabilidad de los componentes que se desarrollan, posibilitando tener un registro de las modificaciones que han ido ocurriendo, y permitiendo establecer en caso de que se produzca un fallo de integración, cuál ha sido el componente que ha producido y que modificaciones específicamente han afectado la construcción.

En el marco de este trabajo, se continúa ampliando el desarrollo de la arquitectura extendiendo el concepto de integración continua hacia la gestión de incidencias, como así también hacia el modelado de indicadores desarrollando las interfaces necesarias para lograr la integración de las herramientas open source elegidas para tal fin. Se espera que este trabajo aporte buenas prácticas a la comunidad científico técnico reduciendo el re trabajo y optimizando los tiempos de desarrollo.

Referencias

1. Brooks G. Team Pace Keeping Build Times Down. Agile GILE 08 Conference (2008).
2. Spinellis D. “Software Builders” IEEE Software. Vol 25, 22-23 (2008).
3. Continuous Integration por Martin Fowler, http://martinfowler.com/articles/continuousIntegration.html
4. D. Kelly and R. Sanders. Assessing the Quality of Scientific Software. First International Workshop on Software Engineering for Computational Science and Engineering. Leipzig, Germany. (2008).
5. Carver J., Kendall R., Squires S., and Post D. Software Development Environments for Scientific and Engineering Software: A Series of Case Studies. IEEE Computer Society, Washington,DC, USA, (2007).
6. The Open Source Initiative, http://opensource.org/
7. Duvall P., Matyas S., Glover Andrew. Continuous Integration: Improving Software Quality and Reducing Risk. (2007).
8. Scot Chacon. Pro Git. (2009).
9. Segal J. Scientists and software engineers: a tale of two cultures. PPIG University of Lancaster, UK, (2008).
10. Segal J. Some problems of professional end user developers. IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC), Coeur dAlene, Idaho, USA. (2007).
11. Chalmers A. What is this called science?. Open University Press. Milton Keynes, UK. (1982).
12. http://opensource.org/licenses.
13. http://cruisecontrol.sourceforge.net/