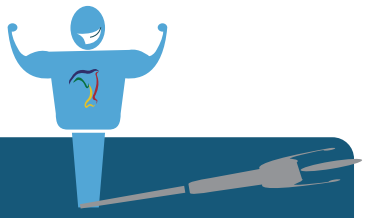
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

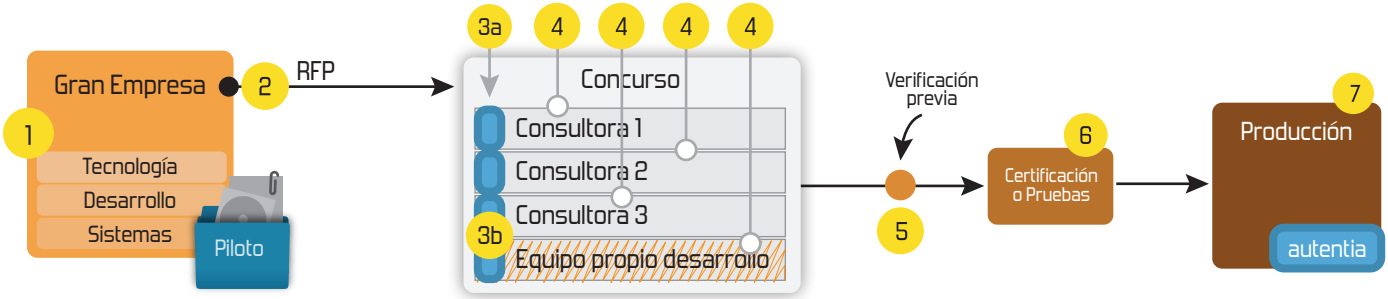
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

DESARROLLADO POR:
Alejandro Pérez García

Alejandro es socio fundador de Autentia y nuestro experto en J2EE, Linux y optimización de aplicaciones empresariales.

Ingeniero en Informática y Certified ScrumMaster

Si te gusta lo que ves, puedes contratarle para darte ayudate con soporte experto, impartir **cursos presenciales** en tu empresa o para que **realicemos tus proyectos como factoría** (Madrid). Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Fecha de publicación del tutorial: 2010-09-22

Reunión Madrid Ágil 21-09-2010: Estrategias de Branches, y división de una historia en tareas

Creación: 22-09-2010

Índice de contenidos

Histórico de NOTICIAS

Últimos Tutoriales

Instalación de subversion

Validación de acciones sobre botones en Jsf con Icefaces

Mercurial, un sistema de control de versiones distribuido

Instalación de Ubuntu Server 8.04 LTS 32bits en una máquina virtual con VMWare Workstation

Manual de Javascript

Últimos Tutoriales del Autor

Cacoo, herramienta colaborativa para hacer diagramas, incluso prototipado de pantallas o UML

Contratos Ágiles y TDD

Ejemplo de arquitectura propuesta por Autentia

EGit un plugin de Eclipse para el sistema de control de versiones distribuido Git

Git y cómo trabajar con un repositorio de código distribuido

1. Introducción

El 21-09-2010 tuvo lugar una de las reuniones de Madrid Ágil, en la que se habló de Estrategias de Branches para el repositorio de código y División de una historia en tareas para que se las reparta el equipo.

En este artículo voy a contar mis impresiones después de la reunión.

2. Estrategias de Branches

Al final las dos formas típicas de trabajar son:

- En **equipos pequeños** (normalmente no más de 4 personas): trabajar sobre el trunk directamente, y mucha comunicación entre el equipo. En este formato el trunk siempre está inestable.
- Para **equipos grandes**: Se hacen branches por funcionalidad. En este formato el trunk siempre esta estable.

Al final la idea es minimizar el número de conflictos que podemos encontrar al hacer un merge. Por eso además es recomendable hacer actualizaciones constantes y commits atómicos por funcionalidades pequeñas y muy frecuentemente, ya que no es lo mismo tener que hacer un merge de 5 ficheros que de 50.

Hay gente que para evitar los conflictos utiliza sistemas bloqueantes, de forma que una vez tienes un fichero se supone que ningún otro desarrollador puede cambiarlo. Pero se ha demostrado que al final estos sistemas no funcionan porque siempre acaba habiendo casos excepcionales que nos obligan a desbloquear alguno de los ficheros.

2.1. Y entonces ¿cuándo debo hacer un branch?

Es muy común abrir branch para hacer un hotfix de una versión (tag) determinada. De forma que se pueda hacer muy rápidamente y sin meter más ruido porque el resto del código ya ha cambiado.

También para ir desarrollando en base a funcionalidades. Cada branch es una funcionalidad.

Cuando se van a hacer cambios muy grandes que van a meter mucho ruido en el código, para no molestar a los compañeros. Aunque esto tiene el problema de que cuando queramos hacer el merge puede ser un auténtico infierno. Es preferible hacer muchos cambios pequeños que uno grande.

Cuando se trabaja con distintas versiones de un mismo producto. A veces pasa que se hacen personalizaciones de un producto para distintos clientes, esto casi como hacer un fork del proyecto. Pero en vez de crear un nuevo repositorio es mejor hacer un branch para que sea más fácil integrar (mergear) cambios concretos en proyecto ortiginal (el trunk).

2.2. ¿Y qué tal son los sistemas distribuido?

Los repositorios distribuidos se están planteando cada vez más como un opción a utilizar, o por lo menos a tener en cuenta.

Actualmente se imponen:

- Mercurial: Es un poquito más sencillo y más seguro porque trabaja con conjuntos de cambios inmutables (un commit es un commit y no lo podrás cambiar nunca).
- Git: Es más potente, pero esto puede ser un arma de doble filo porque puedes llegar incluso a cambiar el contenido de un commit, con lo que si no tienes cuidado la puedes liar parda ;)

Lo que hay que tener en cuenta es que dan mucha más flexibilidad pero, precisamente por esto, son un poquito más complejos. Nuestra elección debe estar justificada para que ese pequeño aumento de complejidad compense al equipo por sacar partido del resto de características. Algunos motivos por los que utilizar un sistema distribuido:

- Se puede trabajar sin conexión a internet, por lo que todas las operaciones son mucho más rápidas (se están haciendo en local). Luego cuando volvamos a tener conexión podremos (debemos) sincronizar nuestro repositorio completo.
- Facilitan el trabajo con las ramas.
- Facilitan los merges.

3. División de una historia en tareas

Esto que al principio puede resultar una tontería decirlo, no lo es tanto, ya que una mala división hace que se produzcan bloqueos que no tendrían por que.

Es importante que cada tarea se pueda realizar de forma independiente y sin importar el orden entre ellas. Deberíamos poder darles la vuelta, elegir una al azar, y ser capaces de hacer la tarea completamente. Si no podemos hacer esto puede ser un indicador de que no estamos haciendo correctamente la división en tareas.

Un caso típico de mala división en tareas es hacer waterfall con las tareas, de forma que tenemos una tareas para hacer el diseño, otra para hacer la implementación, otra para hacer los test.

Siempre deberíamos hacer que las tareas aporten valor por si mismas, es decir que una vez terminada la tarea se tiene terminada una parte del sistema (una parte pequeña, pero que ya se puede usar por otras partes). De forma que la historia se componga como la suma/interconexión de todas esas tareas.

Si las tareas son independientes, cada miembro del equipo puede trabajar sobre una tarea sin tener bloqueos por la tarea que está haciendo su compañero. Con esto lo que conseguimos es que todo el equipo pueda trabajar sobre la misma historia, de forma que hasta que una historia no este terminada no se pasa a la siguiente.

Otro problema típico es cuando vemos muchas historias empezadas a la vez, es decir, hay miembros del equipo trabajando en tareas de distintas historias a la vez. Esto es un "mal olor" de que nuestras historias y tareas no están bien definidas, y esto acaba generando problemas del estilo: se acaba el sprint y tenemos todas las historias al 90% pero ninguna terminada, por lo que al final del sprint no estamos aportando ningún valor al cliente (por esto la importancia de hacer las tareas por prioridad, siendo la prioridad el valor de negocio que aporta la historia).

4. Conclusiones

Todavía nos queda mucho que explorar y aprender, y estas reuniones son geniales para ello ya que nos juntamos muchos de "realidades", "contextos" y tecnologías distintas, por lo que se ven muchos puntos de vista.

También nos queda mucho que mejorar en cuanto a la organización de las propias reuniones para, por ejemplo, no desviarnos tanto de los temas propuestos, que a veces nos da por irnos por los cerros de Úbeda. Pero bueno, para eso tenemos las retrospectivas y las propuestas de mejora en el proceso ;)

5. Sobre el autor

Alejandro Pérez García, Ingeniero en Informática (especialidad de Ingeniería del Software) y Certified ScrumMaster

Socio fundador de Autentia (Formación, Consultoría, Desarrollo de sistemas transaccionales)

mailto:alejandropg@autentia.com

Autentia Real Business Solutions S.L. - "Soporte a Desarrollo"

http://www.autentia.com

COMENTARIOS


XHTML 1.0
CSS
RSS
ATOM