DELIBERACIONES

AGORA@US

21/12/2014

Evolución y Gestión de la Configuración (EGC)

Grupo5

Control de versiones del documento.

Versión	Fecha	Autor	Comentarios
1.0	21/10/2014	Raquel Cerrato	Primera entrega
1.1	11/11/2014	Raquel Cerrato	Actualización
1.1.1	17/12/2014	Raquel Cerrato	Actualización
1.1.2.	19/12/2014	Raquel Cerrato	Actualización
1.2	20/12/2014	Raquel Cerrato	Preparación entrega

Contenido

1 Resumen. 5

2 Introducción. 6

3 Gestión del código fuente. 7

3.1 Ramas. 7

3.2 Parches. 8

3.3 Aprobación de cambios. 8

3.4 Roles en la gestión del código. 9

3.5 Políticas de nombres y estilos. 10

3.6 Ejercicios. 11

4 Gestión de la construcción e integración continua. 14

4.1 Herramientas usadas. 14

4.3 Realización construcción del proyecto. 16

4.5 Mecanismos IC usados. 17

4.7 Ejercicios. 18

5 Gestión del cambio, incidencias y depuración. 32

5.1 Mecanismos de depuración. 32

5.2 Gestión de cambios. 33

5.3 Procesos en la gestión de cambios. 33

5.4 Roles en la gestión de las incidencias. 34

5.5 Estados de los cambios. 35

5.6 Políticas usadas para descartar, fomentar o retardar cambios. 36

5.7 Ejercicios. 36

6 Gestión de liberaciones, despliegue y entregas. 42

6.1 Elementos entregables del proyecto. 42

6.2 Generar nuevos los entregables. 42

6.3 Identificación de entregables. 43

6.4 Publicación, liberación y entrega de entregables. 43

6.5 Roles en la gestión de entregas. 44

6.6 Mecanismos de despliegue definidos. 44

6.7 Procesos en la gestión de entregas. 44

6.8 Plataformas en la gestión de entregas. 45

6.9 Herramientas en la gestión de entregas. 45

6.10 Gestión de despliegue del proyecto completo. 46

frase rara

6.11 Ejercicios. 48
7 Gestión de la variabilidad. 52
7.1 Ejercicios. 52
8 Mapa de herramientas. 54
9 Conclusiones. 56
10 Bibliografía. 57
11 Anexos. 58
A. UML 59

1 Resumen.

En este documento se recoge toda la información relevante sobre el estudio del desarrollo, construcción y distribución del subproyecto Deliberaciones perteneciente al proyecto del grupo Agora@US, siendo una guía para la Gestión de la Configuración del proyecto.

Por tanto, podemos afirmar que se ha realizado una revisión del uso de herramientas que nos permitan al equipo realizar el desarrollo de las tareas de gestión del proyecto de forma organizada, clara y dinámica. Para realizar esto será necesario gestionar áreas como; código fuente, entregables, construcción, depuración, incidencias, depuración, variabilidad, integración y despliegue continuo dentro de nuestro subsistemas y del sistema completo del cual formamos parte.

En la información recogida en este documento podremos encontrar información relevante acerca de diferentes aspectos de la gestión de proyectos, incluyendo en ellas también las técnicas usadas para la correcta utilización como algunos enunciados de ejemplos que nos ayuden a entender mejor lo explicado en cada sección.

Para la realización de este documento ha sido necesario consultar y recopilar información de las fuentes nombrada en la Bibliografía (Véase apartado 10). Además de esto usando los contenidos vistos en la asignatura se ha realizado un análisis de las herramientas necesarias para el correcto desarrollo de nuestro proyecto, teniendo en cuenta siempre las necesidades del equipo.

Hemos facilitado en el documento un mapa de herramientas donde detallamos las herramientas utilizadas y la integración entre ellas, mostrando como se relacionan entre ellas para el correcto desarrollo del proyecto.

Por último, exponemos una conclusión sobre la gestión de nuestro proyecto Agora@US-Deliberaciones y sobre los aspectos detallados a lo largo del documento.

2 Introducción.

Este proyecto ha surgido a raíz de la asignatura Evolución y Gestión de la Configuración, de cuarto curso del Grado de Ingeniería Informática – Ingeniería del Software. En clase se nos planteó realizar un proyecto similar al realizado por unos antiguos compañeros de la Escuela, llamado Agora Voting, el profesor de la asignatura nos planteó realizar un proyecto similar Agora@US, dividiendo dicho proyecto en subproyectos más pequeños como: Autenticación, Administración y creación de voto, Cabina de votación,…

Todos los miembros de este grupo decidimos optar por el subproyecto Deliberaciones, este no era uno de los subsistemas facilitados. Es un subsistema que consiste en una BBDD de comentarios. Se almacenarán los comentarios con información relevante del votante que lo realiza y fecha del día que lo realizó. Los comentarios sólo podrán ser realizados por un votante autentificado y censado en la votación a deliberar.

En el documento actual va a encontrar información y datos relevantes de nuestro subsistema y del subsistema completo, además de ejemplos genéricos. A continuación se detalla cómo se encuentra estructura el documento.

- **Resumen e Introducción:** En estos apartados se pondrá en contexto al lector del documento, dándole a conocer que proyectos hemos seleccionado.
- **Gestión del código fuente:** falta poner algo, no?
- **Gestión de la construcción e integración continua:** Se definirán los procesos que se usan a la hora de construir el proyecto, como las herramientas utilizadas, mecanismos IC,…
- **Gestión del cambio, incidencias y depuración:** Se detallarán los mecanismo de depuración, procesos, roles,… utilizados durante la vida del proyecto.
- **Gestión de liberaciones, despliegue y entregas:** Se detallará que es un entregable, como generarlos e identificarlos, y diversos aspectos que son relevantes de los entregables
- **Gestión de la variabilidad:** Explica los mecanismos empleados, o que pueden ser utilizados para gestionar la variabilidad en el proyecto, y en caso de no existir, define cómo podría llegar a incluirse.
- También contiene un **Mapa de herramientas** cómo las distintas herramientas se interconectan para poder gestionar y desarrollar el código eficientemente tanto el de

nuestro subsistema como el proyecto completo Agora@US y se finalizará el documento con una **Conclusión**, **Anexos** y **Bibliografía**.

3 Gestión del código fuente.

3.1 Ramas.

necesitar no es el verbo, sería "usar"

Para el desarrollo del código fuente en *Deliberaciones* necesitamos la herramienta de gestión de código github que nos permitirá tener un repositorio del código dónde podemos crear ramas según las necesidades que tengamos a lo largo del proyecto.

En este proyecto existen tres tipos de ramas, **master** es la rama principal del proyecto y contiene el código definitivo que finalmente será el código entregado. Una rama para la documentación del proyecto llamada **nombreRama,** que contendrá todos los entregables más importantes del proyecto. Por último, la ramanombreRamaque será la rama destinada a la implementación de la nueva funcionalidad.

Puede darse el caso durante la gestión del código fuente fuera necesario en algún momento añadir una nueva rama al proyecto, usaríamos la siguiente instrucción:

>> git branch nombreRama

Cuando necesitemos añadir nuevos archivos o documentos, se debe realizar antes de hacer el commit para que así estén en el repositorio local, sino se perderían dichos archivos o documentos, procedemos con la siguiente instrucción para añadir el archivo a alguna de las ramas existentes del proyecto:

>>git add nombreArchivo.extension

Los commit se realizarán

>>git commit –m “Comentario”

Para unir una rama a la rama principal nos situaremos sobre la principal y usaremos la siguiente instrucción:

>>git checkout master

>>git merge nombreRama

Falta dar una visión global de cuándo y por qué crear ramas. La política de creación y destrucción. Cómo y cuándo se hacen "merge" El tema de los comandos concretos de git para esto sería parte de un ejercicio pero no necesariamente formaría parte de este apartado.

3.2 Parches.

Si en algún momento a lo largo de la gestión del código fuente se desea añadir parches a una determinada rama tenemos que realizar lo siguiente:

- Creamos la rama donde podremos realizar el parche:

  >>gitcheckout –b (nombreEspacioTrabajo)

- Introducir un parche mediante una URL:

  ¿Qué es? ¿cómo lo usáis?

  >>git cl patch https://.... (url Trabajo)

- Introduciremos el parche a través de números de cuestión:

  >>git cl patch (12345)

le pasa lo mismo que a la sección 3.2

3.3 Aprobación de cambios.

A lo largo de la gestión del código fuente puede darse el caso de que surjan conflictos tanto sintácticos como semánticos, por lo que en estos casos se reunirán los miembros necesarios para la aprobación de cambios, realizando así un estudio de las diferencias existentes entre los códigos conflictivos y decidir la manera de resolverlo, ya sea realizando una adaptación del código en conflicto o descartar uno de ellos.

El equipo de trabajo se ha dividido en dos equipos:

- Equipo de desarrollo de código y gestión de Git:

  José María Caballero

  Antonio León

  Rafael Rodríguez

- Equipo de documentación:

  Raquel Cerrato

  Julio Pineda

  Salvador Herrera

  Francisco Javier Reina

  Edwin P. Arévalo

  Daniel Toledo

Cada uno de los equipos se reunirán y acordarán si se aprueban los cambios realizados en cada equipo o no, y más tarde se pondrá en consentimiento del otro equipo para verificar dicha aprobación de los cambios realizados.

lo mismo: poco completo y no claro

3.4 Roles en la gestión del código.

En la gestión del código fuente hemos utilizado los siguientes roles.

- **Desarrolladores de código:** Son aquellos miembros del grupo de trabajo encargados de generar código y de desarrollar las nuevas funcionalidades del subproyecto Deliberaciones del proyecto Agora@US, Estos son los encargados de realizar los commit del proyecto. Una vez que finalizan el trabajo que les ha sido pedido, deben comunicárselo a los Supervisores de código.

- **Supervisores de código:** Son aquellos miembros del grupo de trabajo encargados de aprobar el código de determinadas áreas concretas del trabajo en el que están especializados, verificando que los cambios realizados por los desarrolladores de código pasan todas las pruebas necesarias y que además cumplen la nueva funcionalidad requerida.

- **Supervisores de rama:** Son aquellos miembros del grupo de trabajo encargados de supervisar las ramas y verificar que se están realizando correctamente …

están bien estos roles pero debería explicarse por qué surgen así, por qué se han elegido estos roles, ejemplos de cómo se han ejercido los roles,...

3.5 Políticas de nombres y estilos.

Las políticas de nombres y estilos utilizadas en este proyecto, como norma general vamos a utilizar las políticas de estilos predefinidos de Java, que se pueden encontrar en la web http://www.oracle.com/technetwork/java/codeconvtoc-136057.html, además de seguir un estilo propio que va a ser las definido a continuación:

- Estilo código Java: A lo largo de todo el proyecto usaremos UTF-8 para controlar los problemas que puede surgir con las tildes y con el carácter "ñ". El nombre de las clases comenzarán por mayúsculas y cada palabra que lo forme también en Java lo escribiríamos; "CreaccionNuevaClase". El nombre de los atributos y métodos deben ir en minúsculas, a no ser que sea un nombre compuesto por más de una palabra irá todo en minúscula salvo la primera letra de la palabra nueva, por ejemplo "Creación nuevo tema" en Java lo escribiríamos: "creaccionNuevoTema".

- Es muy aconsejable que dentro de cada clase los métodos se encuentren bien comentados de forma clara, para que un miembro que no haya realizado esa funcionalidad pueda saber rápida que es lo que realiza cada método de cada clase, para escribir comentarios importantes lo señalaremos con esta forma de realizar comentario en eclipse "/** comentario **/" esto lo usaremos por ejemplo para aclarar que realiza cada método, si por el contrario los comentarios son para orientarnos o recordar algo sin llegar a ser importante usaremos "// comentario"

Hilo.java | *User.java

```
//relationShips


Collection<Comment> comments;
Collection<Hilo> threads;

@NotNull
@OneToMany(mappedBy="user")
public Collection<Comment> getComments() {
    return comments;
}
/**
 * Cambia de valor al atributo comments
 * @param comments
 */
public void setComments(Collection<Comment> comments) {
    this.comments = comments;
}
@NotNull
@OneToMany(mappedBy="user")
public Collection<Hilo> getThreads() {
    return threads;
}
public void setThreads(Collection<Hilo> threads) {
    this.threads = threads;
}
```

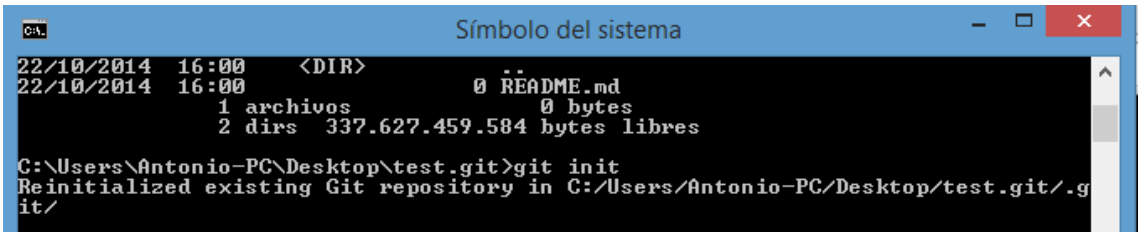
¿¿se han usado herramientas para gestionar los estilos del código??

3.6 Ejercicios.

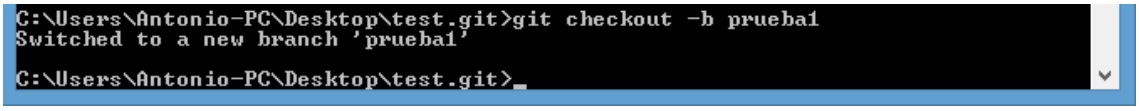
ejercicio bastante siempre y no tan relacionado con el subsitema

Ejercicio 1: Crear una rama a partir de la rama principal añadir un archivo y comprobar que se puede cambiar de rama correctamente.

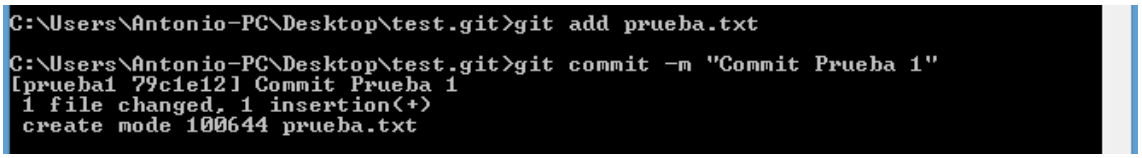
Primero nos situamos en la carpeta donde esté nuestro repositorio e iniciamos git.



A continuación creamos la nueva rama.



Por último añadimos el archivo a la rama mediante el comando add **(gitaddnombre_archivo)**, hacemos commit **(git commit -m "Nombre del commit")** y posteriormente lo subimos al repositorio mediante push **(git push origin nombre_denuestro_respositorio)**.



Ejercicio 2: Hacer un commit.

Nos aseguramos de que no estamos situados en la rama que queremos borrar y ejecutamos el siguiente comando:

```
igin https://github.com/antleocar/Deliberaciones.git
fatal: remote origin already exists.

C:\Users\Antonio-PC\Desktop\UNIVERSIDAD\EGC\Deliberaciones.git>git checkout docu
ementos
error: pathspec 'docuementos' did not match any file(s) known to git.

C:\Users\Antonio-PC\Desktop\UNIVERSIDAD\EGC\Deliberaciones.git>git checkout docu
mentos
Already on 'documentos'

C:\Users\Antonio-PC\Desktop\UNIVERSIDAD\EGC\Deliberaciones.git>git add "Taller d
e creacion de grupos.txt"
fatal: pathspec 'Taller de creacion de grupos.txt' did not match any files

C:\Users\Antonio-PC\Desktop\UNIVERSIDAD\EGC\Deliberaciones.git>git add "Taller d
e creacion de grupos .txt"

C:\Users\Antonio-PC\Desktop\UNIVERSIDAD\EGC\Deliberaciones.git>git commit -m "Su
bida de primer entregable de la wiki"
[documentos 76954bd] Subida de primer entregable de la wiki
 1 file changed, 35 insertions(+)
 create mode 100644 Taller de creacion de grupos .txt

C:\Users\Antonio-PC\Desktop\UNIVERSIDAD\EGC\Deliberaciones.git>_
```

Ejercicio 3: Unir la rama a la branch principal localmente.

Para ello primero nos situamos en nuestra rama **(gitcheckoutnombre_rama)** y a continuación hacemos rebase **(git rebase master)**.

```
C:\Users\Antonio-PC\Desktop\test.git>git checkout prueba1
Already on 'prueba1'

C:\Users\Antonio-PC\Desktop\test.git>git rebase master
Current branch prueba1 is up to date.
```

En muchos sitios los comandos no está bien escritos

Ejercicio 4: Eliminar la rama.

Nos aseguramos de que no estamos situados en la rama que queremos borrar y ejecutamos el siguiente comando:

>>gitbranch -D nombre_de_la_rama_que_queramos_borrar

```
C:\Users\Antonio-PC\Desktop\test.git>git branch -D prueba1
Deleted branch prueba1 (was 79c1e12).
```

Ejercicio 5: Realizar copia del repositorio.

Nos aseguramos de elegir la carpeta en la que queremos que se cree una copia del proyecto, para ello ejecutamos los siguientes comandos:

```
C:\Users\Antonio-PC\Documents\Nuevo Deliberaciones>git clone https://github.com/
ECGDeliberaciones/proyecto.git
Cloning into 'proyecto'...
remote: Counting objects: 263, done.
remote: Compressing objects: 100% (154/154), done.
remote: Total 263 (delta 109), reused 242 (delta 97)
Receiving objects: 100% (263/263), 329.17 KiB | 42.00 KiB/s, done.
Resolving deltas: 100% (109/109), done.
Checking connectivity... done.
```

Ejercicios en general muy básicos de git
Lo ideal sería usar los ejercicios para trabajar con ejemplos reales que hayáis usado en vuestro proyecto

4 Gestión de la construcción e integración continua.

4.1 Herramientas usadas.

Las herramientas que hemos usado para desarrollar nuestro módulo del proyecto son:

- Apache Tomcat Developer 7.0.35
- Apache Tomcat Service 7.0.35
- MySQL Server 5.5
- Apache Maven 3.1.0
- Eclipse Indigo
- JDK 7.u11
- Conjunto de utilidades necesarias: Framework Spring MVC 3.2.4, Hibernate 4.2.3, JPA (Javax 2.5 y JPQL).

Procedemos a describir brevemente las características principales de cada una de las herramientas citadas anteriormente: (no sería realmente necesario)

Apache Tomcat Developer 7.0.35 es el servidor de aplicaciones que hemos elegido. Junto a esta herramienta se usa también la versión Service 7.0.35, que es la que se encarga de lanzar el proceso de ejecución del sistema de información web que hemos creado.

El sistema gestor de bases de datos que hemos utilizado es **MySQL Server 5.5.** Se trata de un SGBD muy utilizado en aplicaciones web y de contrastada eficacia, de administración relacional de bases de datos y un software de código abierto.

Maven es la herramienta elegida para gestionar el proyecto Java de nuestra aplicación. Es un elemento muy interesante dado que es capaz de descargar dinámicamente bibliotecas y plugins necesarios de un repositorio, añadiendo dependencias necesarias. Además, se puede gestionar el proyecto desde una línea de comandos, haciendo tareas básicas como compilar, ejecutar o limpiar dicho proyecto.

En lo referente al entorno de desarrollo integrado, hemos utilizado **Eclipse Indigo**, por la razón de que ya teníamos constancia que era la versión de Eclipse que mejor funcionaba con el resto de componentes utilizados.

De manera análoga al párrafo anterior, se ha usado el **JDK 7.u11** debido a que su contrastado buen funcionamiento con el resto de componentes.

Por último, vamos a describir de manera somera el resto de utilidades utilizadas. El Framework **Spring es** un conjunto de módulos que podemos usar a nuestro antojo, como por ejemplo en nuestro caso, que hemos usado el módulo Spring MVC, Modelo Vista Controlador. El núcleo de Spring realiza una inyección de dependencias, lo que significa que la creación de objetos las lleva a cabo un contendor externo inyectándolos a otros objetos que dependan de los primeros. Esto nos proporciona una mayor limpieza de nuestro código, ya que desaparecerá de él toda la creación de objetos y dependencias. Para reflejar las relaciones entre los objetos y la creación de éstos, tendremos un fichero XML que gestionará estas características.

Hibernate es una herramienta de Mapeo objeto-relacional (ORM) para Java que lleva a cabo el mapeo de atributos de una base de datos relacional típica y el modelo de objetos de nuestra aplicación, mediante archivos declarativos como XML y anotaciones en los beans de las entidades para establecer las relaciones entre ellas en la base de datos. Es un software libre con licencia GNU LGPL.

JPA (Java Persistence API) es la API de persistencia de datos que hemos usado en el proyecto. Se trata de un framework de Java que se encarga de manejar datos relacionales en las aplicaciones. Esta API está definida en el paquete Javax.persistence y también está relacionada con el lenguaje de consulta Java Persistence Query Languaje (JPQL). JPQL se usa para hacer consultas contra las entidades almacenadas en una base de datos relacional (MySQL Server 5.5 en nuestro caso). Está inspirado en el lenguaje SQL y su sintaxis es muy similar. Sin embargo, JPQL opera con objetos entidad de JPA, en lugar de hacerlo directamente con las tablas de la base de datos, lo cual otorga mucha flexibilidad a las consultas, puesto que se realizan sobre objetos, con todas sus ventajas de acceso a métodos y atributos de cada uno.

4.3 Realización construcción del proyecto.

A la hora de llevar a cabo este apartado, hay que decir que existen tres tipos de situaciones en las que se llevará a cabo la construcción de nuestro proyecto:

La primera es por el hecho de comenzar a desarrollarlo. Para que los programadores puedan comenzar a trabajar necesitan tener unas pautas a seguir para montar el entorno donde desarrollar el software requerido. Esta es la opción más lógica y la que con más frecuencia hará que llevemos a cabo este proceso.

La segunda situación es la incorporación de un nuevo miembro al equipo de trabajo, el cual necesitará de un entorno nuevo para poder comenzar a trabajar.

La tercera viene determinada por el hecho de que algún entorno quede inestable por algún error inesperado y las personas que estén haciendo uso de él se queden paradas por no tener el entorno de desarrollo disponible. Hay que matizar que cuando mencionamos "error inesperado" nos estamos refiriendo a cualquier incidencia relacionada con el entorno de trabajo que pueda alterar el flujo normal de trabajo: errores en la base de datos, fallos del servidor web, inconsistencias del workspace, etc.

Con objeto de facilitar la tarea de construcción, hemos creado una máquina virtual con todo el procedimiento de configuración preparado para en caso de tener que realizar una nueva construcción del proyecto, ésta sea inmediata y el tiempo perdido en arreglar el imprevisto sea el mínimo posible.

Interesante. Más detalles serían de apreciar y le darían valor al apartado

Para realizar la construcción de nuestro entorno de trabajo, vamos a necesitar las herramientas que está mencionadas con detalle en el punto 4.1. Dado que ya han sido descritas en dicho punto anterior, ahora se explicará directamente los pasos que hay que seguir para llevar a cabo el proyecto del subsistema "Deliberaciones" de Agora@US.

4.5 Mecanismos IC usados.

La integración continua es una práctica muy importante en el desarrollo software en el que los miembros de un equipo de proyecto integran su trabajo con frecuencia (como mínimo, una vez al día).

Cada integración prueba automáticamente la compilación del código fuente y obtiene un ejecutable. A esto se le llama "build". Además, también se pueden realizar pruebas y métricas para asegurar la calidad del software. Es una buena práctica realizar "builds" periódicamente, ya que así se conseguirá un producto final más fiable y probado, y por tanto, dará menos fallos en producción.

Para poder llevar a cabo este proceso de integración continua, la herramienta más popular es Jenkins, que es un servidor de integración continua gratuito y open source. La base de Jenkins son las tareas. En una tarea se indica qué cosas vamos a realizar en una "build". Un ejemplo de tarea puede ser compilar el código del proyecto. Si el resultado no fuera el esperado, el servidor Jenkins es capaz de notificar el error al desarrollador o a quien sea necesario, por email o cualquier otro medio.

Cabe decir que para usar Jenkins, es fundamental tener un repositorio de control de versiones. Jenkins soporta Git, SVN, Mercurial, etc. Sin un repositorio no sería posible sacar el máximo partido al servidor de integración continua, puesto que uno de sus puntos fuertes es la monitorización del control de versiones y reaccionar ante cualquier cambio que pueda producirse.

En nuestro proyecto, dado que el tema de integración continua y la práctica asociada a Jenkins se vieron ya en un estado bastante avanzado del proyecto, no se han usado estos mecanismos de integración continua. Sin embargo, somos conscientes de la importancia que tiene usar un mecanismo de este tipo en proyectos software. Por esta razón, hemos realizado algunas pruebas con Jenkins para familiarizarnos con una herramienta tan útil como Jenkins.

Dado que nuestro proyecto utiliza el framework Spring junto con Maven usando el servidor web Tomcat, se van a realizar dos ejercicios con Jenkins. El primero será instalar Jenkins en un servidor Tomcat descargando desde la web oficial el archivo war directamente. El segundo ejercicio consistirá en configurar Jenkins para que a través de Maven compile el código de nuestro proyecto.

No se detallan los procesos. Usar Jenkins no hace que directamente ya estemos usando IC pues depende de cómo lo usemos nos dará mejor o peor resultado.

4.7 Ejercicios.

<u>Ejercicio 1:</u> Suponiendo que, por alguna de las razones descritas arriba, es necesario construir un entorno de trabajo para el proyecto, realizar todos los pasos necesarios para tener disponible el entorno de desarrollo y empezar a trabajar.

1. **Instalación y configuración de las herramientas.**

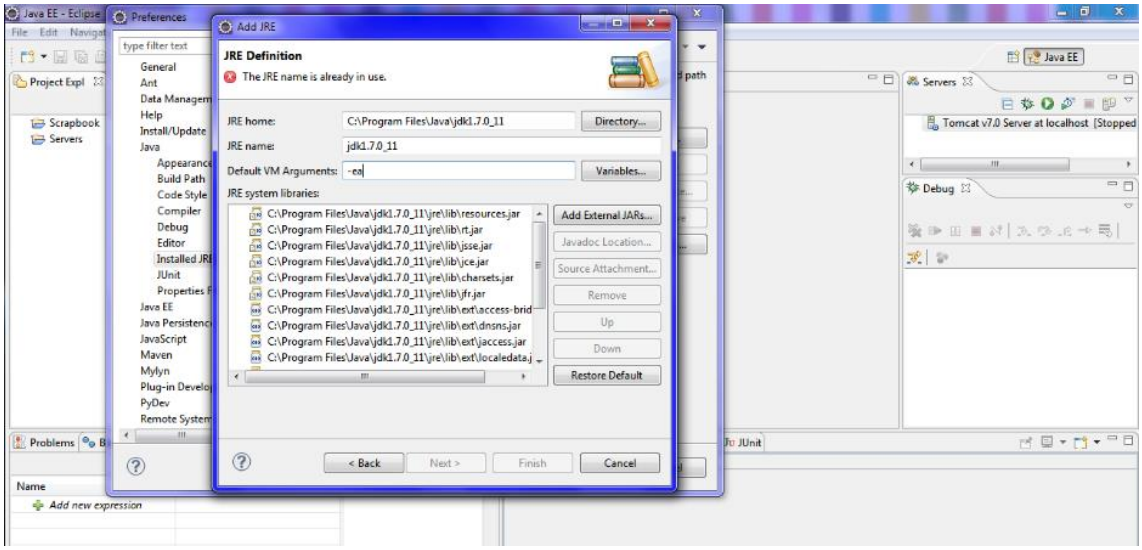
El primer paso para configurar el entorno de desarrollo es instalar las herramientas mencionadas anteriormente con los ajustes que se muestran a continuación.

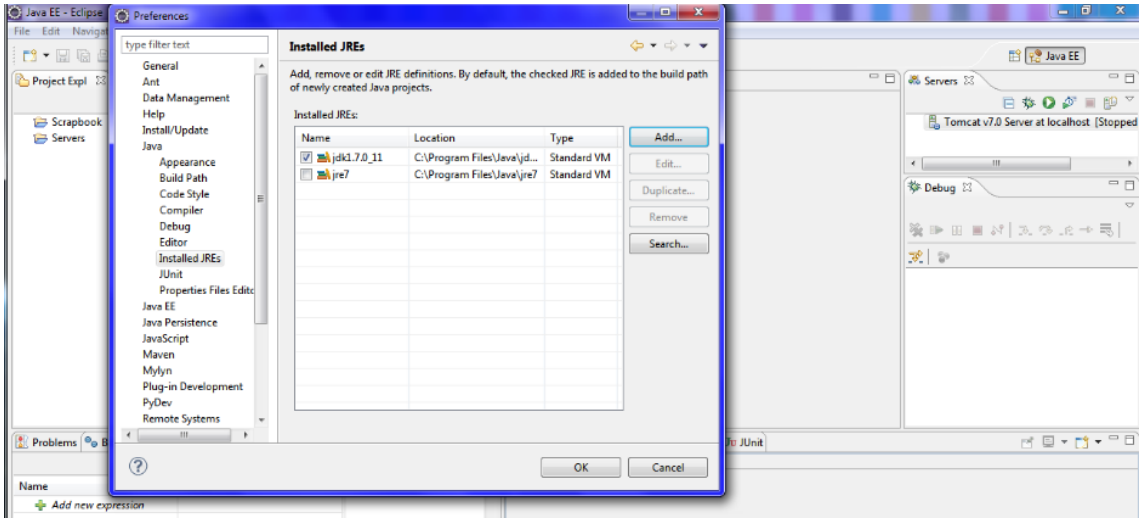
2. **Configurar Eclipse.**

2.1. Establecer Java Runtime Environment

Abrimos Eclipse y configuramos Java Runtime Environment, de la siguiente forma:

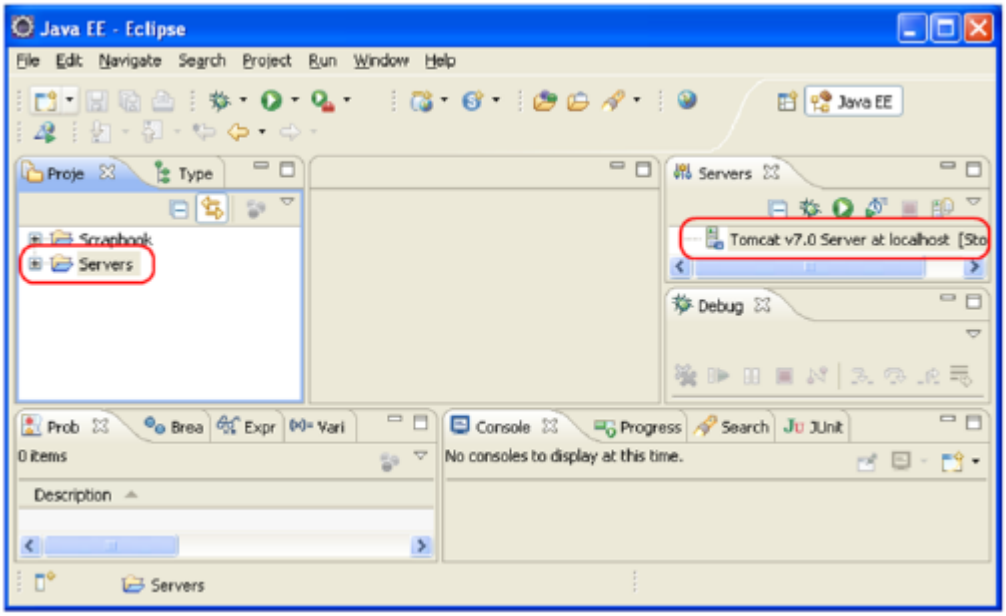
Windows → Preferences → Java right tab → Installed JREs → Add → Standard VM → Next → En el botón del directorio (buscar in **C:/Programa/Java/jdk1.7.0_11** (seleccionar y pulsar OK)) **→Finish.**



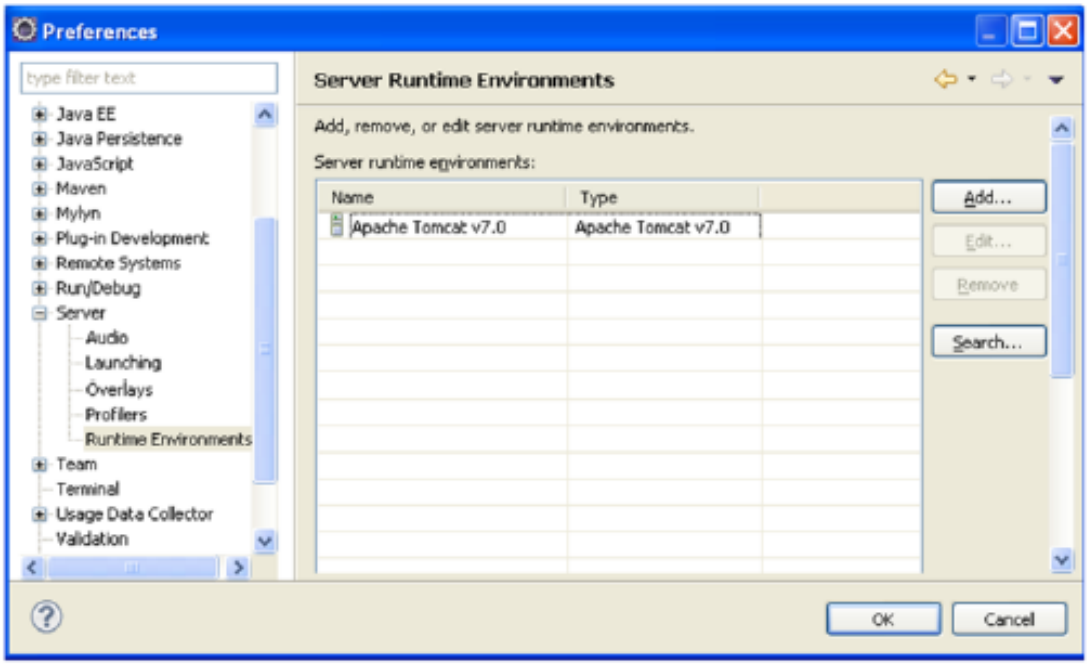


2.2. Servidor Tomcat.

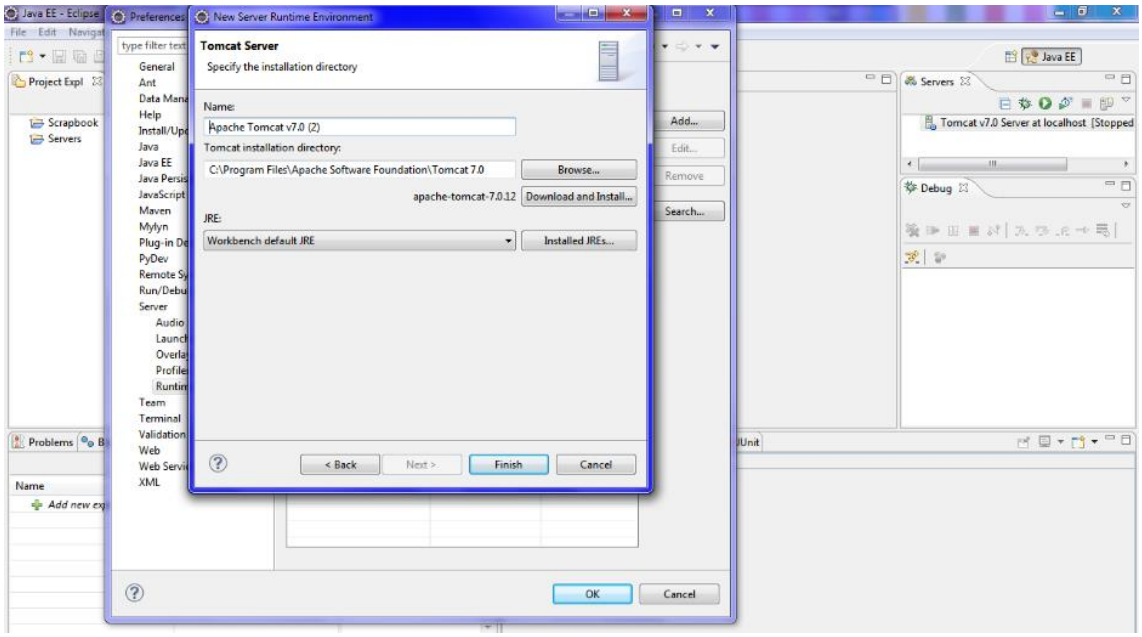
Una vez que se ha configurado el JRE, debemos borrar el servidor Tomcat que nos aparece.



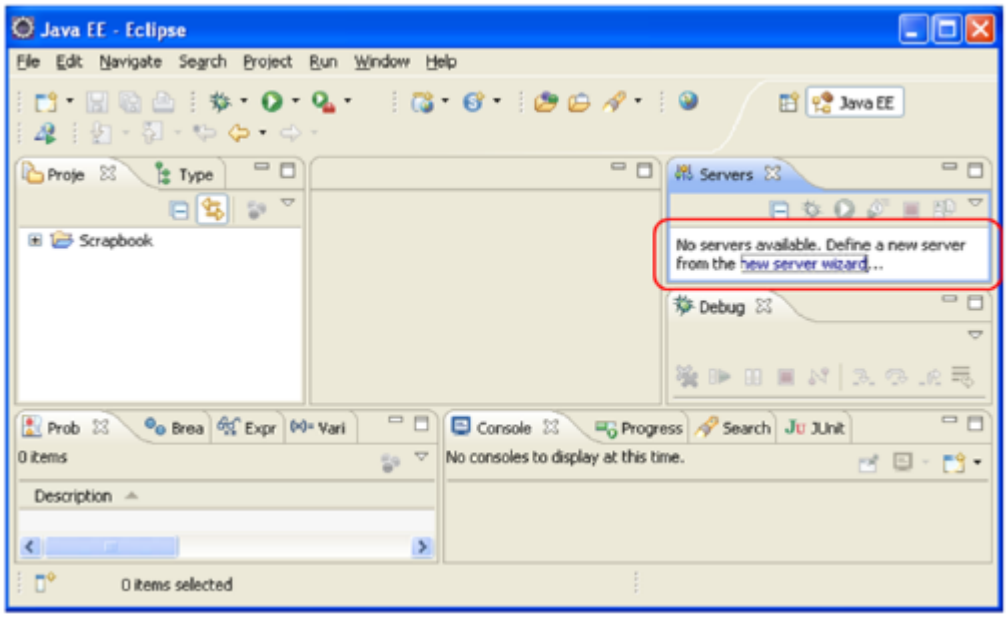
Para añadir un nuevo servidor Tomcat, necesitamos realizar los siguiente pasos: **"Window > Preferences**" y luego buscar "Runtime Environments". Esto muestra una pantalla similar a la que está en la imagen. Si existe algún servidor disponible, al igual que dicha imagen, se deben eliminar. Una vez que se haya hecho esto, debemos hacer clic en "Add".



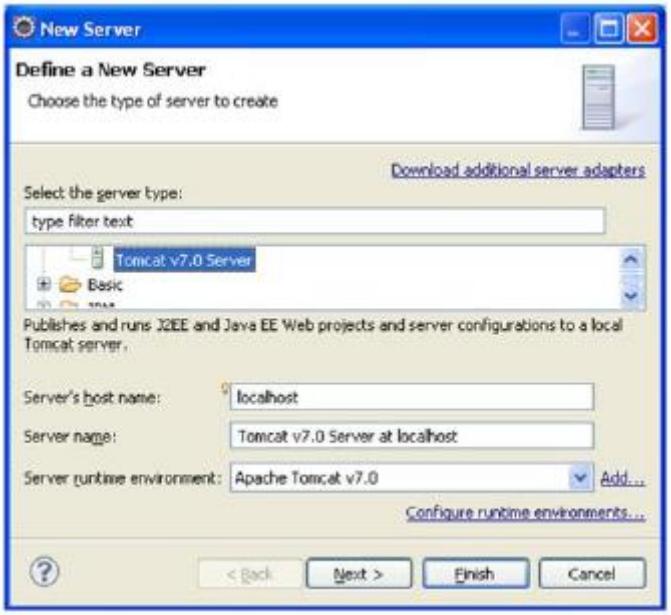
Seleccionamos el servidor "Apache Tomcat v7.0" y hacemos clic en "Next>". Luego hacemos clic en "Browse…" y elegimos la carpeta donde está instalado el Tomcat. Debemos asegurarnos de haber seleccionado "Workbench default JRE". Seguidamente, hacemos clic en "Finish".



En los pasos anteriores hemos registrado un nuevo servidor en Eclipse. Ahora debemos crear una instancia del servidor. Para realizar esto, hacemos clic en el enlace "new server wizard" en la lista de servidores.

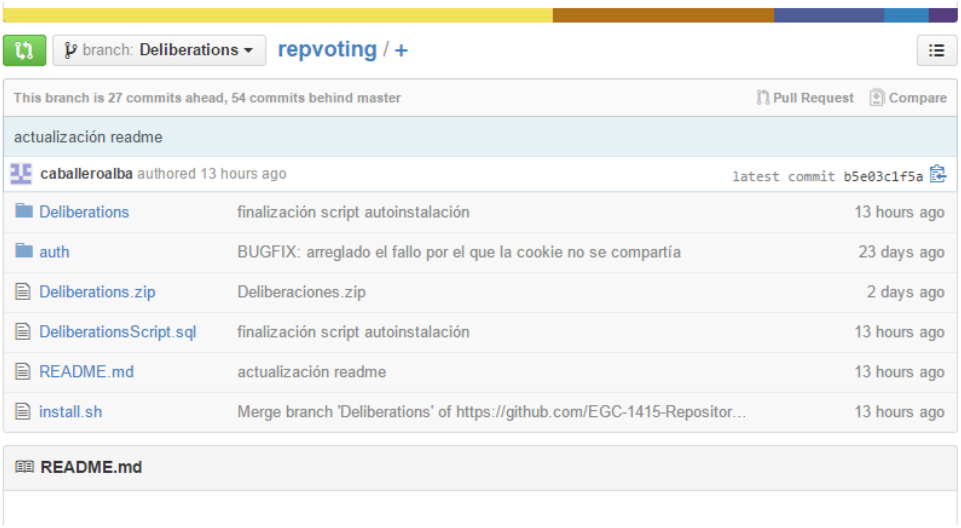


Configuramos esta instancia tal cual se aprecia en la imagen de abajo.



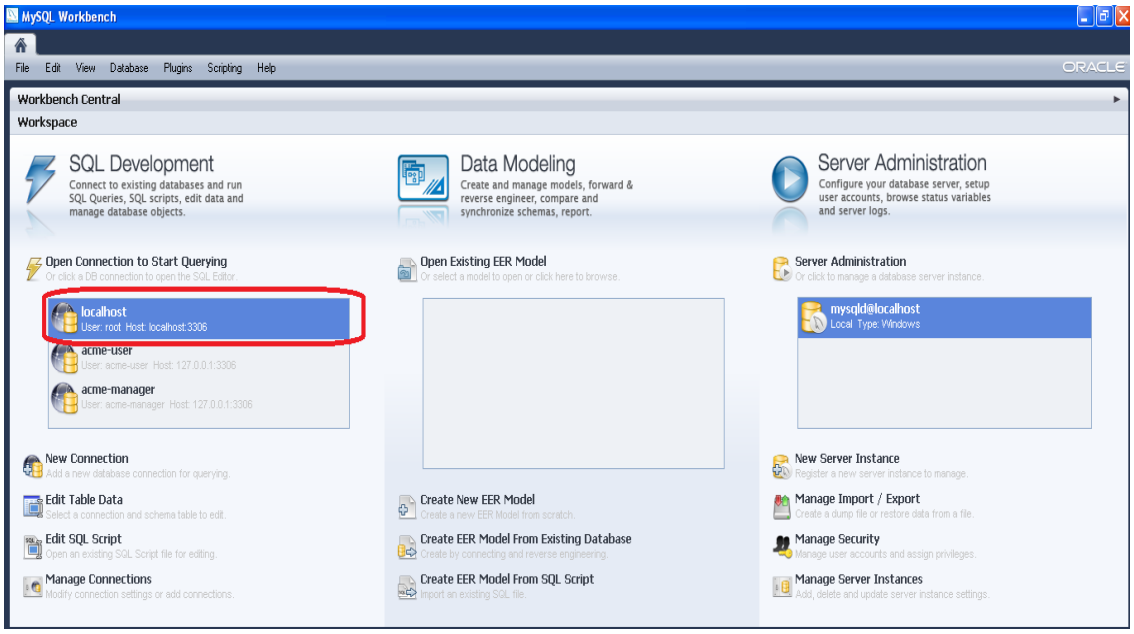
3. Importar el Proyecto.

Para trabajar con nuestro proyecto necesitamos importarlo desde Github. La URL para clonar el proyecto es https://github.com/EGC-1415-Repositorio-compartido/repvoting.git



4. Creación de la base de datos.

Abrimos MySQL Workbench, haciendo doble clic en "localhost":



Debemos introducir y ejecutar los siguientes scripts:

```
create user 'acme-user'@'%'
identified by password
'*4F10007AADA9EE3DBB2CC36575DFC6F4FDE27577';
create user 'acme-manager'@'%'
 identified by password
'*FDB8CD304EB2317D10C95D797A4BD7492560F55F';
```

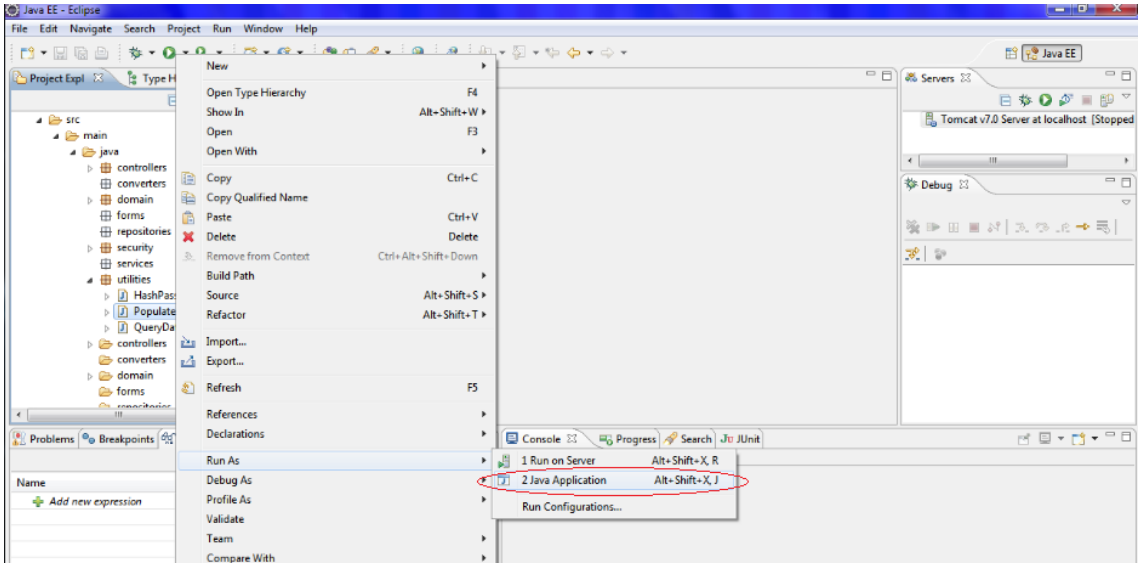
```
drop database if exists `Deliberations`;
create database ` Deliberations `;

grant select, insert, update, delete
        on ` Deliberations `.* to 'acme-user'@'%';
grant select, insert, update, delete, create, drop, references, index, alter,
      create temporary tables, lock tables, create view, create routine,
      alter routine, execute, trigger, show view
        on ` Deliberations `.* to 'acme-manager'@'%';
```

Demasiados detalles de cómo "montar" el entorno de desarrollo compardo con los detalles de otras partes más importantes del documento.

5. Popular la base de datos.

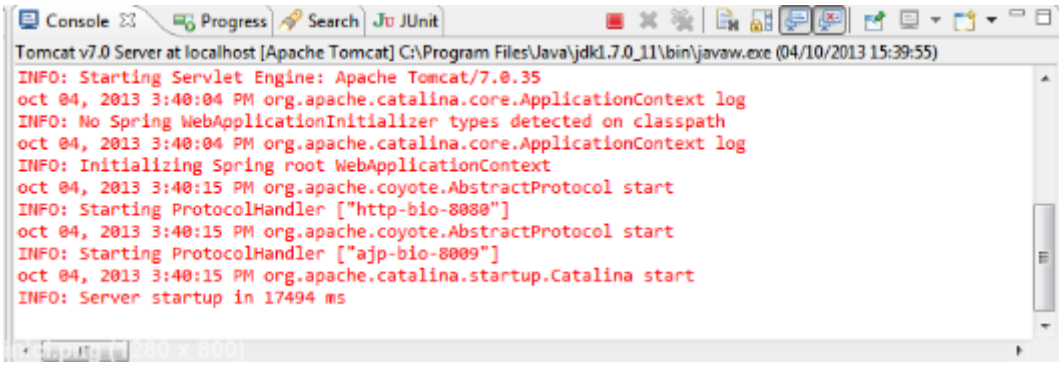
Para popular la base de datos con nuestra información e introducirla en nuestra base de datos, buscamos la clase "PopulateDatabase.java", hacemos clic en el botón derecho del ratón y seleccionamos Run as > Java Application.



Con esto ya tenemos nuestro proyecto con algunos datos insertados en la base de datos.

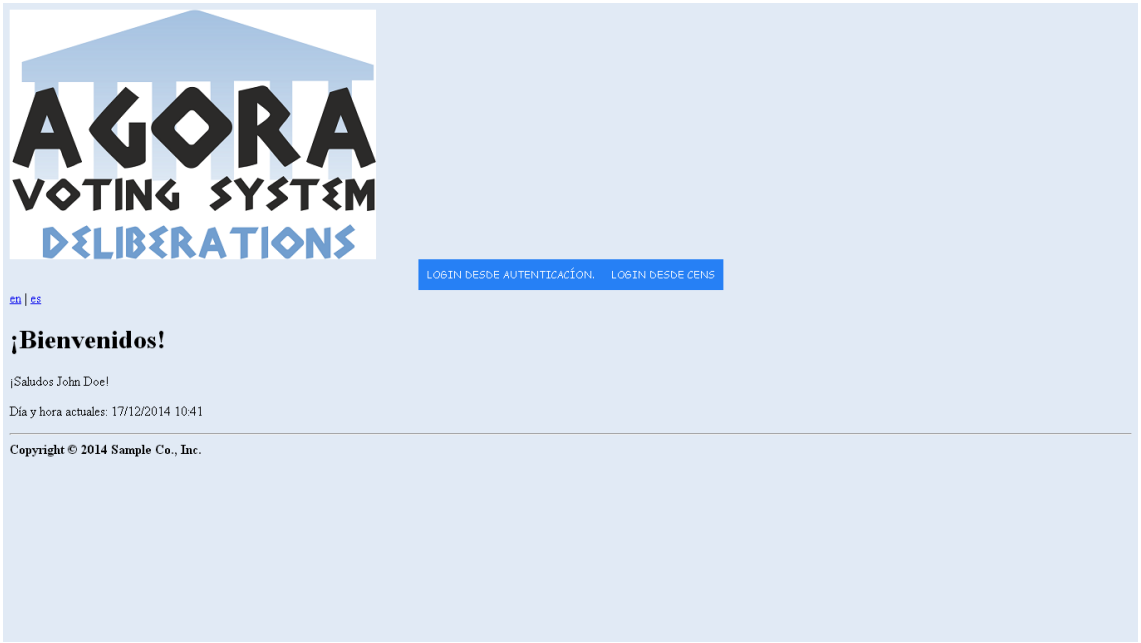
6. Iniciar Tomcat.

Para inicializar el servidor hacemos clic con el botón derecho en Tomcat v7.0> Add and Remove y añadiremos el proyecto. Tomcat se ha inicializado y nosotros podremos acceder a nuestra aplicación desde cualquier navegador con "localhost: 8080/ Deliberations"



Esta imagen muestra la información de la consola de Tomcat indicando que nuestro servicio ya está corriendo en el servidor.

Finalmente, accedemos a la dirección "localhost: 8080/ Deliberations" y entramos a la web del proyecto.

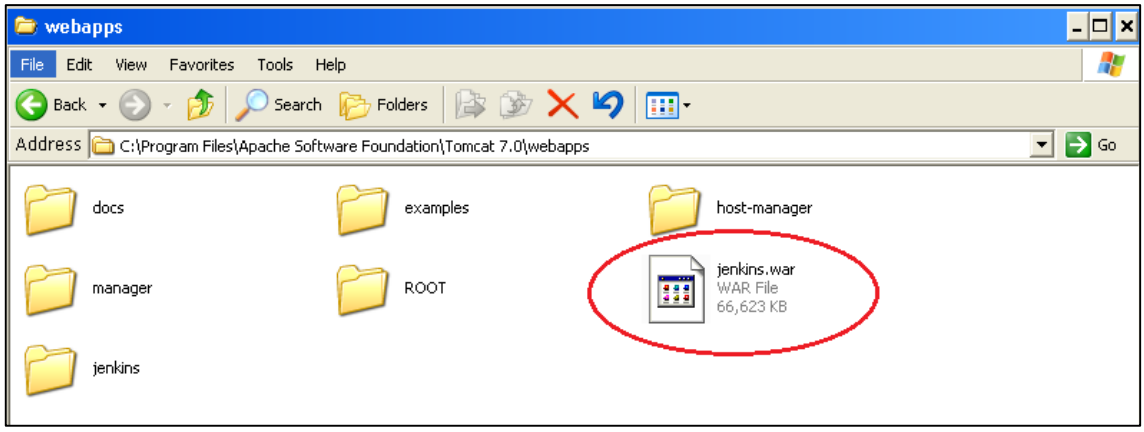


Ejercicio 2: Instalar Jenkins en un servidor web Tomcat 7.

Dado que ya tenemos un servidor Tomcat instalado en nuestra máquina de pre-producción, lo primero que debemos hacer es descargar de la página oficial de Jenkins la versión war del programa. En nuestro caso, para sistema operativo Windows XP de 64 bits.

Instalar jenkins en sí no tiene tanto interés como saber cómo usarlo y para qué

A continuación tenemos que copiar el archivo war descargado en el directorio "webapps" de Tomcat.



Además tendremos que decirle a Tomcat que tiene que usar codificación UTF8 para que entienda las URLs. Esto lo haremos editando el fichero "server.xml" situado en la carpeta "conf" y añadiendo el parámetro "URIEnconding="UTF-8" en el conector del puerto 8080.

```
<Connector port="8080" protocol="HTTP/1.1"
    connectionTimeout="20000"
    redirectPort="8443"
    URIEncoding="UTF-8" />
```

El siguiente paso será reiniciar el servidor Tomcat para que los cambios efectuados tengan efecto. A continuación tecleamos la siguiente dirección en nuestro navegador: http://127.0.0.1:8080/jenkins .

Si todo ha ido bien, deberá aparecer lo siguiente:

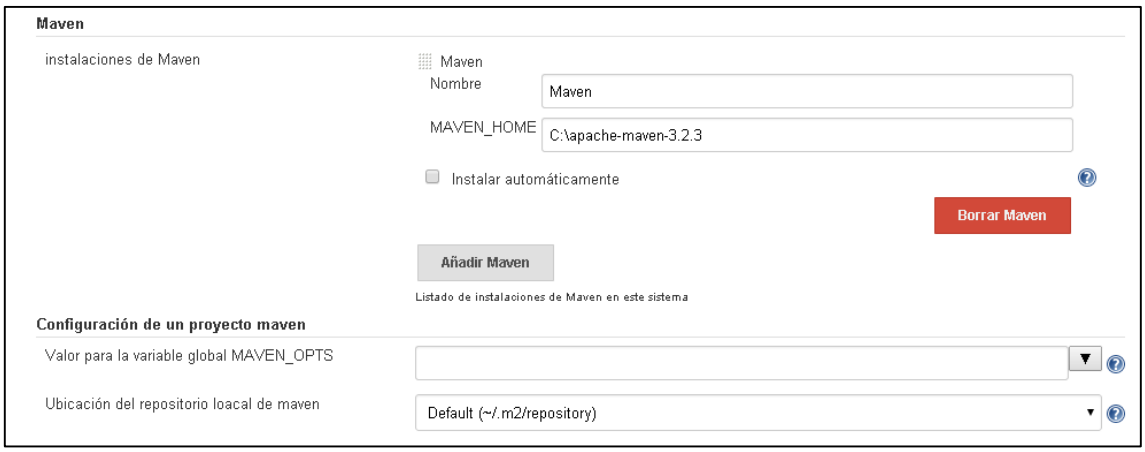
Esto es todo. Tras estos pequeños pasos ya tenemos el servidor de integración continua Jenkins instalado en nuestra máquina para poder trabajar con él.

Ejercicio 3: Configurar adecuadamente Jenkins y crear una tarea que asocie Jenkins con Maven para compilar el código de nuestro proyecto.

El primer paso que vamos a dar es configurar Jenkins pulsando en la opción "Administrar Jenkins" y luego, "Configurar el Sistema".

Como vamos a lanzar pruebas de un proyecto que está gestionado por Maven, debemos indicarle a Jenkins los parámetros de configuración de Maven.

Jenkins tiene la capacidad de descargar Maven automáticamente, pero como nosotros ya lo tenemos instalado en nuestra máquina, solo le indicaremos la ruta donde se encuentra.

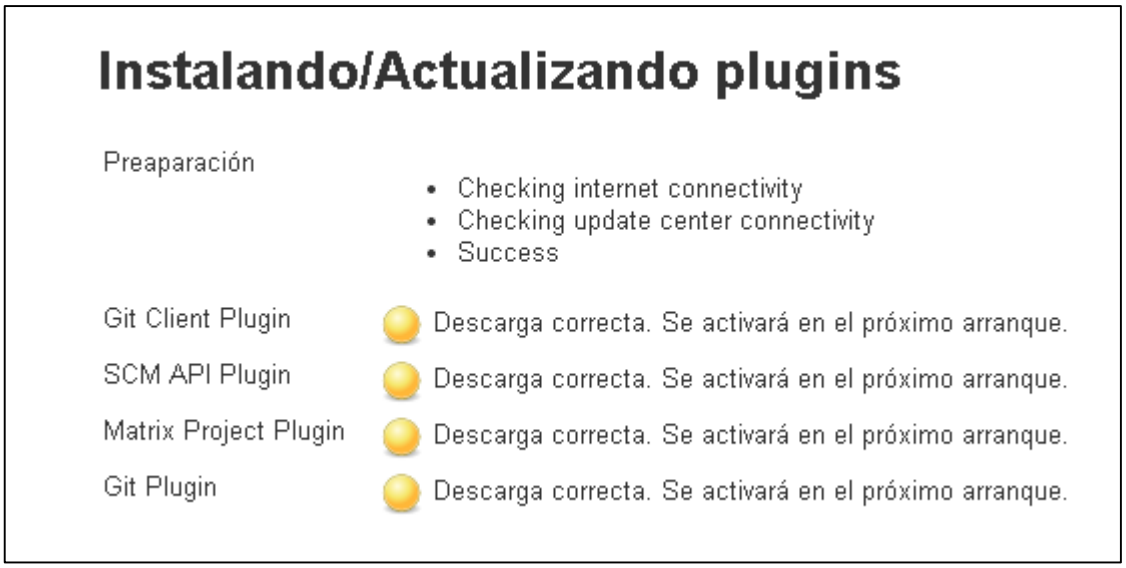


El siguiente parámetro a configurar es el JDK.

El siguiente paso a realizar es configurar Git en Jenkins, ya que Git no viene por defecto instalado. Así que a vamos a instalar el plugin de Git para Jenkins. Nos vamos a "Administrar Jenkins", sobre la opción "Administrar plugins". Una vez ahí, nos vamos a la pestaña de "Todos los plugins" y filtramos la búsqueda escribiendo en el buscador "git plugin". Seleccionamos después dicho plugin y pulsamos en "instalar sin reiniciar".

Hay gente que ha usado un plugin para "github", ¿lo habéis intentado?

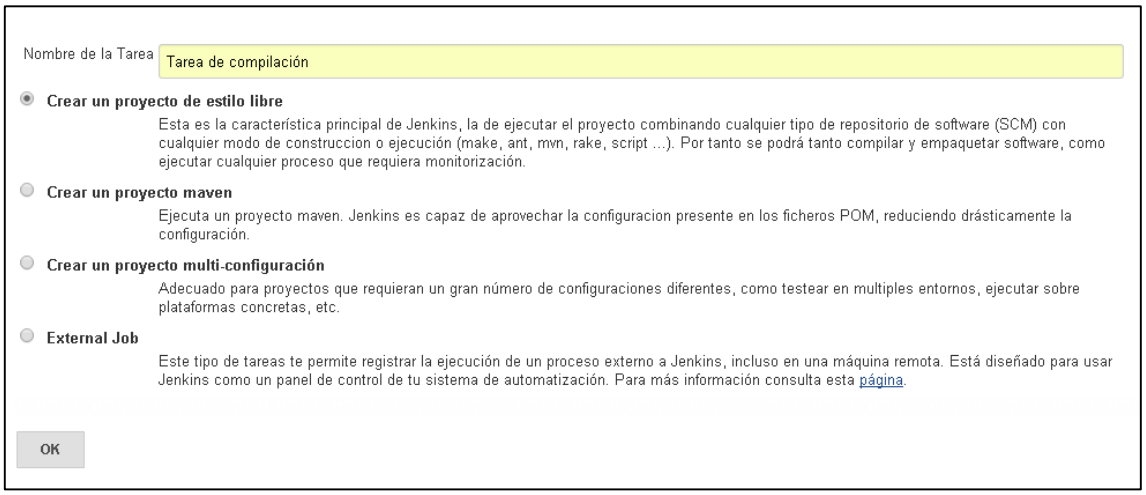
Una vez hecho lo anterior, debe aparecer la siguiente pantalla:



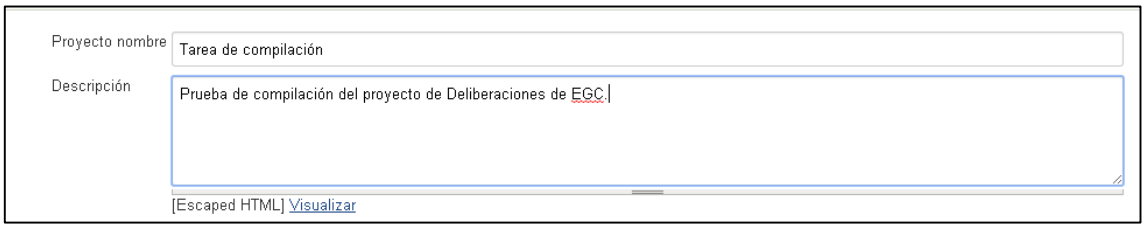
Después de terminar la instalación, reiniciamos nuestro servidor para que los cambios surtan efecto y ya podamos disponer del plugin de Git en nuestro servidor Jenkins.

El siguiente paso será el de crear al fin una tarea. Para ello nos vamos al menú "Nueva Tarea".

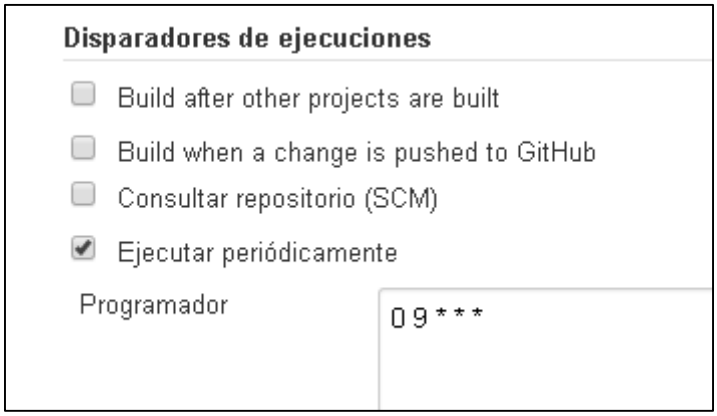
Elegimos un nombre para la tarea (en este caso se ha llamado "Tarea de compilación) y seleccionamos la opción "Crear un proyecto de estilo libre".



En la siguiente pantalla seleccionamos una descripción para nuestra tarea:

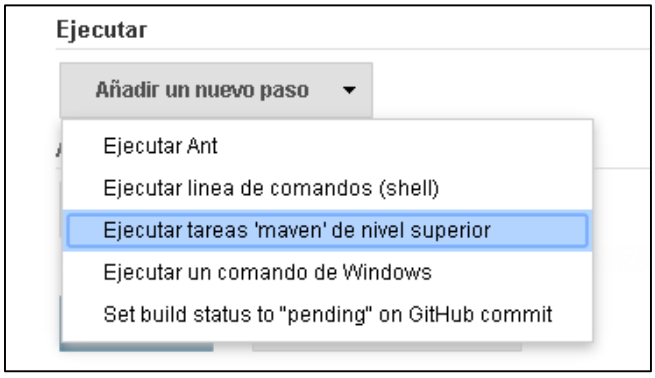


En las opciones avanzadas del proyecto, podemos indicar cada cuanto tiempo se lanzará la tarea. En este caso, vamos a ponerla a ejecutar todos los días a las 9 de la mañana. Indicamos esto mediante esta secuencia: 0 9 * * *. La forma de introducir los intervalos de tiempo tiene la sintaxis del comando "cron". Es necesario familiarizarse antes con dicha sintaxis para poder entender bien los rangos de tiempo posibles.



Ahora vamos a indicar dónde se encuentra nuestro código fuente. En este caso, se encuentra alojado en Github. Debemos seleccionar la URL del repositorio y unas credenciales de acceso.

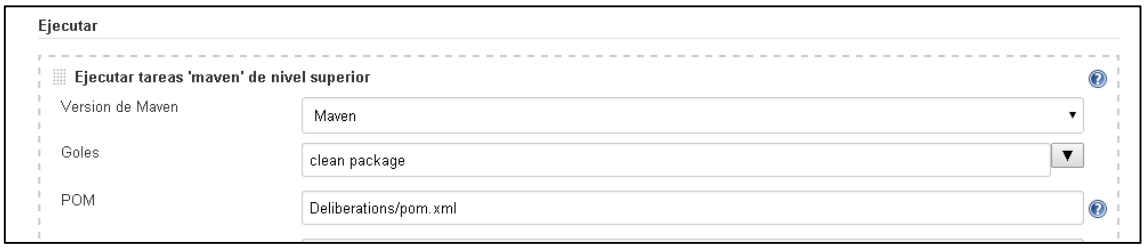
Ahora vamos a la pestaña "Ejecutar", donde vamos a indicar a Jenkins que vamos a ejecutar una tarea de Maven, por lo que seleccionamos la opción "Ejecutar tareas 'maven' de nivel superior".



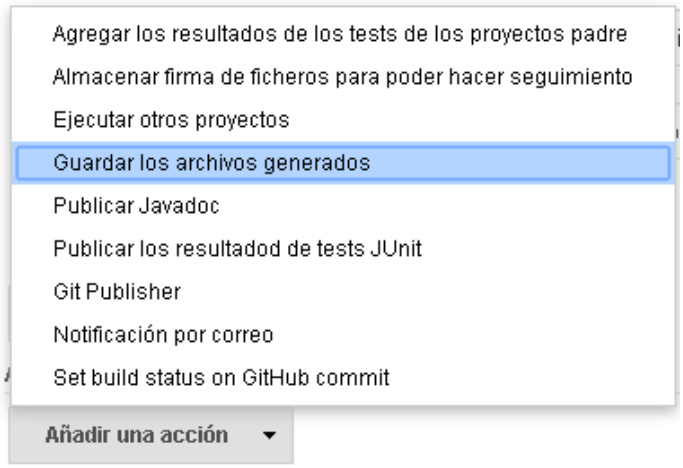
En la pestaña "goles" indicamos la opción "clean package". En la sintaxis de Maven esto significa que con "clean" vamos a borrar las compilaciones anteriores que hayamos podido realizar y con "package" se llevarán a cabo las siguientes acciones:

- Verificar que el proyecto es válido y contiene todos los elementos necesarios.
- Compilar el código.
- Si se están realizando pruebas con el framework "unit testing framework", éstas se ejecutan.
- Se compila el código y se empaqueta.

También indicamos la ruta de nuestro fichero "pom.xml". En conjunto, quedaría así:



Podemos indicar a Jenkins qué debe hacer tras llevar a cabo todo lo que se le ha pedido. En este caso, le diremos que guarde los archivos que se hayan generado de la siguiente manera:



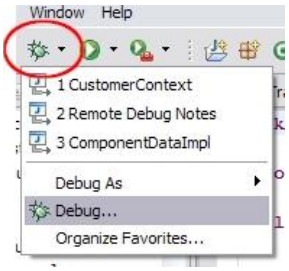
Y esto es todo. Una vez guardada la configuración solo hace falta esperar a que se ejecute la tarea y determinar si todo ha ido bien o algunos pasos han fallado, lo cual se mostrará con un círculo azul si todo ha ido bien, y un círculo rojo si se han producido fallos.

Se podría haber finalizado el ejercicio poniendo el resultado de la tarea una vez ejecutada. En general está mejor este ejercicio.

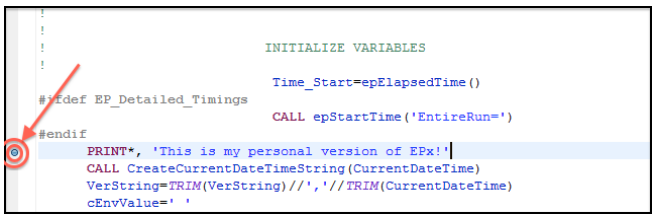
5 Gestión del cambio, incidencias y depuración.

5.1 Mecanismos de depuración.

Para la depuración del proyecto de nuestro subsistema se va a utilizar la herramienta de depuración Debug. Esta herramienta de depuración ya viene instalada por defecto en nuestro entorno de programación Eclipse. Este proceso será llevado a cabo por el equipo de desarrollo.



Primero se realizará un diagnóstico de la mejora a realizar o incidencia a corregir y luego se procederá a acotar la magnitud del problema para después intentar solucionarlo. La técnica que se utilizará será la del uso de puntos de ruptura (breakpoints).



Son mecanismos básicos de depuración. Poner una situación real de depuración hablando del proceso descrito y de qué hipótesis se han hecho, cuánto tiempo se ha tardardo en detectar el origen del error, etcétera sería más ilustrativo y mejor.

5.2 Gestión de cambios.

La gestión de los cambios se llevará a cabo a través del software de control de versiones Git. Concretamente, se utilizará la aplicación GitHub que es un sistema de control colaborativo de revisión y desarrollo de software que utiliza el sistema de control de versiones Git.

¿qué hace esta figura aquí? ¿a qué ayuda?

5.3 Procesos en la gestión de cambios.

Dada la necesidad de realizar un cambio en el proyecto, ya sea por una mejora o a una corrección en el código debido al reporte de una incidencia, los pasos a seguir son los siguientes:

¿qué quiere decir esto?

1º Configuración: descargar e instalar GitHub para el sistema operativo que se esté utilizando y crearnos una cuenta.

2º Crear un repositorio: se procederá a la creación de un repositorio nuevo que contendrá nuestro proyecto.

3º Crear un directorio: se creará un directorio nuevo al que se le anidará nuestro repositorio mediante la url que GitHub nos proporciona para poder acceder a éste.

(Los tres primeros pasos solo habría que realizarlos la primera vez).

4º Hacemos un Checkout de nuestro repositorio y así tener nuestro directorio actualizado desde el último cambio que se haya realizado en el proyecto.

5º Se procede a modificar el código del proyecto.

6º Registramos los cambios realizados en el proyecto mediante la función Add.

7º Subimos los cambios realizados a Git mediante la función Commit.

esto no viene a cuento aquí sería, si acaso, de SCM

8º Conflictos:

Un conflicto surge cuando dos o más desarrolladores acceden simultáneamente a una misma versión del repositorio y la modifican. Tras el Commit del primer desarrollador que proceda a subir cambios en el proyecto, los siguientes Commit's del resto de desarrolladores que accedieron a la misma versión del proyecto darán lugar a un conflicto. Otro de los motivos por los que puede surgir un conflicto es cuando dos usuarios modifican la misma línea, o bien cuando han modificado un fichero binario; por eso los ficheros binarios no deben estar en un repositorio. Git nos da la posibilidad de trabajar con ramas remotas, pero al fusionarlas, también se podría llegar a un conflicto. Por esto y porque el equipo de desarrollo es considerablemente de tamaño pequeño, nosotros hemos decidido trabajar solo con una única rama, la rama Master.

La solución de un conflicto sería gestionada ya fuera de GitHub, teniendo que ponerse en contacto los partícipes y tendrían que llegar a un acuerdo sobre qué hacer con el/los archivos afectados en dicho conflicto.

En el siguiente enlace se proporciona un tutorial para una configuración y uso de GitHub a más bajo nivel que el explicado anteriormente:

http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=githubFirstStepsUploadProject

No se dice nada de lo que realmente debería hablarse aquí. Estos no son "procesos en la gestión del cambio"

5.4 Roles en la gestión de las incidencias.

-Usuario: usuario final que usa la aplicación. Si un usuario percibe una posible incidencia podrá reportarla y así informar al equipo de desarrollo de la misma. Para ello, este usuario deberá enviar un correo electrónico a la dirección deliberaciones@agoravoting.com con un breve comentario informando de la incidencia junto con la máxima información posible acerca del entorno software en el que se encuentre. La última información sobre el entorno software (sistema operativo, versión del

No es un sistema realmente profesional.

navegador, etc.) no se le exigirá estrictamente para evitar una pérdida de tiempo que le haga cambiar de opinión en su decisión de querer reportar dicha incidencia.

-Equipo de control: persona o grupo de personas pertenecientes al proyecto que se encargará/n de filtrar las incidencias reportadas por los usuarios antes de que estas lleguen al equipo de desarrollo, y posteriormente, revisarán los cambios realizados.

Cuando un usuario reporta una incidencia, primeramente, el equipo de control contestará al usuario con un e-mail estándar agradeciéndole y confirmándole la recepción de su correo de incidencia y que esta será revisada. A continuación estudiará si dicha incidencia se cumple realmente. En caso de que el reporte de incidencia no fuera real, acabaría aquí el proceso, pero en caso de que sí, se generaría un informe lo más explícito posible y ampliando en él la información proporcionada por el usuario y se le haría llegar al equipo de desarrollo.

-Equipo de desarrollo: persona o grupo de personas pertenecientes al proyecto encargada/s de solucionar (entre otras cosas) las incidencias reportadas por un usuario una vez que ésta ha pasado por el equipo de control y éste ha generado un informe de la misma para llevar a cabo su resolución.

5.5 Estados de los cambios.

El estado de un cambio tendrá tres tipos de niveles:

- Pendiente: este nivel de estado recogerá aquellas solicitudes de cambio que están a la espera de empezar a ejecutarse.
- En ejecución: este nivel de estado recogerá aquellos cambios que se están llevando a cabo en ese momento.
- Finalizado: este nivel de estado recogerá aquellos cambios que ya se han llevado a cabo.

Gestionar estos estados sin una herramienta es bastante tedioso en cuánto haya algunas incidencias y el equipo sea grande.

5.6 Políticas usadas para descartar, fomentar o retardar cambios.

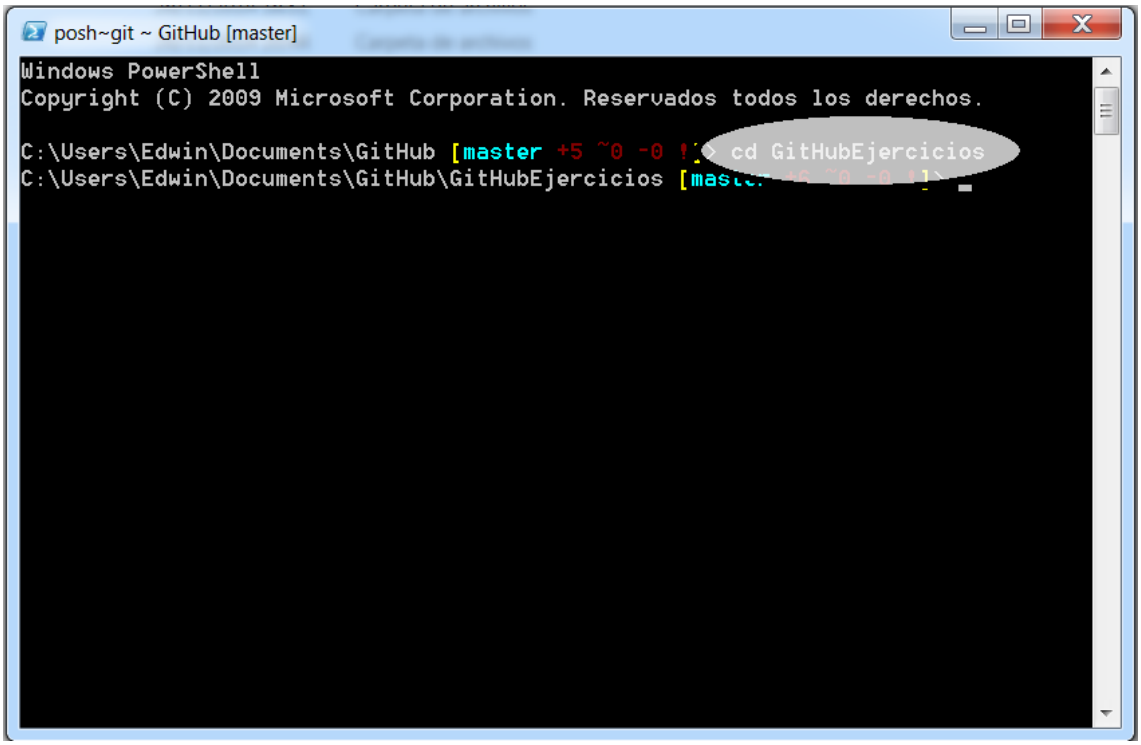
La política usada para descartar, fomentar o retardar cambios podrá llevarla a cabo tanto el equipo de control como el equipo de desarrollo. Eso sí, todo descarte, fomento o retardo de un cambio deberá ser consensuado y compartido entre ambos equipos antes de aplicarse.

5.7 Ejercicios.

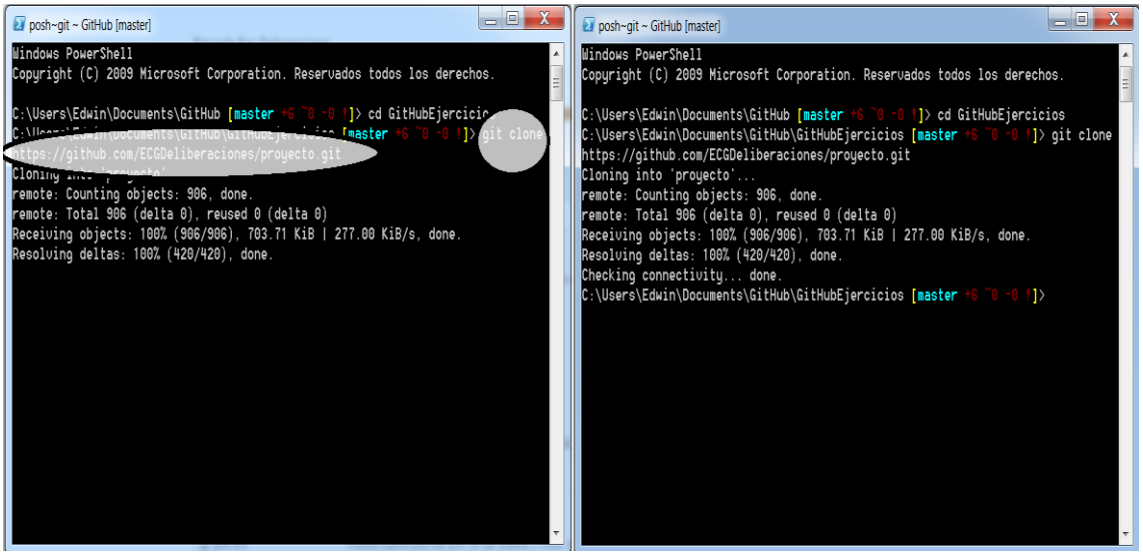
Este ejercicio no es procedente en este capítulo del proyecto

Ejercicio 1: Clonar el repositorio o hacer un "Checkout".

1º Arrancamos Git Shell y accedemos al directorio en el cual vamos a trabajar.



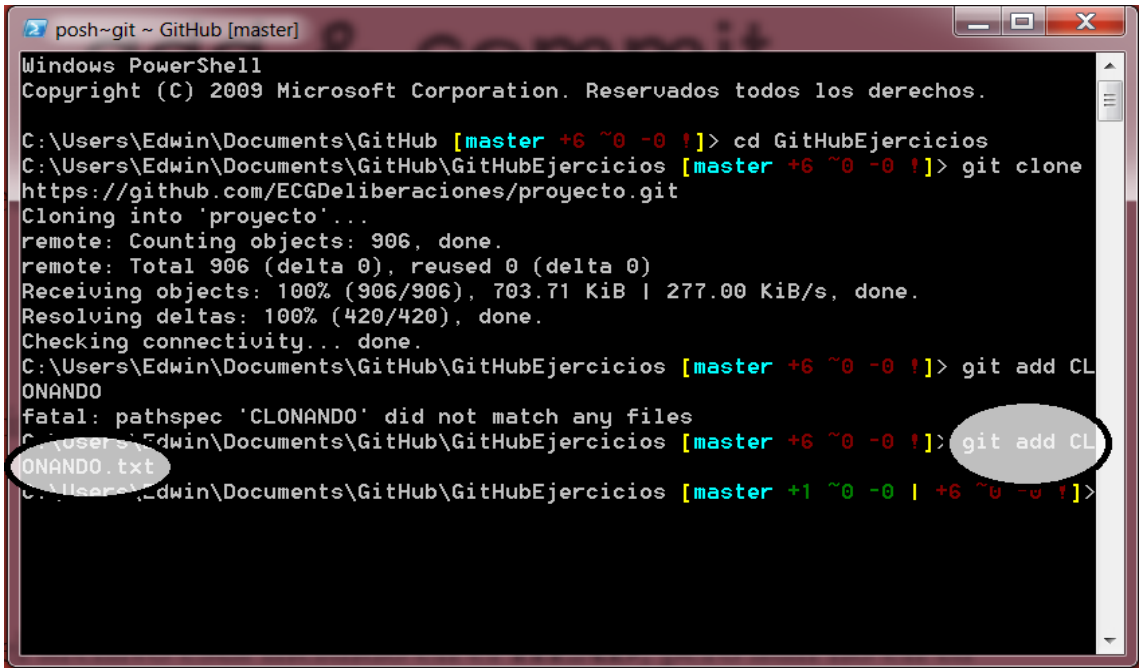
2º Ejecutamos el código: git clone "URL del repositorio que queremos clonar".



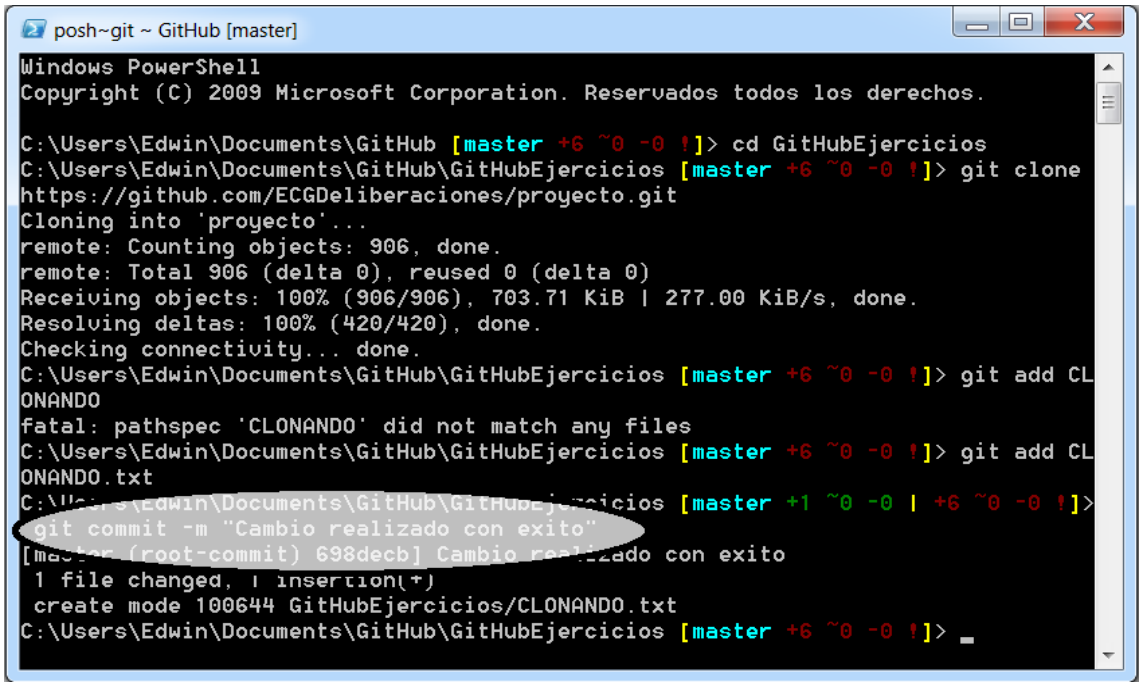
Ejercicio 2: Add y Commit:

1º Modificamos un fichero de nuestro directorio de trabajo y guardamos los cambios.

2º En Git Shell ejecutamos el comando: git add "nombre del fichero.extensión".

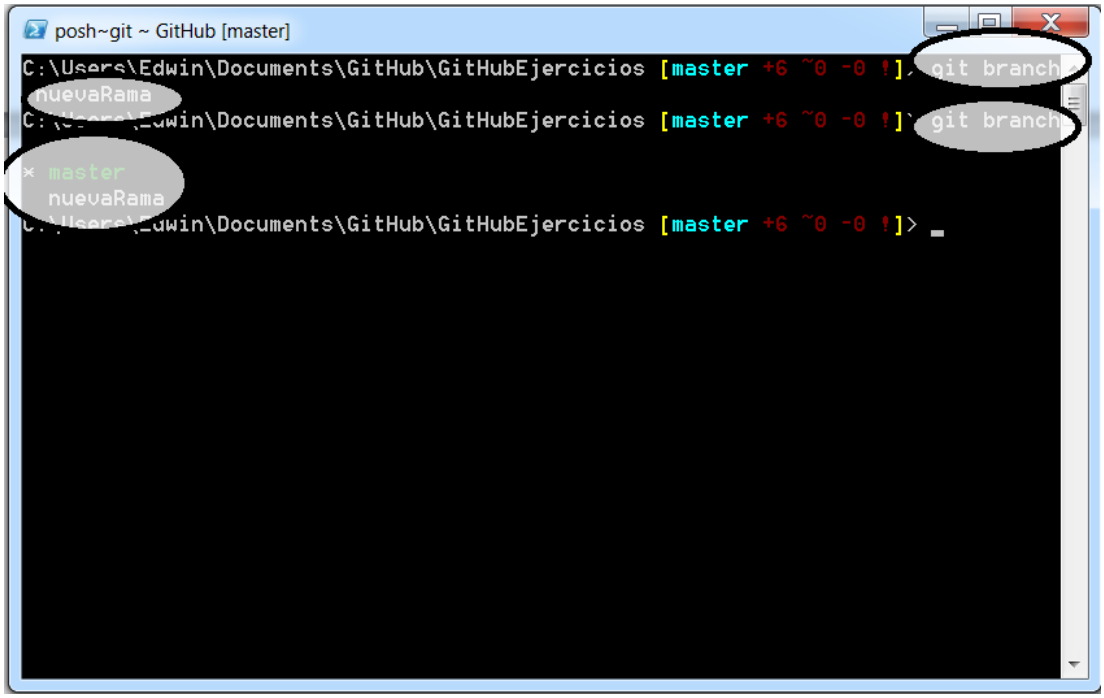


3º Realizamos el Commit de nuestro cambio mediante el código: git commit –m "comentario opcional".

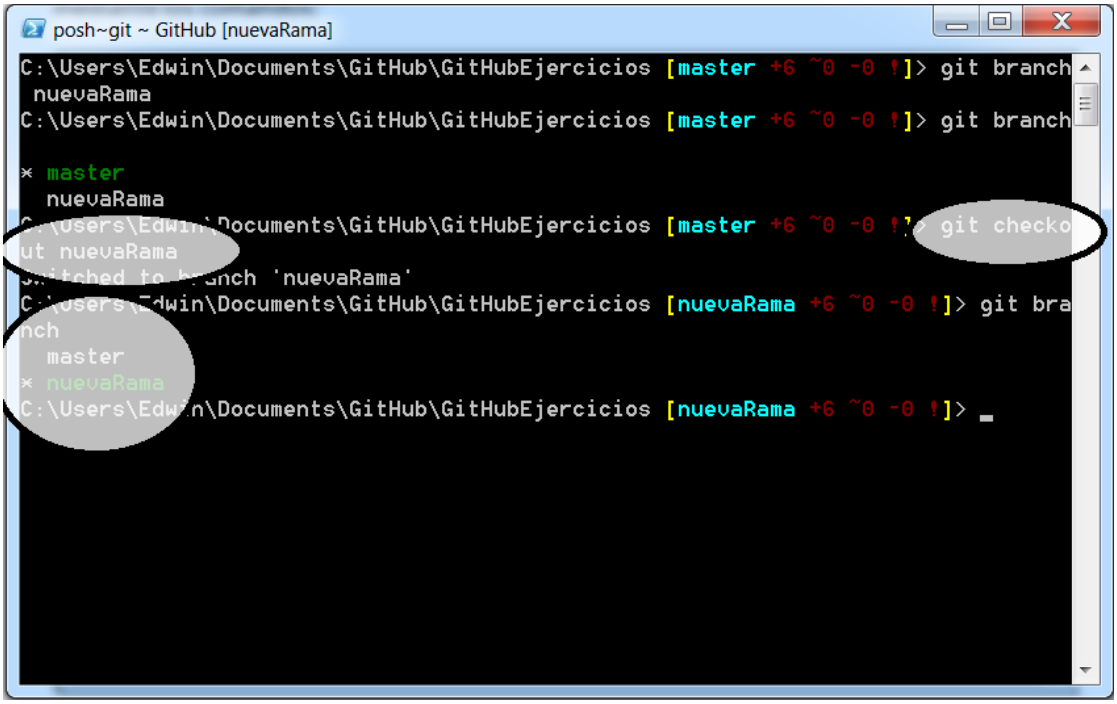


Ejercicio 3: Crear y cambiar de rama.

1º Creamos una nueva rama y posteriormente mostramos las ramas que tenemos actualmente.



2º Una vez creada la rama deseada, cambiamos de rama mediante el comando: git checkout "nombre de la rama a cambiar".



Ejercicio 4: Resolver un conflicto:

1º Clonamos el repositorio:

>> git clone URL

2º Nos colocamos en la carpeta en la cual vamos a modificar los ficheros:

>> cd nombreDeLaCarpeta

3º Creamos una nueva rama

>> git branch nombreDeLaNuevaRama

4º Modificamos el fichero en la rama master y ejecutamos los siguientes códigos:

>> git add "nombre del fichero modificado.extensión"

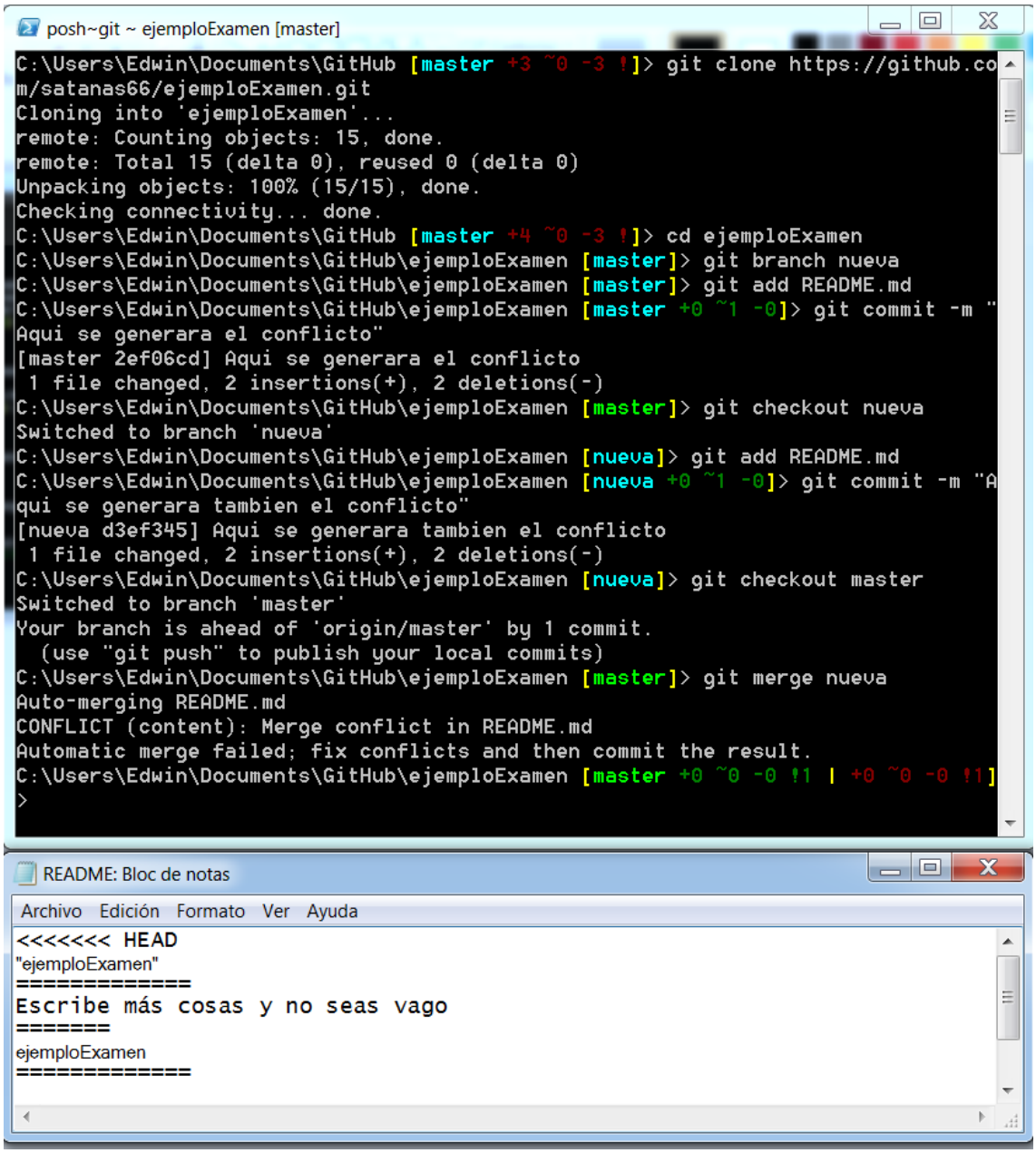
>> git commit –m "Comentarios"

5º Cambiamos de rama.

>> git checkout nombreDeLaNuevaRama

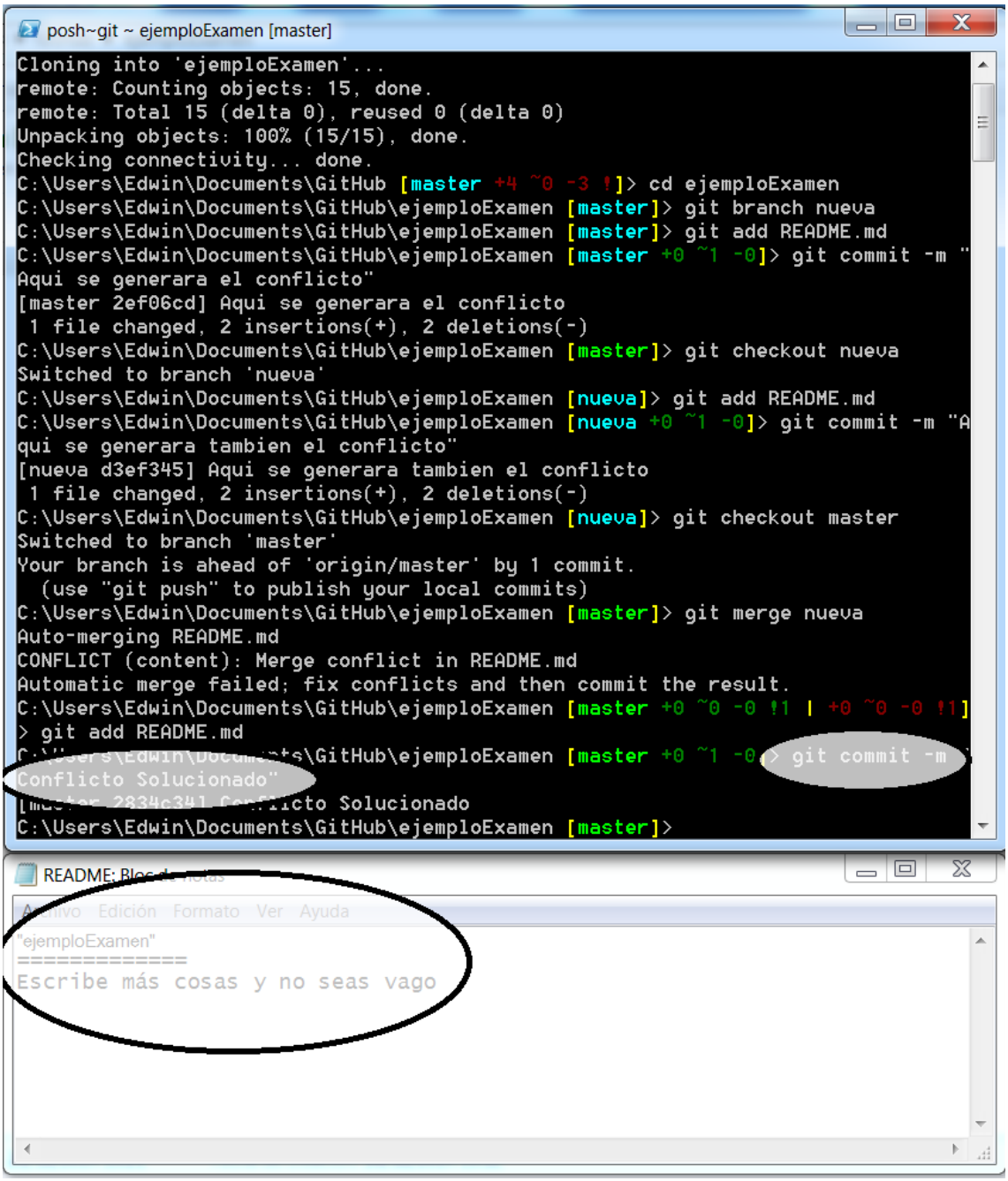
6º Repetimos el paso 4º.

7º Repetimos el paso 5 pero con la rama master.



8º Unimos las ramas git merge nombreDeLaNuevaRama

9º Se nos genera el conflicto y debemos tomar una decisión de con que modificación nos quedamos.



10º Una vez tomada la decisión repetimos el paso 4º.

El ejercicio se podría subir a la parte de gestión del código fuente.

6 Gestión de liberaciones, despliegue y entregas.

6.1 Elementos entregables del proyecto.

Los elementos entregables del proyecto del subsistema Deliberaciones son de tres tipos constituidos por la máquina virtual desarrollada para nuestro proyecto, el documento final correspondiente al proyecto que hemos realizado y un diario de grupo para reflejar todas las decisiones importantes tomadas a lo largo del proyecto y todo el trabajo realizado por cada miembro sobre el mismo.

6.2 Generar nuevos los entregables.

Para generar los entregables tenemos 3 métodos, uno por cada entregable de nuestro sistema.

- En el primero se desarrolla la generación de las distintas partes del código necesario para nuestra máquina virtual.
- El segundo de ellos se corresponde a la realización de una serie de documentos que constituirán el documento final necesario para el proyecto.
- Por último un diario de grupo que iremos rellenando a lo largo de la elaboración de nuestro proyecto y que será cumplimentado por todos los miembros del grupo.

6.3 Identificación de entregables.

Para la identificación de los diferentes entregables de nuestro proyecto vamos a definir una serie de pautas que explicamos a continuación:

Se realizará una numeración de los diferentes entregables para indicar a que versión corresponde el entregable y distinguirlo de futuras versiones, para ello definimos una norma para que todos los entregables sigan una misma estructura.

De modo que tendríamos por ejemplo DocumentoFinalXXX donde xxx correspondería a la versión en cuestión realizada. Del mismo modo identificaríamos las versiones del diario de grupo y de la máquina virtual a entregar.

6.4 Publicación, liberación y entrega de entregables.

Para la publicación de los entregables seguimos lo que tenemos a continuación:

En primer lugar se publicará un documento final en el que se van a desarrollar todos los aspectos de nuestro proyecto.

En segundo lugar publicaremos una máquina virtual que contendrá todos los elementos necesarios para el desarrollo de nuestro proyecto además de las herramientas necesarias para su funcionalidad.

Por último publicaremos un diario de grupo en el que quedará reflejado todo lo decidido en el proyecto a lo largo de su realización.

Para la liberación de los diferentes entregables del proyecto se acordarán una serie de fechas en las que se darán las diferentes versiones que se vayan desarrollando de cada uno de los 3 tipos de entregables que aparecen en nuestro proyecto.

Para la entrega de dichos entregables se acordará una fecha en la que se entregarán los distintos entregables que constituyen el proyecto desarrollado por el grupo Deliberaciones. Para dicha entrega se usará el portal Opera.

6.5 Roles en la gestión de entregas.

Para la gestión de entregables usamos los mismos roles que para la gestión del código fuente. Los desarrolladores de código que son los miembros del equipo que se encargan de generar todo el código necesario para el proyecto y desarrollar toda la funcionalidad requerida, una vez han terminado su labor se comunican con los supervisores de código, que son los encargados de verificar que todo el trabajo realizado por los desarrolladores cumple con el objetivo que se busca. Por último los supervisores de ramas para dirigir el control y la gestión sobre las nuevas ramas.

6.6 Mecanismos de despliegue definidos.

Crearemos en primer lugar un artefacto war, que será un archivo con todas las imágenes, clases, archivos de configuración del sistema etc. Una vez esto se haya producido procedemos a desplegar en un servidor el artefacto, este despliegue se va a realizar sobre un servidor de Tomcat, para que el sistema se pueda dirigir.

6.7 Procesos en la gestión de entregas.

Los procesos que se seguirán serán distintos dependiendo del tipo de entregable que se esté tratando.

Si se produce un cambio en un entregable de código los desarrolladores de código se deben poner en contacto con los supervisores para que el cambio quede realizado e indicado correctamente. Si el cambio se realiza en un entregable de documentación se debe comunicar al director del proyecto el cambio en cuestión y quedar de este modo indicado.

6.8 Plataformas en la gestión de entregas.

Las plataformas para la entrega de los entregables serán acordadas entre el equipo de trabajo del subsistema y todos los demás subsistemas que forman el sistema Agora@US. Se ha acordado que la plataforma para realizar la entrega sea el portal Opera.

6.9 Herramientas en la gestión de entregas.

Hemos usado Meld en el diario de grupo para saber que algo ha sido modificado por algún miembro del equipo.

Para hacer la documentación usamos Microsoft Office 2010 y Adore Reader.

Para el código de la máquina virtual hemos utilizado Eclipse Indigo y Virtual Box para la máquina virtual.

Para la gestión y control de versiones hemos usado un repositorio creado en GitHub que indicamos en con esta URL: https://github.com/ECGDeliberaciones/proyecto.git, además hemos hecho uso de la plataforma Dropbox en la cual tenemos todo el material que hemos desarrollado para nuestro sistema, aquí dejamos la dirección para acceder a el: https://www.dropbox.com/home/EGC.

Esto de las entregas está estructurado pero el contenido es pobre

6.10 Gestión de despliegue del proyecto completo.

Una vez que todos los subsistemas han finalizado su código y se ha probado la integración con todos en distintos entornos de desarrollo, al no encontrar ningún fallo, se pasa a realizar el despliegue. Esto consiste en poner las aplicaciones en un entorno, no de desarrollo, sino con lo justo para que se pueda consumir, es decir, los servidores para que puedan correr las aplicaciones.

El despliegue de todos los subsistemas que forman Agora@US se hará sobre una maquina Debian 7. Para que el despliegue general se realice sin más problemas debemos tener instalados un conjunto de programas:

- Git
- Maven
- Tomcat 7
- Java openjdk7
- Mysqlserver
- Mysqlclient
- Python
- Xampp (debemos cambiar el puerto de mysql para evitar problemas)

La máquina Debian 7 para el entorno del proyecto completo ha sido realiza por parte de tres de los subgrupos de clase que han sido Creación/Administración de censos, Deliberaciones (nuestro subgrupo) y Creación/administración de votaciones.

Se parte de la base de que todo el código está en el repositorio compartido de Github y se pondrán distinguir entre tres tipos de proyectos según su tecnología:

- Proyectos Java (creación/administración de votaciones, deliberaciones, recuento, creación/administración de censos, almacenamiento)
- Proyectos Django (cabina de votación)
- Proyectos PHP (auth, frontend de resultados, visualización de resultados)

En el caso de los proyectos java que usen el framework spring + maven se debe crear la base de datos según el nombre usado en el proyecto y para su construcción, se hará uso de maven y Tomcat. Del primero, se utiliza la directiva mvn clean install (que limpia archivos temporales e inicializa el contexto de spring para su funcionamiento), que es necesario para crear los war (Web Application Archive).

Este directiva hay que realizarlo dentro de cada una de las ramas, donde su ubica el fichero pom.xml que es el que contiene todas las dependencias del proyecto. Una vez generados los war, debemos desplegarlos con Tomcat para que estén accesibles desde una URL, en concreto, copiar el war a var/lib/webapps/tomcat7

Para los proyectos PHP, por ejemplo auth, se copia a /opt/lampp/htdocs/auth sus ficheros, creamos la base de datos egcdb y su tabla con el script proporcionado. Una vez realizado, se debe crear un usuario "usuario" con password "password"

Otros proyectos, como por ejemplo result_view, al no tener una base de datos, solo tendríamos que copiar su código a /opt/lamp/htdocs/ y al arrancar el Tomcat, ya podríamos acceder.

El proyecto Python es el de cabina de votación. Para este proyecto, se debe crear una estructura de carpetas como la siguiente:

- Carpeta raíz: cabina-integración
- cabina-agora-us: dentro la raíz (será la que contenga el código del subsistema)
- Dos scripts (dentro de la raíz): install.sh y run.sh

La primera vez que ejecutamos el proyecto de cabina, debemos ejecutar el install.sh (instala paquetes necesarios) y después run.sh (ejecuta el proyecto). Las veces siguientes que arranquemos este proyecto, solo será necesario ejecutar el segundo script.

Almacenamiento no corresponde a ningún grupo de los proyectos descritos a integrar debido a que es un subsistema externo, no sería necesaria su integración en la maquina Debian.

Otro caso de proyecto que no encaja en ninguna categoría, pese a ser un proyecto Java, es verificación puesto que este lo utilizan otros subsistemas para la encriptación de votos, por lo que este grupo genera una librería, la cual es consumida por otros subsistemas.

Lo más cómodo seria instalar esta librería en el repositorio maven oficial y que cada proyecto que la necesite añada las dependencias a esta, por lo que ya no tendríamos que preocuparnos de las rutas de las librerías. Ya que esto no es viable, ya que no es una librería oficial de algún proyecto estable, no está disponibles en los repositorios de apache, por lo tanto lo mejor es que cada subsistema que lo necesite, la incluya en su build path.

Cuando todo se haya integrado, tras realizar unas últimas pruebas de ejecución, por ejemplo con JMeter, el sistema Agora@US estaría listo para ser consumido por usuarios.

6.11 Ejercicios.

<u>Ejercicio 1:</u> Identificar, generar, publicar y entrega de entregables.

Los principales entregables son el código y la memoria del subsistema.

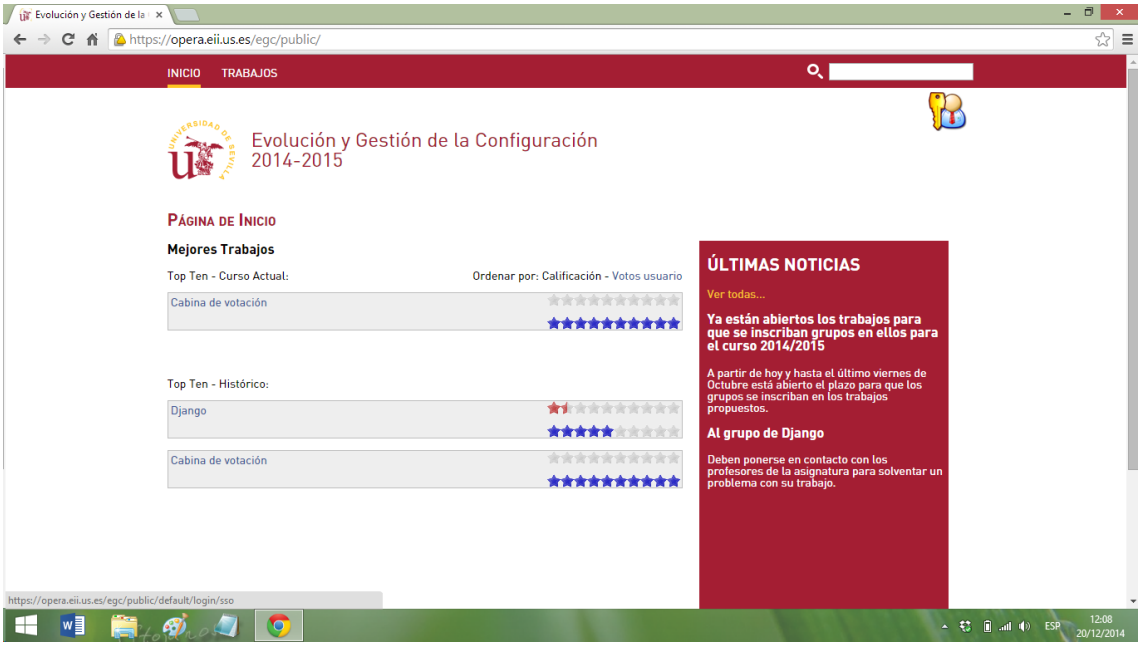
El código es generado mediante la aprobación e inserción en los diferentes repositorios git de los gestores de la configuración y los objetos generados a partir de la compilación del código los cuales siguen el mismo versionado que el código.

<u>Ejercicio 2:</u> Identificar los roles en los procesos de gestión de entregas.

Las entregas son realizadas por los gestores de la configuración del proyecto. La publicación de estos entregables debe ser aprobada por el Jefe del proyecto para su posterior publicación.

<u>Ejercicios 3:</u> Realizar subida de entregable.

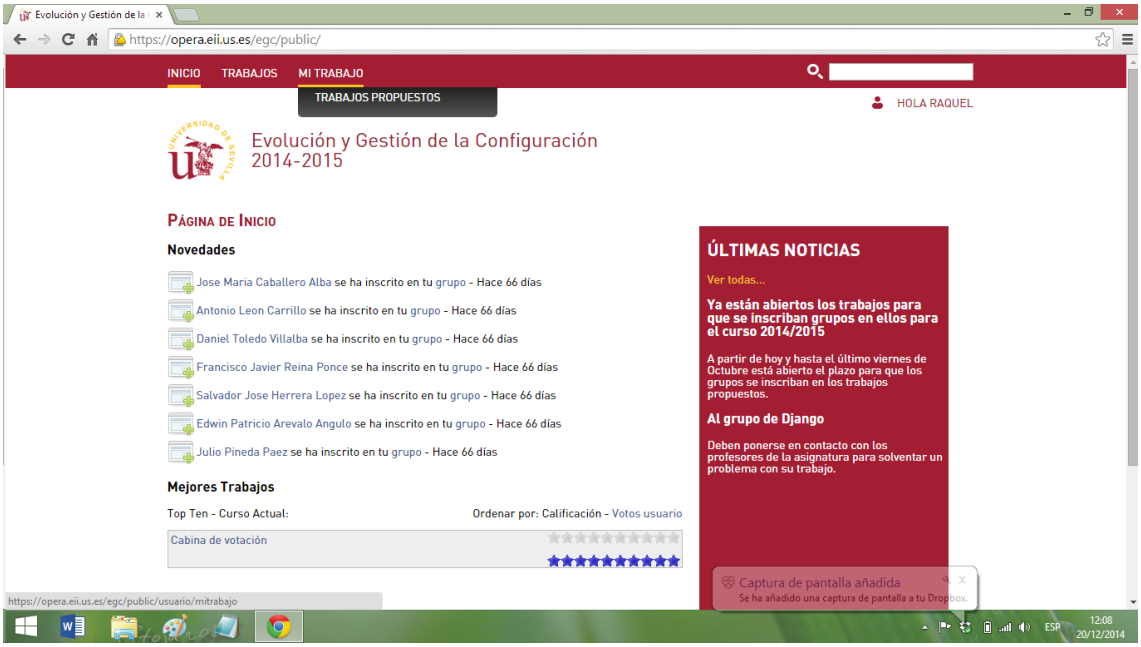
1º Ingresamos en la página https://opera.eii.us.es/egc/ donde debemos subir nuestro entregable.



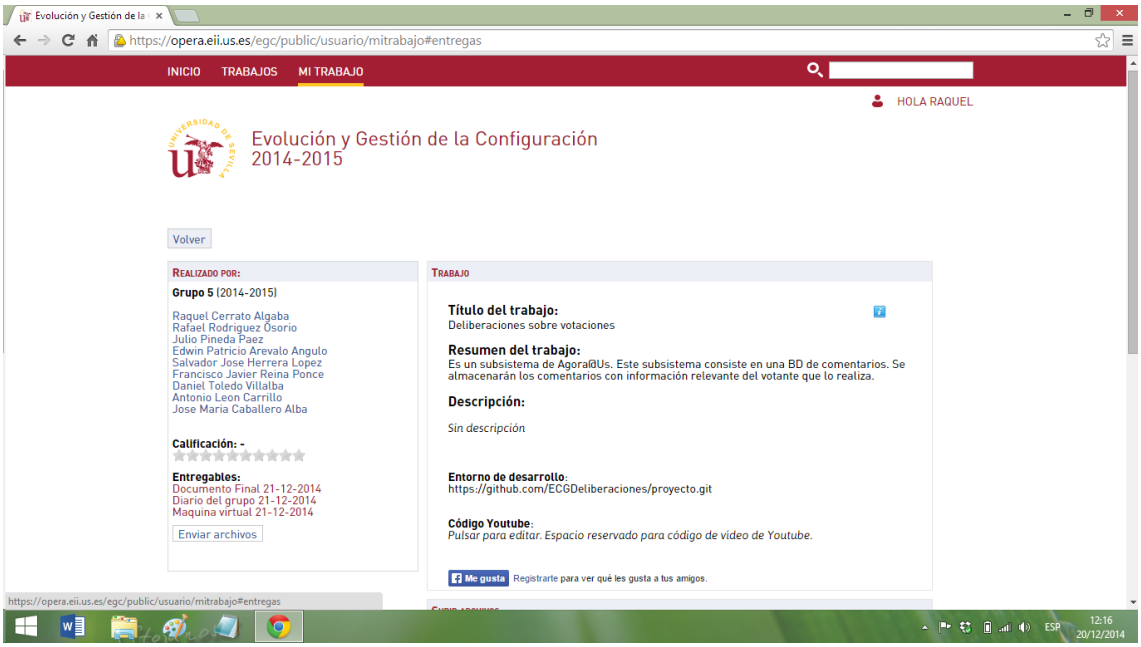
2º Accedemos a ella mediante nuestro usuario de la Universidad de Sevilla

todo esto sobra, no aporta nada...

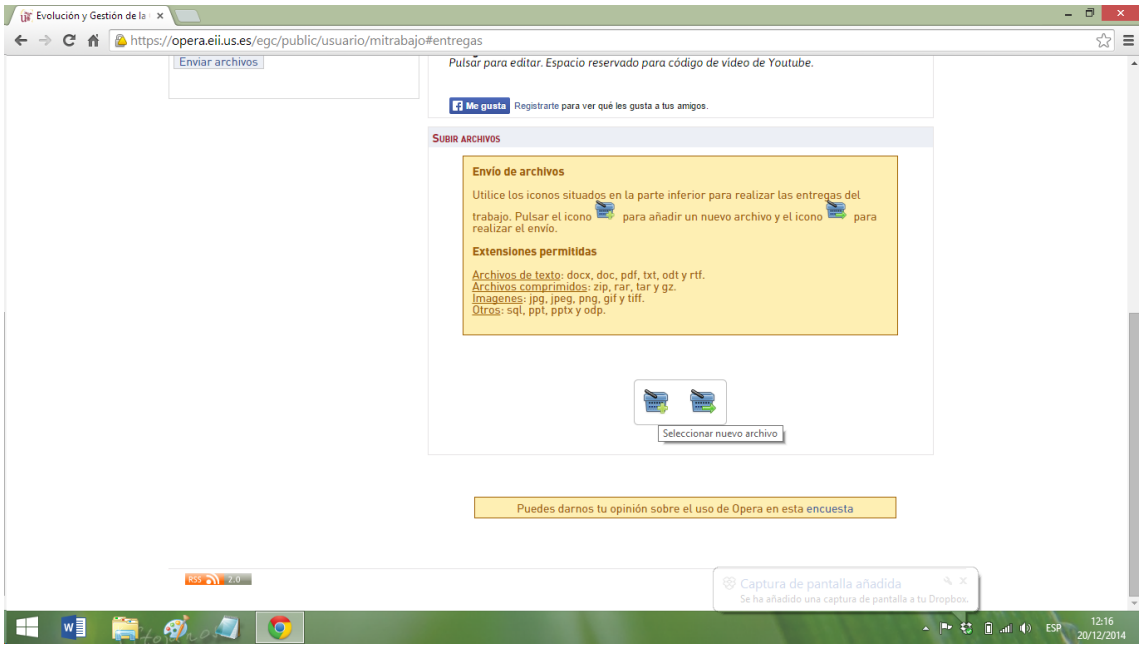
3º A continuación nos dirigimos a la sección de nuestro grupo de EGC.



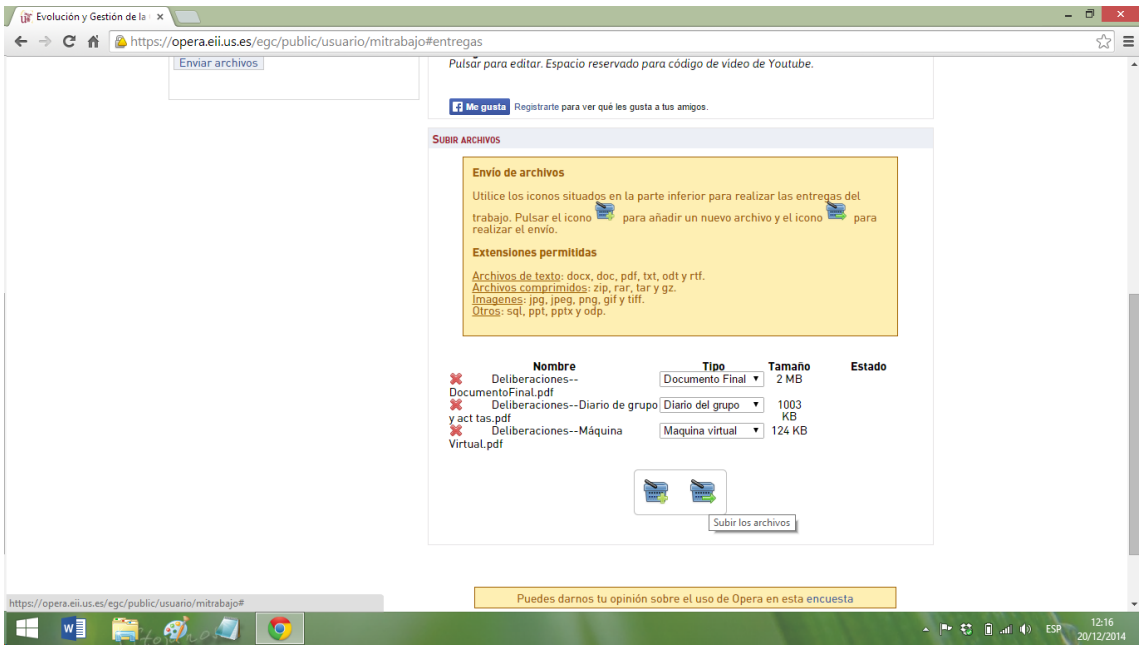
4º Luego accedemos a nuestro trabajo en la asignatura. En esta área podemos distinguir varios apartados como, los componentes del grupo, detalles del trabajo y la posibilidad de subir nuestro proyecto.



5º Seguido, accedemos a nuestro sistema mediante el link.



6º Seleccionamos para subir, nuestra Memoria, el Diario de Grupo y nuestra Máquina Virtual.



7 Gestión de la variabilidad.

En el proyecto de gestión de código en general no se utilizan ninguno de los principios de gestión de la variabilidad impartidos en clase. Para gestionarlo, utilizamos GIT que es un sistema de control de versiones mediante GitHub (www.github.com). El programa, una vez instalado (Git), está completo y no se da elección a alterar el contenido en su instalación.

Una posible aproximación a la gestión de la variabilidad podría ser la opción que nos permite GitHub, en el propio repositorio, de incluir o no ciertas "extensiones" por así llamarlas, como podrían ser, por ejemplo, añadir una wiki al repositorio, asignar tareas a los usuarios por medio de "Issues", añadir etiquetas o hitos (Milestone).

este apartado se puede borrar. No tiene nada que ver con lo que es "gestión de la variabilidad"

7.1 Ejercicios.

Para poder hacer el ejercicio de variabilidad, hemos optado por hacerlo tomando como referencia GitHub. Hemos considerado marcar las posibles opciones como *, dentro del propio repositorio, ya que dentro de él tenemos la opción de añadirlas al no venir configuradas por defecto. A continuación pasamos a explicar cada una de estas opciones y posteriormente veremos el gráfico.

Pull Request.

Es un método para la presentación de contribuciones a un proyecto de desarrollo abierto. A menudo es la forma preferida de presentación de contribuciones a un proyecto mediante un sistema de control de versiones distribuido (DVCS) como Git. La Pull Request ocurre cuando un desarrollador pide cambios comprometidos con un repositorio externo y pide su inclusión en el repositorio principal de un proyecto.

Tareas (Issues):

Permite asignar tareas a los miembros que contribuyan con ese repositorio y/o proyecto. Puede servir para organizar la gestión de dicho proyecto.

Wiki:

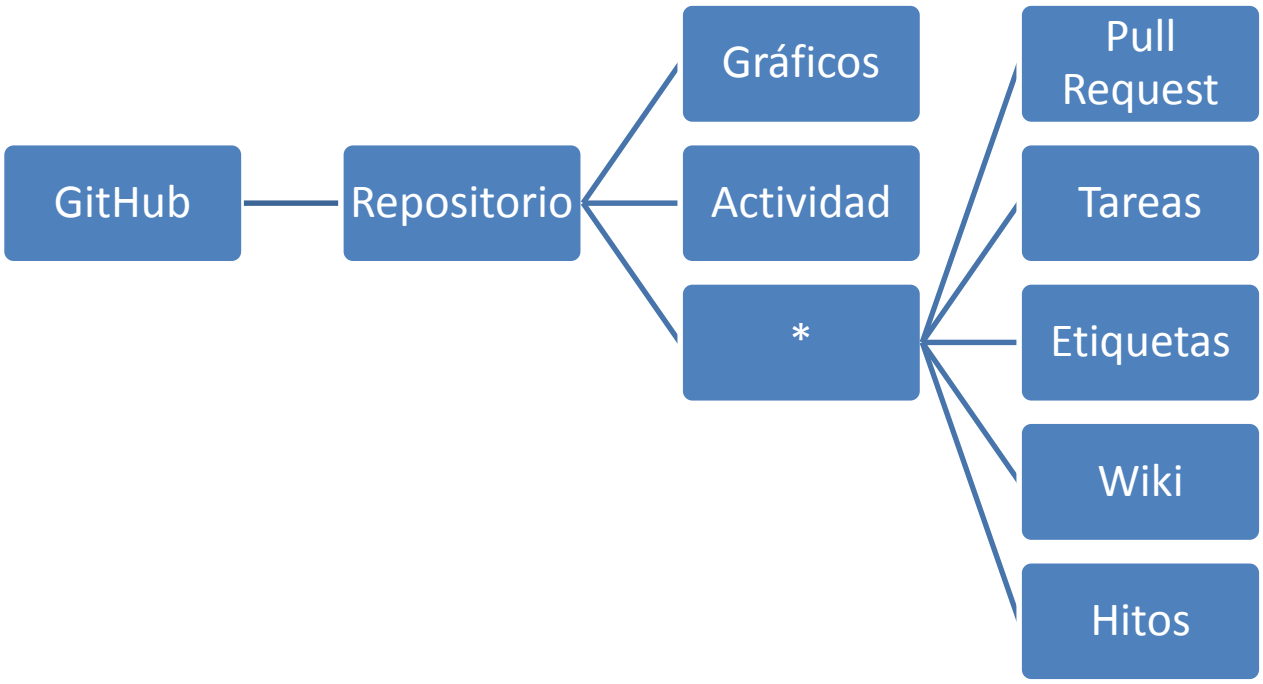
Sitio web cuyas páginas pueden ser editadas directamente donde los usuarios crean, modifican o eliminan contenidos que, generalmente, comparten.

Etiquetas (Labels):

Se pueden asignar etiquetas a ciertas partes del proyecto para acceder de forma posteriormente de forma rápida y fácil.

Hitos (Milestone):

Son grupos de temas que corresponden a un proyecto, característica, o periodo de tiempo. Se usan de diferentes maneras en el desarrollo de software.



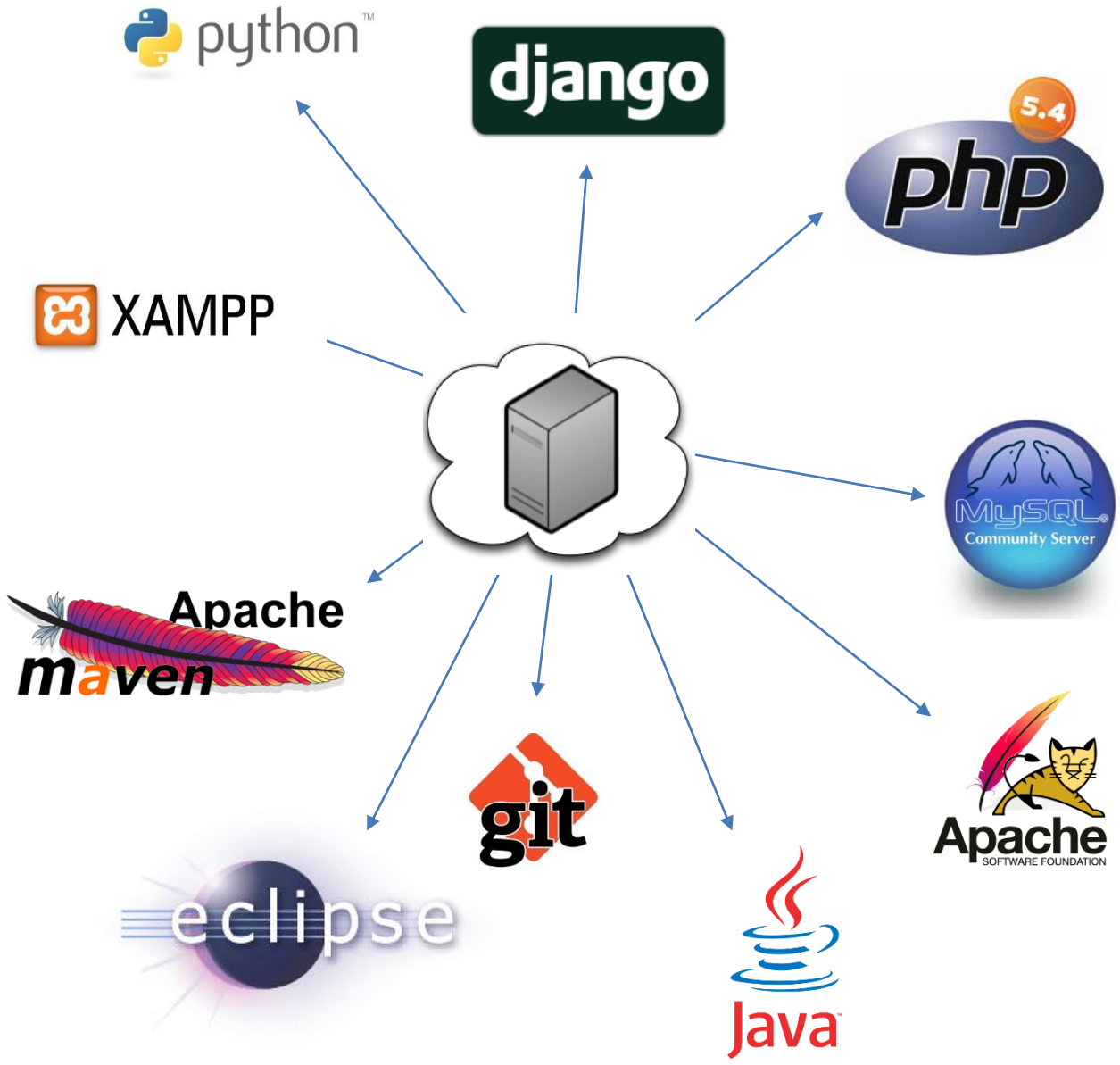
8 Mapa de herramientas.

Para el desarrollo de nuestro subsistema usaremos una máquina virtual con todos los programas necesarios para la realización del código y las pruebas del subsistema a desarrollar. Las herramientas que contiene dicha máquina virtual son los que se muestran a continuación:

Se puede obtener nuestra máquina virtual en el siguiente enlace:

https://www.dropbox.com/sh/hq4ch8q1ux4pac8/AABYvfSEMOPWaMFb3GSDHLtBa?dl=0

El mapa de herramientas utilizado para desarrollar los diferentes subsistemas con los cuales estamos relacionados en el sistema central de Ágora Voting es:



9 Conclusiones.

Tras analizar y realizar la gestión de la configuración completa del subsistema elegido Deliberaciones y la integración con el resto de subsistemas del proyecto Agora@US, podemos concluir que es difícil manejar todos los aspectos técnicos de la gestión de proyectos, pero tras enfrentarnos a ellos e ir aprendiendo poco a poco como usarlas de manera eficaz para el desarrollo de nuestro proyecto.

Respecto a las herramientas utilizadas, hemos comprendido que el uso de repositorios de control de versiones es muy útil sobre todo por poder integrar herramientas que nos facilitan la realización del proyecto, además de su utilidad de poder compartir y gestionar el código.

Finalmente, podemos decir que estamos satisfechos con el proyecto desarrollado por nuestra parte y por parte del grupo completo al integrar todos los subsistemas de Agora@US y tener la lección aprendida de que gestionar bien un proyecto es muy importante para tener una base fiable y comprensible de cómo desarrollarlo.

10 Bibliografía.

- https://1984.lsi.us.es/wiki-egc/index.php/2014/2015
- https://agoravoting.com/
- https://1984.lsi.us.es/wiki-egc/index.php/Comic_explicativo_del_proceso_de_cifrado_y_de_anonimizaci%C3%B3n_de_AgoraVoting
- http://oss-watch.ac.uk/resources/pullrequest
- http://es.wikipedia.org/-
- https://guides.github.com/features/issues/
- http://www.javiergarzas.com/2014/05/jenkins-en-menos-de-10-min.html
- http://www.notodocodigo.com/blog/instalacion-y-primeros-pasos-con-jenkins/
- http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=githubFirstStepsUploadProject

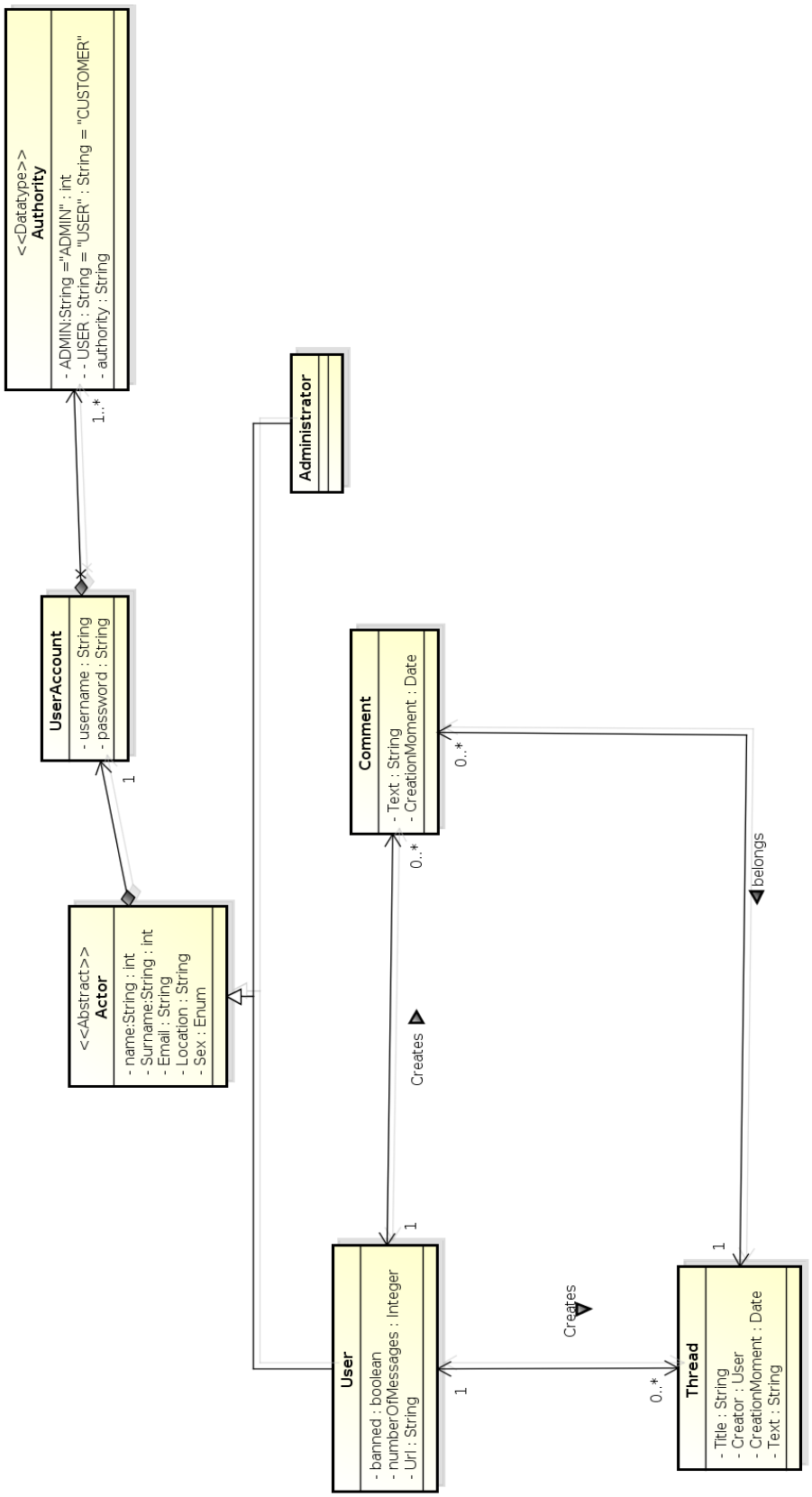
11 Anexos.

- Junto al documento de trabajo se adjuntan el diario de grupo y las actas de reunión (Deliberaciones--DiarioDelGrupo.pdf)

- Enlace para descargarla máquina virtual de desarrollo. https://www.dropbox.com/sh/hq4ch8q1ux4pac8/AABYvfSEMOPWaMFb3GSDHLtBa?dl=0

- UML

A. UML


<<Datatype>>
Authority
- ADMIN:String ="ADMIN" : int
- - USER : String = "USER" : String = "CUSTOMER"
- authority : String
1..*
UserAccount
- username : String
- password : String
1
Administrator
Comment
- Text : String
- CreationMoment : Date
0..*
0..*
belongs
<<Abstract>>
Actor
- name:String : int
- Surname:String : int
- Email : String
- Location : String
- Sex : Enum
Creates
1
User
- banned : boolean
- numberOfMessages : Integer
- Url : String
Creates
1
0..*
Thread
- Title : String
- Creator : User
- CreationMoment : Date
- Text : String
1