Tema 2 – Desarrollo Dirigido por Pruebas

Parte 4 – Integración continua y automatizada

1

Definición (1/2)

- Cada vez que se genera una nueva función o porción de código se integra con el código que ya se había generado y probado anteriormente.
- Se pretende construir y expandir incrementalmente la funcionalidad, en lugar de construir todos los componentes por separado e integrarlos al final.
- Se integra el código temprano y a menudo.
  - El código se debe integrar como mínimo una vez al día, y realizar las pruebas sobre la totalidad del sistema.

2

Definición (2/2)

- Es importante que todo el equipo pueda conocer el estado de la integración en un momento determinado, antes de añadir un nuevo código.
  - Luz roja: nadie debe incluir nada porque no es seguro (se está trabajando en un error producto de la integración en curso)
  - Luz verde: Se puede comenzar con la siguiente integración
- Se debe integrar todo el código en una máquina y realizar todas las pruebas hasta que estas funcionen al 100%.
- No podrás integrar después de unas pocas horas, a menos que:
  - Puedas ejecutar pruebas rápidamente.
  - Codifiques en parejas, así hay la mitad de cambios a integrar.
  - Recodifiques, así hay piezas más pequeñas, reduciendo la posibilidad de conflictos.

3

Características

- Localización centralizada del código fuente
- Utilización de un único comando para compilar y enlazar los ejecutables
- Soporte a la automatización de las pruebas
- Todos pueden acceder a un ejecutable confiable del sistema

4

Beneficios

- Se pretende reducir los riesgos técnicos del proyecto.
- Se acabaron las pesadas sesiones de integración en las que participaban numerosas personas para detectar los diferentes errores que se producían. Se integra desde el primer momento
- Los principales errores se manifiestan el mismo día que se producen
- Las versiones estables del código se proporcionan continuamente
  - Cuantas más integraciones, mejor
  - Cada semana – 2 horas
  - Cada día – 15 minutos
  - Cada 15 minutos – 5 minutos desatendidos
- Solamente se puede conseguir mediante integración automatizada

5

Desventajas

- Costes de puesta en marcha
  - Complejidad para instalar, configurar, mantener y operar la plataforma de integración.
  - Mantenimiento de los scripts de configuración (pueden llegar a ser complejos).
  - Dificultad para incluir código preexistente.

6

Herramientas

- Facilitan la construcción de las sucesivas integraciones a partir del código existente en el repositorio:
  - Jenkins (código abierto)
  - Bamboo
  - Cascade
  - Cruisecontrol
  - Hudson

7

Condiciones para considerar una integración correcta (1/2)

1. Última versión del código en repositorio, sin copias más actualizadas en los equipos de desarrollo:
   - Se han obtenido las últimas versiones de todos los elementos software del sistema de gestión de configuración
   - Cada fichero se ha compilado de nuevo
   - Los objetos resultantes se han enlazado y distribuido para su ejecución
2. Se ejecutan sin error las pruebas
   - Registro de pruebas
   - Script para lanzar las pruebas
   - Entono similar al de producción

8

Condiciones para considerar una integración correcta (2/2)

3. Informe de resultados fácilmente accesible a todos los miembros del equipo de desarrollo para favorecer el orden
4. Ejecutable en repositorio de código
5. Proceso con un único comando sin intervención humana durante el proceso y en minutos

9

Scripts automáticos de construcción

- Algunas veces la compilación es tan simple como (ejemplo C#) :
  - Csc *.cs
    - Pero el sistema puede necesitar muchas referencias externas, ficheros de recursos, etc. que deben estar debidamente configurados.
    - Se deben ejecutar pruebas para verificar la corrección de la construcción
    - No se desea instalar el entorno de programación en la máquina servidora de integración

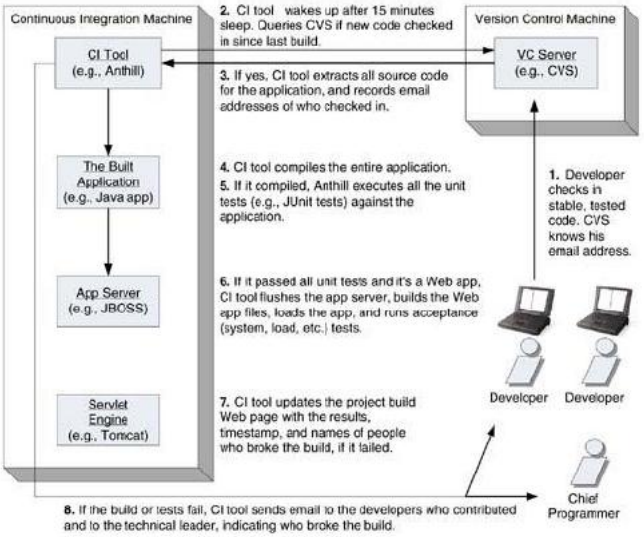
10

Ejemplo de Script de Construcción

```
<?xml version="1.0"?>
<project name="NUnit Integration" default="test">
 <property name="build.dir" value="\dev\src\myproject\" />

 <target name="build">
  <csc target="library" output="account.dll">
   <sources>
    <includes name="account.cs" />
   </sources>
  </csc>
 </target>

 <target name="test" depends="build">
  <csc target="library" output="account-test.dll">
   <sources>
    <includes name="account-test.cs" />
   </sources>
   <references>
    <includes name="nunit.framework.dll" />
    <includes name="account.dll" />
   </references>
  </csc>

  <nunit2>
   <test assemblyname="${build.dir}account-test.dll" />
  </nunit2>
 </target>
</project>
```

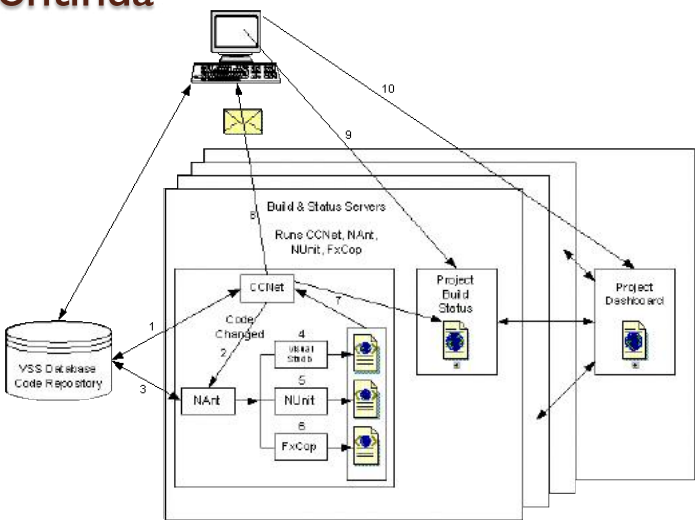

11

Integración continua



12

Entorno práctico de integración continua



13

¿Cómo se configura?

```
<cruisecontrol>
<project name="PruebaNUnit" webURL="http://xx.xx.xx.xxx/ccnet" publishExcepNons="true">
<sourcecontrol type="vss" autoGetSource="true">
<executable>C:\Archivos de programa\MicrosoT Visual SourceSafe\SS.EXE</executable>
<project>$/PruebaNUnit/PruebaNUnit</project>
<username>dhip</username>
<password>dhip2005-2006</password>
<ssdir>C:\Vss\</ssdir> <workingDirectory>C:\dhip\SourceCode\PruebaNUnit</workingDirectory> <culture>e
ES</culture>
<cleanCopy>false</cleanCopy>
</sourcecontrol> <tasks>
<nant>
<executable>C:\Archivos de programa\Nant\bin\nant.exe</executable> <nologo>false</nologo> <buildFile>\\
xx.xx.xx.xxx \DhipPruebaNUnit\PruebaNUnit.build</buildFile> <targetList>
<target>build</target>
<target>test</target> </targetList>
</nant> </tasks>
<publishers>
<xmllogger logDir="log" /> <buildpublisher>
<sourceDir>C:\dhip\Public\PruebaNUnit</sourceDir>
<publishDir>\\163.117.154.99\PruebaNUnit</publishDir> </buildpublisher>
</publishers>
</project>
</cruisecontrol>
```


14