

Agora@US











Creación y Administración de Censos

Grupo 7

Evolución y Gestión de la configuración

2014-2015

INTEGRANTES

David Álvarez Silva	Antonio Juan Amador Salmerón	Francisco Javier Delgado Vallano	Guiomar Fdez. de Bobadilla Briosó
Jefe de proyecto	Desarrollador	Gestor de la configuración	Desarrollador
			
José Luis García Mora	Sebastián Garrocho Capacete	Javier Guisado Torres	Rafael Quesada García
Gestor de documentación	Gestor de documentación	Gestor de la configuración	Gestor de pruebas
			

HISTORIAL DE VERSIONES

Versión	Autor	Cambio	Fecha
0.1	David Álvarez Silva	Creación de la estructura del documento	28/10/2014
0.2	Javier Guisado Torres	Integración de las diferentes partes de la memoria	02/12/2014
0.3	David Álvarez Silva	Corrección de formatos	12/12/2014
0.4	TODOS	Revisión y corrección	18/12/2014
1.0	David Álvarez Silva	Envío para revisión	21/12/2014
1.1	Javier Guisado Torres Guiomar Fdez. de Bobadilla Brioso	Reestructuración del apartado Gestión de ramas y añadir nuevo apartado de migración de SVN a GitHub	23/01/2015
1.2	David Álvarez Silva Antonio Juan Amador Salmerón	Reestructuración y ampliación del apartado de Gestión de la construcción e integración continua	24/01/2015
1.3	José Luis García Mora Sebastián Garrocho Capacete	Reestructuración y ampliación del apartado de Creación y aplicación de patch, Roles y Política de nombre y estilos utilizados en el código fuente	25/01/2015
1.4	Fco. Javier Delgado Vallano Rafael Quesada García	Ampliación de los apartados Gestión de Cambios y Depuración.	04/02/2015
2.0	TODOS	Revisión y corrección del entregable para la mejora.	05/02/2015

1. INDICE

INTEGRANTES	1
HISTORIAL DE VERSIONES	2
1. INDICE.....	3
2. INDICE DE FIGURAS	6
3. RESUMEN.....	8
4. INTRODUCCIÓN.....	9
5. GESTIÓN DEL CÓDIGO FUENTE	11
5.1. GESTIÓN DE RAMAS	11
5.1.1. PROBLEMA	11
5.1.2. SOLUCIÓN PROPUESTA.....	12
5.1.3. LECCIONES APRENDIDAS	20
5.2. SVN A GIT	21
5.2.1. PROBLEMA	21
5.2.2. SOLUCIÓN PROPUESTA.....	21
5.2.3. LECCIONES APRENDIDAS	24
5.3. CREACIÓN Y APLICACIÓN DE PATCH	25
5.3.1. PROBLEMA	25
5.3.2. SOLUCIÓN PROPUESTA.....	25
5.3.3. LECCIONES APRENDIDAS	28
5.4. ROLES	29
5.4.1. PROBLEMA	29
5.4.2. SOLUCIÓN PROPUESTA	29
5.4.3. LECCIONES APRENDIDAS	30
5.5. POLÍTICA DE NOMBRE Y ESTILOS UTILIZADOS EN EL CÓDIGO FUENTE	31
5.5.1. PROBLEMA	31
5.5.2. SOLUCIÓN PROPUESTA.....	31
5.5.3. LECCIONES APRENDIDAS	33
5.6. EJERCICIO	34
6. GESTIÓN DE LA CONSTRUCCIÓN E INTEGRACIÓN CONTINUA.....	38
6.1. GESTIÓN DE LA CONSTRUCCIÓN	38
6.1.1. PROBLEMA	38
6.1.2. SOLUCIÓN PROPUESTA.....	38
6.1.3. LECCIONES APRENDIDAS	46

6.2.	INTRODUCCIÓN A LA INTEGRACIÓN.....	47
6.2.1.	INTEGRACIÓN INTERNA.....	47
6.2.1.1.	PROBLEMA	47
6.2.1.2.	SOLUCIÓN PROPUESTA.....	47
6.2.1.3.	LECCIONES APRENDIDAS	57
6.2.2.	INTEGRACIÓN EXTERNA	59
6.2.2.1.	PROBLEMA	59
6.2.2.2.	SOLUCIÓN PROPUESTA.....	59
6.2.2.3.	LECCIONES APRENDIDAS	64
6.2.3.	EJERCICIO 1.....	66
6.2.4.	EJERCICIO 2.....	73
7.	GESTIÓN DEL CAMBIO, INCIDENCIA Y DEPURACIÓN	78
7.1.	GESTIÓN DE CAMBIOS.....	78
7.1.1.	PROBLEMA	78
7.1.2.	SOLUCIÓN PROPUESTA.....	78
7.1.3.	LECCIONES APRENDIDAS	80
7.2.	GESTIÓN DE INCIDENCIAS	81
7.2.1.	PROBLEMA	81
7.2.2.	SOLUCIÓN PROPUESTA.....	81
7.2.3.	LECCIONES APRENDIDAS	91
7.2.4.	EJERCICIO 1.....	92
7.2.5.	EJERCICIO 2.....	96
7.3.	DEPURACIÓN.....	100
7.3.1.	PROBLEMA	100
7.3.2.	SOLUCIÓN PROPUESTA.....	100
7.3.3.	LECCIONES APRENDIDAS	103
8.	GESTIÓN DE DESPLIEGUE	104
8.1.	PROPUESTA DE DESPLIEGUE	104
9.	MAPAS DE HERRAMIENTAS	107
9.1.	MAPA DE HERRAMIENTAS PROPIO	107
9.1.1.	SISTEMA OPERATIVO.....	107
9.1.2.	ENTORNOS.....	107
9.1.3.	LENGUAJES	107
9.1.4.	SERVIDORES.....	108
9.1.5.	BASE DE DATOS	108
9.1.6.	REPOSITORIOS	109

9.1.7.	FRAMEWORKS.....	109
9.1.8.	HERRAMIENTAS DE ENTORNOS	110
9.1.9.	HERRAMIENTAS DE INTEGRACIÓN.....	110
9.2.	MAPA DE HERRAMIENTAS GENERAL.....	112
9.2.1.	SISTEMAS OPERATIVOS	112
9.2.2.	ENTORNOS.....	112
9.2.3.	LENGUAJES	112
9.2.4.	SERVIDORES.....	113
9.2.5.	BASE DE DATOS	113
9.2.6.	REPOSITARIOS	113
9.2.7.	FRAMEWORKS.....	114
9.2.8.	HERRAMIENTAS DE ENTORNO	115
9.2.9.	INTEGRACIÓN CONTINUA.....	115
9.2.10.	REPRESENTACIÓN GRÁFICA DE LA RELACIÓN ENTRE LAS HERRAMIENTAS	116
10.	CONCLUSIONES	117
	BIBLIOGRAFÍA	118
	GLOSARIO DE TÉRMINOS	119
	ANEXOS	122
	INSTRUCCIONES MÁQUINA VIRTUAL	122

2. INDICE DE FIGURAS

FIGURA 1: VISTA DE TODOS LOS TAGS DEL REPOSITORIO	13
FIGURA 2: VENTANA DE COMMIT.....	15
FIGURA 3: MENÚ TEAM DESPLEGADO	16
FIGURA 4: REALIZANDO UN MERGE	17
FIGURA 5: CREACIÓN DE UN NUEVO TAG.....	18
FIGURA 6: REVISIÓN DE CONFLICTO	19
FIGURA 7: COMMIT DEL CAMBIO REALIZADO.....	20
FIGURA 8: RAMAS LOCALES Y REMOTAS	23
FIGURA 9: HISTORIAL EN GITHUB	23
FIGURA 10: SITUARSE EN LA RAMA DE CAMBIOS	25
FIGURA 11: APLICAR PARCHE	26
FIGURA 12: CAMBIO EN EL CÓDIGO.....	27
FIGURA 13: FICHERO PATCH	27
FIGURA 14: EJEMPLO DE CÓDIGO FUENTE.....	32
FIGURA 15: ESTRUCTURA DEL REPOSITORIO	34
FIGURA 16: MENÚ TEAM	35
FIGURA 17: RAMA CON LA QUE REALIZAR EL MERGE	35
FIGURA 18: MENÚ TEAM.....	36
FIGURA 19: CREACIÓN DE TAG.....	37
FIGURA 20: VISTA DEL REPOSITORIO EN PROJETSII	41
FIGURA 21: CAMBIOS REALIZADOS EN EL REPOSITORIO (PROJETSII)	41
FIGURA 22: ENTORNO LOCAL DE DESARROLLO.....	43
FIGURA 23: ESTRUCTURA DEL REPOSITORIO	44
FIGURA 24: VERIFICACIÓN DEL PROYECTO	50
FIGURA 25: EJECUCIÓN PERIÓDICA DE JENKINS	52
FIGURA 26: NOTIFICACIONES DE CORREO	52
FIGURA 27: COMMIT EN LOCAL	63
FIGURA 28: COMMIT AND PUSH.....	64
FIGURA 29: INICIAR SERVICIO XAMPP	66
FIGURA 30: MENÚ CONTEXTUAL DE LA VENTANA PROJECT EXPLORER.....	67
FIGURA 31: VENTANA IMPORT	67
FIGURA 32: VENTANA GIT.....	68
FIGURA 33: IMPORTAR PROYECTO GIT	68
FIGURA 34: VISTA DE TODAS LAS RAMAS DE UN REPOSITORIO.....	69
FIGURA 35: SELECCIÓN DE LA RAMA A IMPORTAR	70

FIGURA 36: IMPORTAR PROYECTO EXISTENTE	70
FIGURA 37: CONFIRMAR PROYECTO A IMPORTAR	71
FIGURA 38: VISTA DEL WORKSPACE	72
FIGURA 39: CREACIÓN DEL PROYECTO EN JENKINS	73
FIGURA 40: CONFIGURACIÓN DE REPOSITORIO GIT EN JENKINS	74
FIGURA 41: CÓDIGO DE TAREA EN JENKINS	74
FIGURA 42: SUBSISTEMA CREACIÓN Y ADMINISTRACIÓN DE CENSOS FUNCIONANDO	75
FIGURA 43: SELECCIÓN DE PROYECTOS EN SONARQUBE	76
FIGURA 44: ANÁLISIS DEL CÓDIGO DE ADMCENSUS	76
FIGURA 45: DOCUMENTACIÓN GENERADA POR DOXYGEN	77
FIGURA 46: NUEVA INCIDENCIA EN BUGTRACKER	85
FIGURA 47: NOTIFICACIONES EN BUGTRACKER	87
FIGURA 48: LISTADO DE ERRORES EN BUGTRACKER	87
FIGURA 49: ESTADO DE UNA INCIDENCIA EN BUGTRACKER	88
FIGURA 50: NOTIFICACIÓN DE ERROR SOLUCIONADO	89
FIGURA 51: INFORME DE ERRORES	89
FIGURA 52: FICHERO CON INCIDENCIA POR RESOLVER	92
FIGURA 53: RESOLUCIÓN ESPERADA DE LA INCIDENCIA	92
FIGURA 54: INCIDENCIA POR RESOLVER	93
FIGURA 55: DETALLES DE LA INCIDENCIA	93
FIGURA 56: CAMBIO DE ESTADO EN UNA INCIDENCIA	94
FIGURA 57: HISTORIAL DE CAMBIOS EN UNA INCIDENCIA	95
FIGURA 58: CREACIÓN DE ISSUE EN GITHUB	96
FIGURA 59: DETALLES DE INCIDENCIA	97
FIGURA 60: NOTIFICACIÓN EN GITHUB DE RESOLUCIÓN	97
FIGURA 61: NOTIFICACIÓN POR EMAIL DE RESOLUCIÓN	98
FIGURA 62: REGISTRO DE INCIDENCIAS ABIERTAS	99
FIGURA 63: EJEMPLO DE EXCEPCIÓN	101
FIGURA 64: MODO DEPURACIÓN DE ECLIPSE	102
FIGURA 65: MAPA DE HERRAMIENTAS PROPIO	111
FIGURA 66: MAPA DE HERRAMIENTAS GENERAL	116

3. RESUMEN

En este proyecto realizamos el subsistema de creación y administración de censos del sistema Agora@US. A lo largo de las siguientes páginas se detalla la evolución y gestión de la configuración del nombrado subsistema.

Se abordan temas como la gestión del código fuente en el que utilizamos dos herramientas distintas de control de versiones como son Subversion y GitHub (para integración interna y externa respectivamente).

En la gestión de la construcción y la integración continua, tanto interna como externa, abordamos la construcción del proyecto a partir del código fuente (haciendo uso de la herramienta Jenkins, explicada en el desarrollo de la asignatura, y Maven), el análisis de la calidad del código con SonarQube, la generación de la documentación con Doxygen, etc.

Para la gestión de cambio tratamos la gestión de impactos en el proyecto que a su vez está íntimamente unida a la depuración para minimizar o eliminar los diversos problemas que estos cambios puedan traer al proyecto que no son más que las incidencias o reportes de estos errores.

A la hora del despliegue, se ha elaborado una propuesta en común con otros grupos, para realizarlo con el resto de subsistemas.

En lo referente a las herramientas utilizadas, tanto interna como externamente, se detalla la relación entre estas en el apartado de mapa de herramientas.

Finalmente se aportan una serie de conclusiones en cuanto al desarrollo y evolución del trabajo en la asignatura de EGC.

4. INTRODUCCIÓN

Agora Voting es una herramienta que nos permite realizar votaciones de manera segura y fiable a través de Internet. Dicha aplicación será recreada en la medida de lo posible por la asignatura de Evolución y Gestión de la Configuración (EGC) en el curso 2014/2015. Debido a la complejidad del sistema, se optó por dividir en módulos funcionales, los cuales se asignaron a distintos subgrupos con un máximo de ocho componentes. Los módulos en los cuales se acordó dividir el proyecto son:

[Autenticación](#)

[Creación/Administración de votaciones](#)

[Sistema de modificación de resultados](#)

[Almacenamiento de votos](#)

[Deliberaciones](#)

[Recuento](#)

[Creación/Administración de censos](#)

[Frontend de Resultados](#)

[Visualización de resultados](#)

[Verificación](#)

[Cabina de votación](#)

Nuestro grupo, se encarga de la Creación/Administración de censos.

Dicho módulo se ocupa de asegurar cuales son los posibles votantes y que estos estarán recogidos en un censo asociado a una votación. Los votantes podrán ver que su voto ha sido almacenado correctamente y si ha votado con anterioridad.

De manera técnica se trata de realizar una interfaz del sistema, donde un usuario, en este caso el administrador/creador de la votación puede indicar cuales son los participantes/votantes de dicha votación. De estos participantes se llevará un control para evitar que un votante pueda votar en más de una ocasión, impidiendo así problemas a la hora de relacionar a un votante con su voto, ya que si esto fuera posible se podrían modificar los votos y por tanto no sería fiable dicho sistema. Además de proporcionar dicha información a los módulos que así lo requieran.

El objetivo general del trabajo es que el grupo ponga en práctica y observe cómo se ponen en funcionamiento en un proyecto real todos los conceptos teórico-prácticos impartidos en la asignatura y se profundice en ellos todo lo que su motivación lo lleve.

En nuestro caso, tenemos una relación directa con los módulos de *Autenticación*, *Creación/Administración de votaciones*, *Cabina de votación* y *Deliberaciones*. Por tanto, se determinará en cada apartado los problemas y soluciones que hemos encontrado al relacionarnos con dichos módulos y durante el desarrollo del nuestro.

5. GESTIÓN DEL CÓDIGO FUENTE

5.1. GESTIÓN DE RAMAS

5.1.1. PROBLEMA

Para un buen uso de un repositorio en el cual van a trabajar varios colaboradores, es fundamental tener un control de versiones y poder trabajar en distintas funcionalidades del proyecto haciendo el uso de ramas. Si estas no fueran utilizadas y desarrollara todo el equipo a la vez surgirían dos problemas: trabajar en local hasta tener versiones estables de la funcionalidad asignada, lo cual a la hora de unir el código generaría multitud de fallos; o subir los cambios conforme se trabaja con lo que podría haber código repetido, conflictos o incluso funcionalidades incompletas que hagan que el resto del trabajo realizado pueda fallar.

Por tanto en dicha colaboración se deben marcar las pautas a seguir de cómo, cuándo y dónde se deben aplicar. Es por ello que se hacen uso de distintos “tipos” de *branch*, pudiendo organizar en una rama las versiones estables del proyecto, en otra tener una copia de seguridad además de crear tantas como se necesiten a la hora de desarrollar funcionalidades distintas para impedir/disminuir los problemas que se pueden presentar a la hora de trabajar en un mismo código en local. Si la gestión de la rama no es adecuada se pueden presentar diversos problemas a la hora de desarrollar. Alguno de estos problemas, puede ser, realizar un requisito funcional en un *branch* usado para versiones estables, junto a lo que esto conlleva.

5.1.2. SOLUCIÓN PROPUESTA

La gestión de ramas se ha dividido en dos gestores de versiones, SVN con ProjETSII y GitHub.

El motivo de ello es debido a que utilizamos SVN para el desarrollo del subsistema en el equipo mientras que en GitHub se iban subiendo versiones estables de funcionalidad para tener al resto de subsistemas con la versión más actualizada de nuestra parte del proyecto. Debido a esta circunstancia se ha migrado el código y su historial desde SVN hacia un nuevo repositorio de Git el cual se usará para posibles cambios futuros dando por finalizado el uso de SVN.

Como se han usado distintos repositorios a lo largo del tiempo, explicaremos en primer lugar como hemos realizado la gestión de SVN y a continuación, una propuesta de gestión para el nuevo repositorio en el cual hemos migrado el contenido de este.

Con SVN, a diferencia de Git, todas las acciones se realizan con el repositorio remoto.

Se diferencian dos tipos de ramas en las que se ha ido desarrollando el sistema, una que sería el *trunk*, rama común a todos los desarrolladores, en la cual se almacena una versión estable del sistema, es decir, la última versión que ha sido testeada y por tanto aprobada. Y otra rama destinada al desarrollo de nuevas funcionalidades o modificaciones de la versión almacenada en el *trunk* ya sea por cambios de requisitos, cambios en funcionalidades, etc.; que parte del *trunk* para pasar a ser posteriormente una nueva versión del mismo.

Además de esto, en la gestión de ramas hay que tener en cuenta otro elemento que es el *tag*. Éste mantiene *snapshots* o copias de seguridad del proyecto, es decir, del código almacenado en *trunk*.

En la siguiente imagen se muestra la gestión explicada, donde se pueden ver las dos ramas utilizadas en el proyecto siendo “ADMCensus” la rama principal o *trunk* y “Cambios 10-11-2014” una nueva rama para realizar cambios en las funcionalidades de la rama principal.

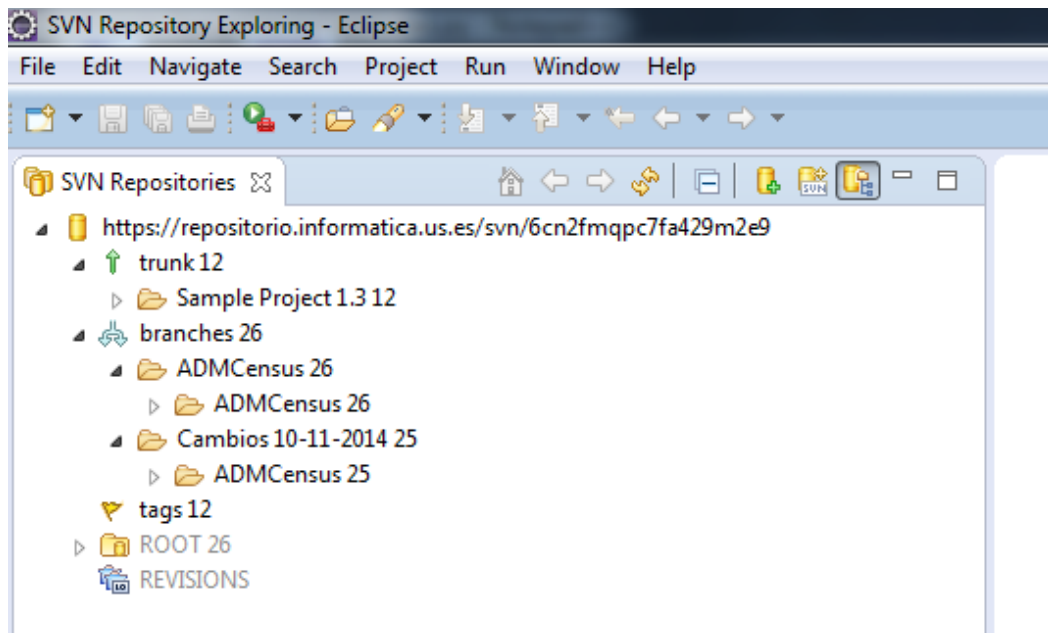


Figura 1: Vista de todos los Tags del repositorio

Como *baseline* inicial se estableció un sistema que sólo contenía la clase de dominio a manejar, que en éste caso se denomina “Census”, la cual contiene los atributos y los datos necesarios para la manipulación de datos del sistema.

Por tanto, una vez establecido el *trunk*, el administrador, el cual es el creador del repositorio y el gestor del mismo, tiene la opción de abrir uno o varios *branches* o ramas, en las que se llevarán a cabo las tareas de desarrollo paralelamente. Aunque la creación de ramas está permitida por todos, solo la realiza el administrador, para poder administrar de manera eficiente las nuevas funcionalidades, evitando varias ramas para una misma funcionalidad.

Estos *branches* son repartidos a distintos integrantes del grupo a los que se les asignarán los permisos correspondientes, ya sea de lectura, escritura o ambos. Estos permisos van en función del rol que desempeñan en el proyecto. En nuestro caso, debido a la escasa funcionalidad del subsistema, solo han sido necesario dos desarrolladores, los cuales tenían todos los permisos.

En el desarrollo del sistema, se abrieron dos *branches*, ya que debido a la envergadura del sistema, no fueron necesaria más. El *trunk* o rama principal que contendría finalmente los cambios realizados en las ramas; y el *branch* “Cambios 10-11-2014” donde se realizaron cambios tanto de requisitos no contemplados y/o equivocados en un principio, y de funcionalidades y comunicación con otros subsistemas.

El desarrollo realizado en un *branch* determinado queda salvado por los ***commits***, en los que antes de realizar esta acción hay que describir las tareas realizadas con el fin de llevar un control de las versiones de los mismos y en qué consiste dicha tarea.

En cuanto al sistema de *commit* se decidió no realizar uno “por cada línea de código realizada”, ya que genera dificultades a la hora de comparar versiones. Al mismo tiempo, no se vió factible la idea de hacer *commit* el último día de la entrega por dos razones:

- Posibilidad de pérdida de cambios en local por error o rotura de equipo
- Puede que una funcionalidad dependa de la otra y por tanto pueden existir daños colaterales, por lo que podemos atrasar el desarrollo de uno de los desarrolladores.

Cada desarrollador trabaja en una funcionalidad determinada en local hasta que este decide que su tarea esta lista y puede hacer un *commit* en la rama.

En la siguiente imagen podemos comprobar un ejemplo de ello.

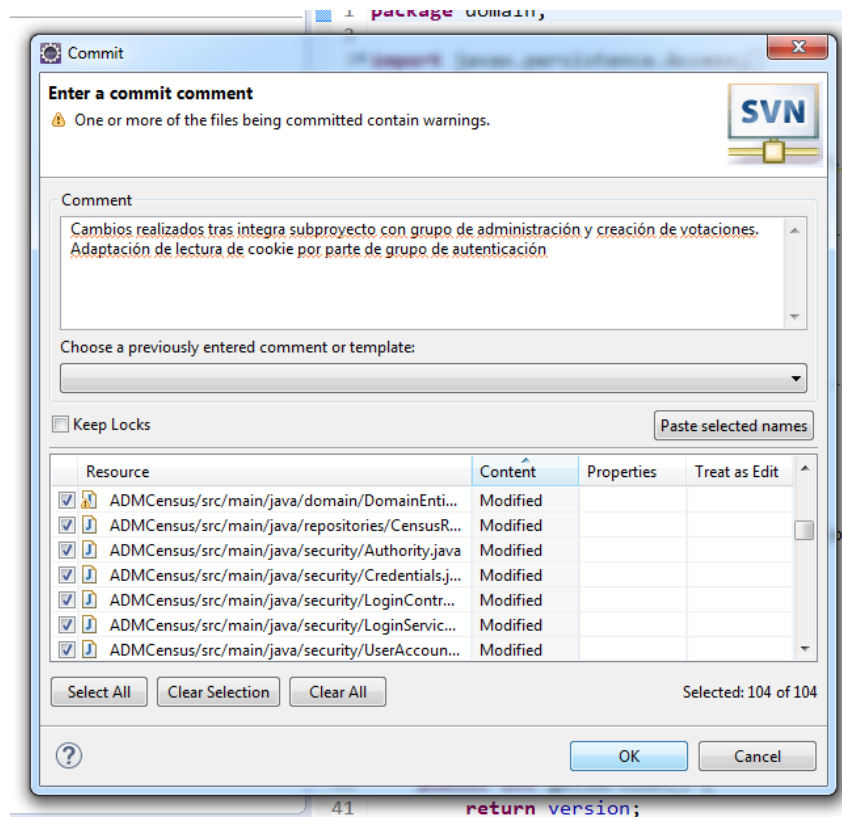


Figura 2: Ventana de commit

Las ramas fueron creadas con una fecha límite, la cual está previamente establecida, por lo que antes de la finalización de dicho plazo, la funcionalidad debía ser completada y probada, con los test pertinentes, para posteriormente realizar un *merge*.

Los **merges**, son la integración de las nuevas funcionalidades con la rama principal o *trunk*, de modo que dicha rama pasaría a tener una nueva versión estable y el código con la máxima funcionalidad posible. Éstos fueron realizados por el administrador del repositorio, tras confirmar que los cambios en las ramas eran los correctos, es decir, que el administrador los haya realizado o porque el otro desarrollador lo haya probado.

En las siguientes imágenes se puede observar cómo realizar un *merge*:

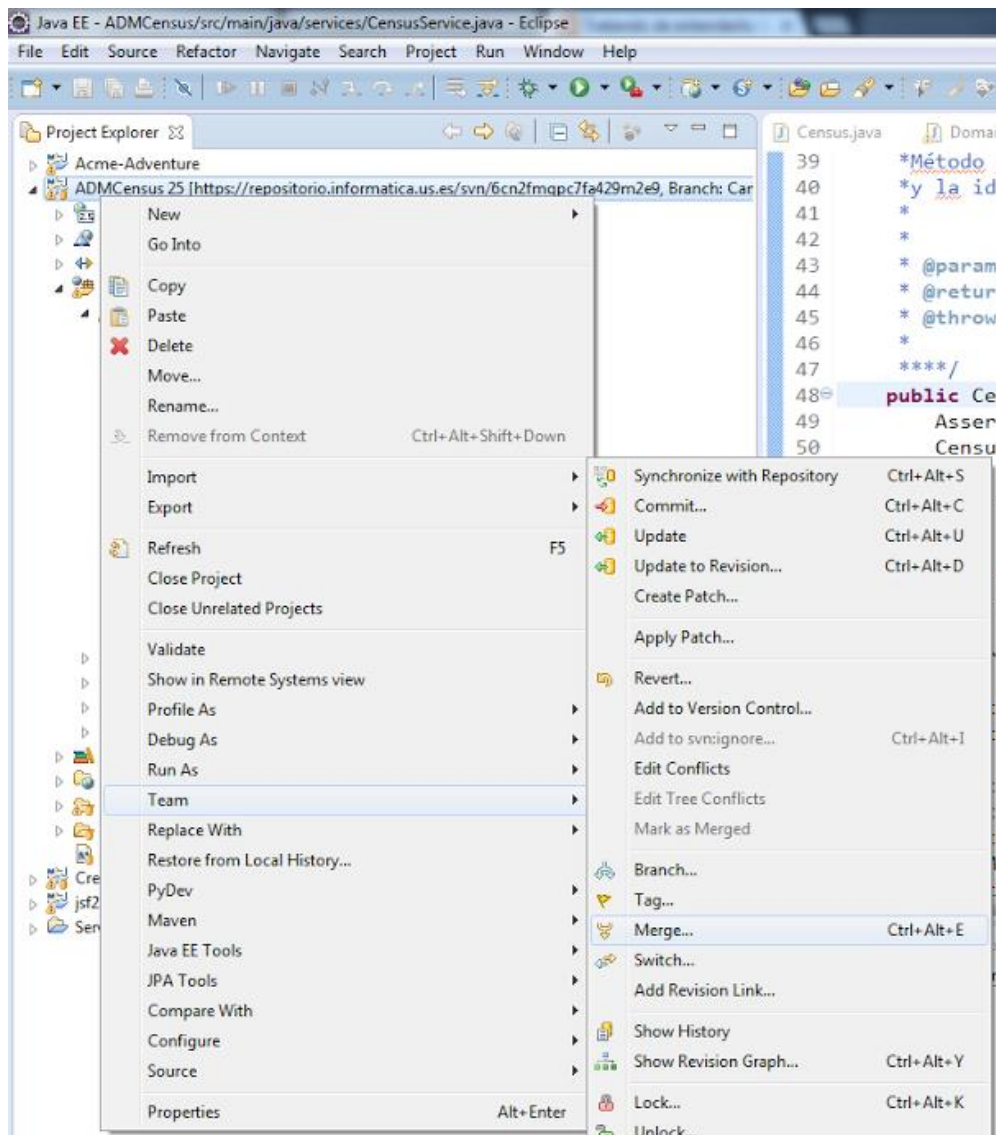


Figura 3: Menú Team desplegado

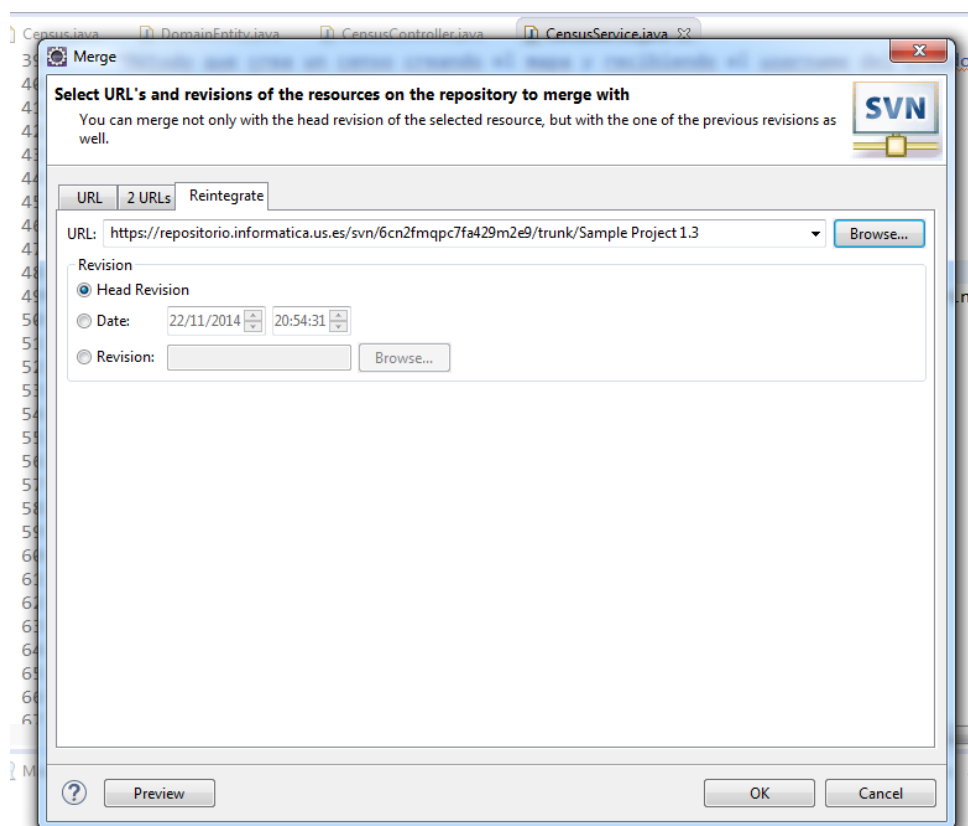


Figura 4: Realizando un Merge

Una vez realizado el *merge* de una determinada rama, ésta se considera cerrada y llegado a este estado, dicha rama deja de usarse por el equipo de desarrollo. Como aclaración, decir que las ramas no se eliminan, ya que por buenas prácticas consideramos que esto no se debe realizar.

Una rama cerrada será usada por el equipo de desarrollo en los casos en los cuales se requiera modificar la funcionalidad tratada en dicha rama.

Entonces, una vez alcanzado este punto en el que la versión del *trunk* ha cambiado, se crea un nuevo *tag*, ya que se ha alcanzado una nueva versión estable.

En la próxima imagen se muestra la creación de un nuevo *tag*:

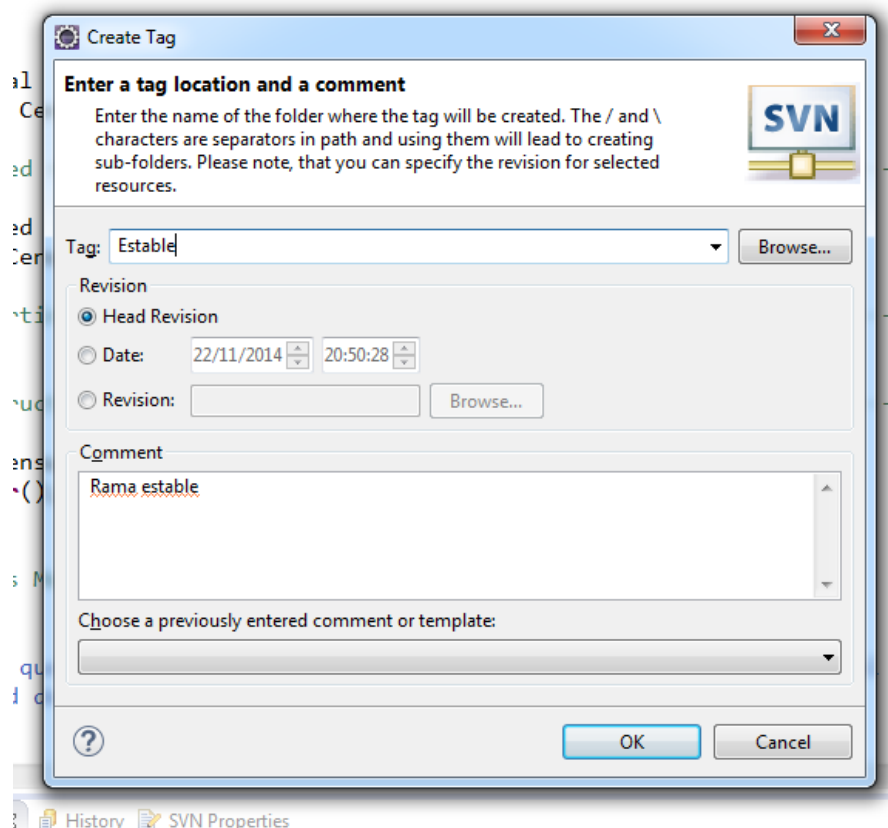


Figura 5: Creación de un nuevo Tag

Por tanto, como podemos comprobar la ventaja de esto reside en que si al desarrollar la tarea asignada en un determinado *branch* se encuentra con un error o no puede continuar con el desarrollo debido a algún problema como puede ser que dicha funcionalidad no haga falta desarrollar, se puede volver a la versión almacenada en el *tag*, es decir, al *baseline*.

Finalmente, cabe destacar, las situaciones en las que se producen **conflictos**, aquellas en las que un determinado archivo que se pretende subir al repositorio tiene distinta versión a la que tiene el remoto o bien se ha modificado simultáneamente una línea existente por ambos desarrolladores, por lo que se generan conflictos.

Dichas situaciones son difíciles de solventar, por lo que para llevar a cabo el desarrollo del sistema se han mantenido buenas prácticas de desarrollo con el fin de evitar esto mismo, aunque es complicado evitar los conflictos en el desarrollo software.

Aun así, en el caso de que surja algún conflicto, el encargado de resolverlo será el mismo desarrollador que lo haya generado, debiendo asegurarse de qué termina subiéndose al repositorio y qué es lo que se desecha.

En la siguiente imagen se puede comprobar lo anteriormente descrito de cómo se revisa un conflicto para ver qué cambios se persisten. En este caso, vemos que tenemos en local una variable “fecha_comienzo” (parte izquierda de la imagen) y en el repositorio “fecha_inicial” (parte derecha de la imagen), en este caso, vamos a llevar el cambio del repositorio al local para quitar el conflicto:

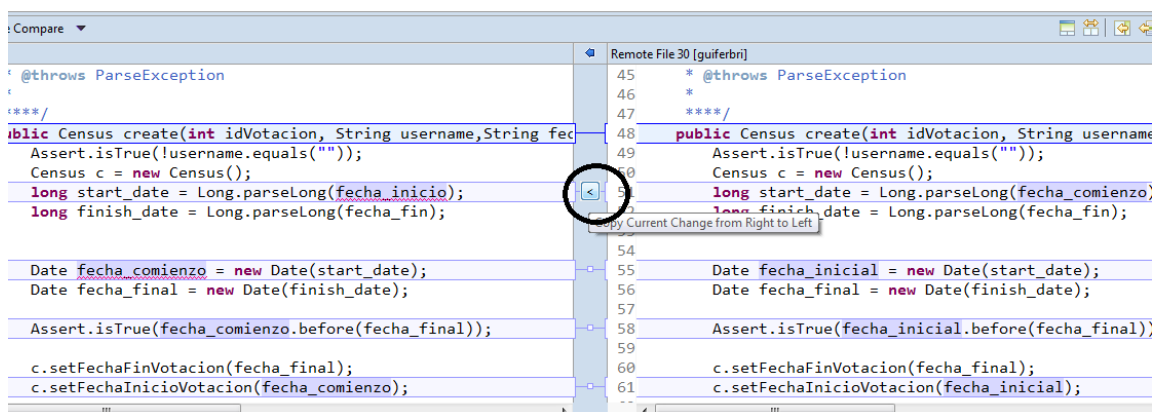


Figura 6: Revisión de conflicto

Por seguridad, el encargado de resolver los conflictos, si ve que es un cambio en la funcionalidad importante, deberá realizar un par de test funcionales con la herramienta JUnit para asegurar el correcto funcionamiento del proyecto.

Una vez asegurado dicho funcionamiento, en la siguiente imagen, vamos a realizar un *commit* para llevar este cambio al repositorio.

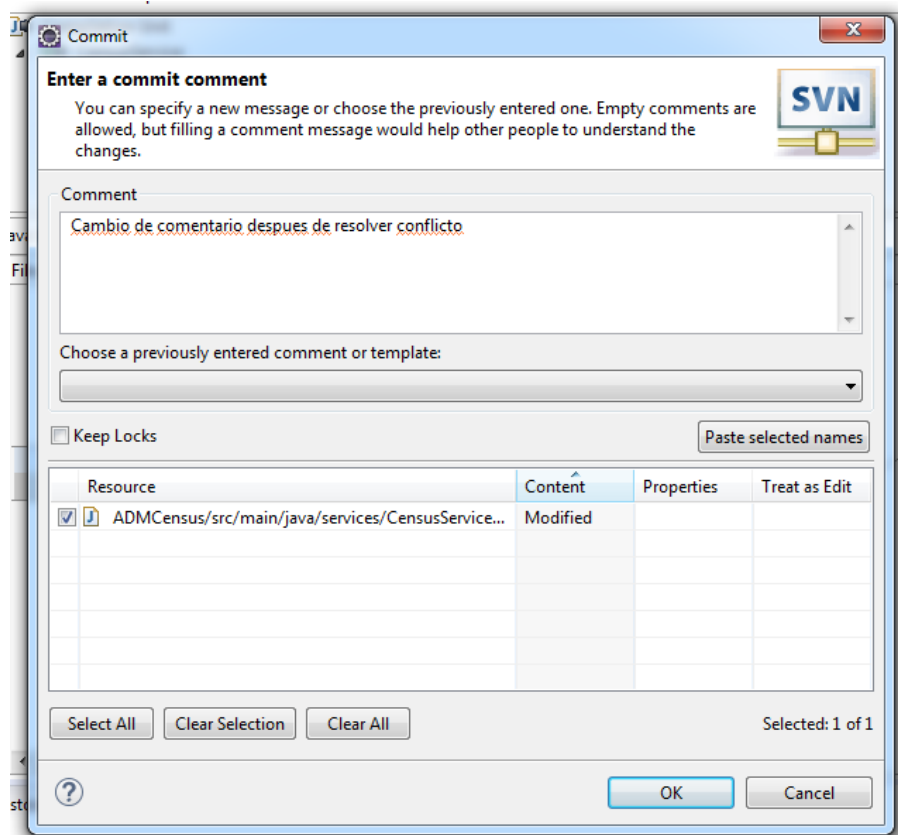


Figura 7: Commit del cambio realizado

5.1.3. LECCIONES APRENDIDAS

Utilizar ramas es la mejor forma de trabajo ya que se podrán distinguir distintas ramas: una en la cual habrá siempre una versión estable del trabajo para que todos los colaboradores puedan seguir desarrollando sin fallos de funcionalidades que no le han sido asignadas o están incompletas. Y otras ramas donde se irán desarrollando las distintas funcionalidades.

De esta forma se mantendrá el código ordenado y estable en cualquier momento del desarrollo, además de un buen control de versiones.

5.2. SVN A GIT

5.2.1. PROBLEMA

Debido a que se realizaban integraciones periódicas con los distintos subsistemas, se propuso utilizar un repositorio compartido en el cual cada subsistema tendría actualizado su parte de código a la última versión estable que tuvieran. Por ello, se decidió realizar una migración del historial del repositorio de SVN a Git, para así mantener la versión estable. En caso de futuras modificaciones/funcionalidades se realizarán en dicho repositorio.

5.2.2. SOLUCIÓN PROPUESTA

Cuando se consiguió una primera versión final, ya se dejó de utilizar SVN y se empezaron a realizar pequeños cambios sobre GitHub para solventar pequeños fallos reportados por otros grupos o detectados por el nuestro.

Si se detectaba un error se realizaban directamente sobre la rama asignada a nuestro subsistema en el repositorio compartido de GitHub. En el caso en el cual el error necesite un mayor tiempo para su resolución, proponemos que este se realice en una rama dentro del repositorio de GitHub EGCAAdminCensos, el cual contiene el código y el historial del repositorio de SVN ya que hemos migrado estos elementos de un repositorio a otro.

A continuación explicaremos como se ha realizado dicha migración pero antes de esto expondremos cual ha sido el motivo por el cual hemos decidido realizarla. El motivo principal fue la decisión de utilizar un repositorio común para facilitar la integración con los distintos subsistemas, consideramos que en dicho nuevo repositorio deberíamos tener el código y su historial, por ello migramos desde el repositorio interno (SVN) hacia el nuevo repositorio de EGCAAdminCensos en GitHub el cual hemos tenido que crear.

En primer lugar, creamos un repositorio en GitHub donde alojaremos el código y los *commits* realizados anteriormente en el repositorio SVN donde se trabajó en un principio. En nuestro caso el repositorio de GitHub se llama “EGCAAdminCensos”.

En nuestra carpeta en local donde trabajaremos, inicializamos y clonamos el repositorio creado. A continuación, clonamos el repositorio de SVN en dicha carpeta con el siguiente comando:

```
git svn clone https://repositorio.informatica.us.es/svn/6cn2fmqpc7fa429m2e9  
-T trunk ./
```

Con esto se realiza un Checkout del repositorio SVN al repositorio Git, con `-T trunk` indicamos que la rama principal se llama trunk y con `./` para que use la carpeta en local donde estamos trabajando como inicio. Con este comando los *commits* realizados en SVN se están convirtiendo en *commits* realizados por Git y mantener de esta forma los cambios realizados junto a sus comentarios aclarando/explicando las modificaciones realizadas, es decir, traspasar el historial de cambios de SVN a Git. Notar que se han traspasado los cambios, todavía no se han pasado los archivos ubicados en SVN. Para ello realizamos lo siguiente:

Ejecutamos el comando `git branch -a` y veremos que aparece una nueva rama “trunk” que es de un repositorio remoto (el de SVN), que contiene, como hemos dicho antes, el historial de cambios.

Ejecutamos entonces el comando `git merge remotes/trunk` para copiar los ficheros desde el repositorio remoto hacia “EGCAdminCensos”.

En la siguiente imagen se muestran las ramas anteriormente nombradas, tanto locales (origin/HEAD y origin/master) como remotas (trunk):

```

posh~git ~ EGCAAdminCensos [master]
M      ADM Census/src/main/webapp/views/census/list.jsp
M      ADM Census/src/main/webapp/views/master-page/messages.properties
M      ADM Census/src/main/webapp/views/master-page/header.jsp
M      ADM Census/src/main/webapp/views/master-page/messages_es.properties
M      ADM Census/.settings/org.eclipse.jdt.core.prefs
M      ADM Census/.settings/org.eclipse.m2e.core.prefs
M      ADM Census/.settings/org.eclipse.wst.common.project.facet.core.xml
M      ADM Census/.settings/org.eclipse.jpt.core.prefs
M      ADM Census/.settings/.jsdtscope
M      ADM Census/.settings/org.eclipse.wst.validation.prefs
M      ADM Census/.settings/org.eclipse.wst.common.component
r34 = 2a28d1192a2dbc3da504cc69ef5d89756e210421 (refs/remotes/trunk)
D:\Universidad\Cuarto\Evolucion y Gestion de la Configuracion (ECG)\RepoCensos\EGCAAdminCensos [master]> git branch -a
* master
remotes/origin/HEAD -> origin/master
remotes/origin/master
remotes/trunk
D:\Universidad\Cuarto\Evolucion y Gestion de la Configuracion (ECG)\RepoCensos\EGCAAdminCensos [master]> git merge remotes/trunk

```

Figura 8: Ramas locales y remotas

Una vez realizado estos pasos, solo falta subirlo al repositorio de GitHub, ya que como hemos dicho anteriormente, todos estos cambios se han realizado en local. Para ello ejecutamos: **git push**

Tras realizar esta subida con éxito vamos al repositorio de GitHub para comprobar que todo se ha realizado correctamente y hemos migrado los cambios del repositorio antiguo al nuevo.

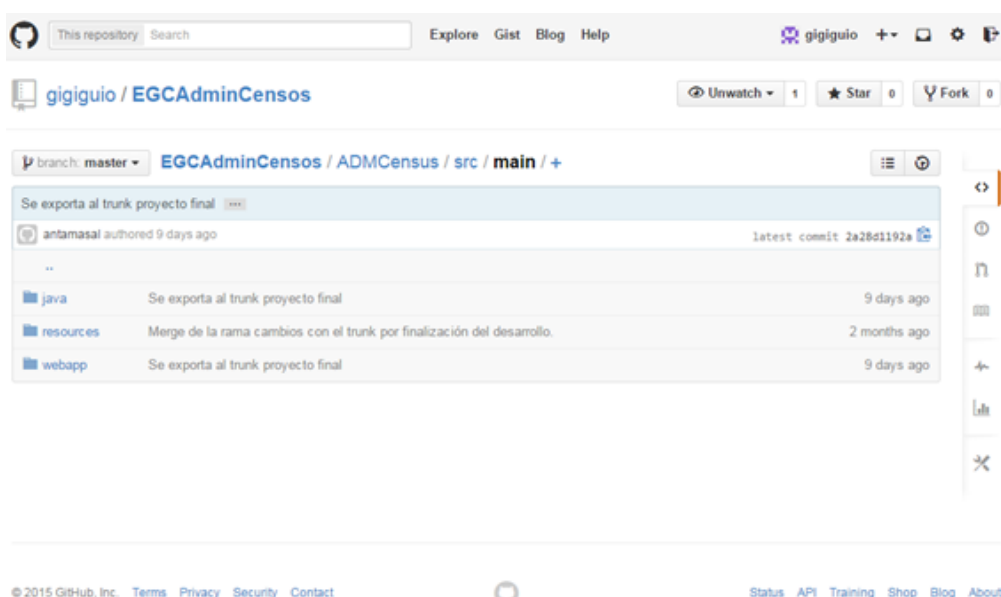


Figura 9: Historial en GitHub

Para posibles mejoras futuras se seguiría trabajando en el nuevo repositorio creado (<https://github.com/gigiguio/EGCAdminCensos.git>) cuya gestión sería similar a la llevada en SVN. Es decir, crear ramas para cambios, nuevas funcionalidades y mantener en la rama *master* o principal la versión estable del subsistema. A su vez, se cierran ramas cuando la funcionalidad esté acabada, quedando inutilizadas para el desarrollo, exceptuando que exista una modificación futura en dicha funcionalidad. Una vez se considere finalizado el desarrollo se pasarían los cambios a la rama asignada en el repositorio compartido para que de esta forma todos los subsistemas tengan acceso a la última versión estable del código.

5.2.3. LECCIONES APRENDIDAS

Creemos que ha sido interesante el uso de GitHub porque para su uso futuro es más fácil y ágil a la hora de realizar cambios sin la necesidad de hacer copias del repositorio de SVN o descargarnos la última versión para poderlo subir a Git. Sin embargo, deberíamos haberlo usado desde un primer momento ya que hemos tenido que gestionar dos repositorios de manera simultánea y posteriormente migrar los cambios de uno al otro para mantener el historial de versiones de lo desarrollado en SVN.

5.3. CREACIÓN Y APLICACIÓN DE PATCH

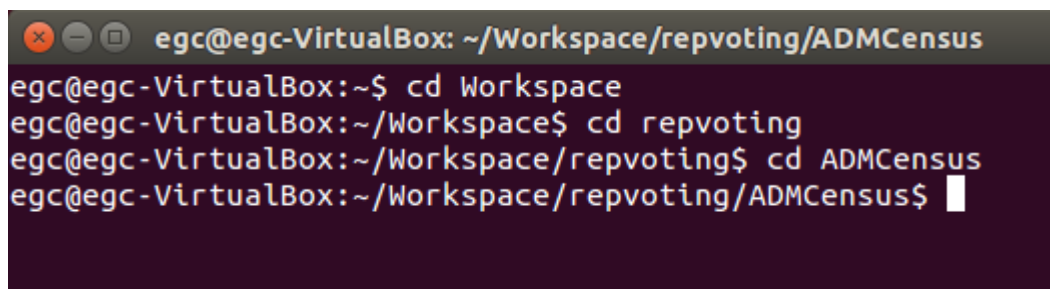
5.3.1. PROBLEMA

En todo proyecto informático es necesaria la modificación del código fuente con la finalidad de arreglar errores, modificar funcionalidades, etc. A veces, dichas modificaciones son realizadas por uno de los desarrolladores, pero es necesaria para todo el equipo de trabajo.

Para realizar dicha modificación sin tener que realizar una subida a repositorio, existen los *patch*. Los *patch* son partes del código modificados que se pueden aplicar automáticamente al código fuente para solventar los problemas mencionados anteriormente.

5.3.2. SOLUCIÓN PROPUESTA

Como solución se decidió generar los parches con la herramienta Subversion. Generar un *patch* con Subversion es sencillo usando una ventana de comandos. En primer lugar hay que dirigirse hasta el directorio en el cual se encuentra el *WorkSpace* con el proyecto enlazado al repositorio mediante SVN. Si el fichero del cual se desea generar un *patch* se encuentra en una rama, como es el caso de la siguiente imagen, habrá que dirigirse a dicha la rama:

A terminal window titled 'egc@egc-VirtualBox: ~/Workspace/repvoting/ADMCensus'. The terminal shows a sequence of commands to navigate to the workspace directory: 'cd Workspace', 'cd repvoting', and 'cd ADCensus'. The prompt is currently at the ADCensus directory.

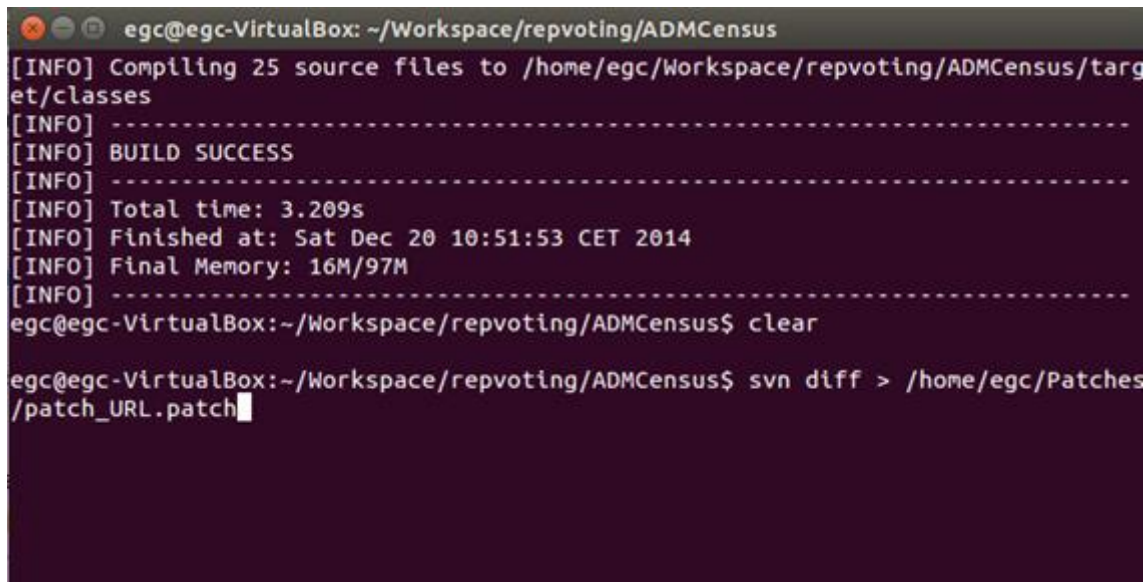
```
egc@egc-VirtualBox: ~/Workspace/repvoting/ADMCensus
egc@egc-VirtualBox:~$ cd Workspace
egc@egc-VirtualBox:~/Workspace$ cd repvoting
egc@egc-VirtualBox:~/Workspace/repvoting$ cd ADCensus
egc@egc-VirtualBox:~/Workspace/repvoting/ADMCensus$
```

Figura 10: Situarse en la rama de cambios

En el ejemplo de la imagen nos hemos situado en el *WorkSpace* “Workspace” y el directorio llamado “ADMCensus” se trata de una rama del repositorio.

Una vez situados en dicha rama, añadimos el parche (modificaciones en el código fuente sin haber realizado todavía *commit*) y ejecutamos la siguiente línea de comando:

```
svn diff > rutaDondeGuardarFichero/nombrePatch.patch
```

A screenshot of a terminal window titled 'egc@egc-VirtualBox: ~/Workspace/repvoting/ADMCensus'. The terminal shows the output of a compilation process, including '[INFO] Compiling 25 source files to /home/egc/Workspace/repvoting/ADMCensus/target/classes', '[INFO] BUILD SUCCESS', and '[INFO] Total time: 3.209s'. After the compilation, the user enters 'clear' to clear the screen. Then, the user enters 'svn diff > /home/egc/Patches/patch_URL.patch', which generates a patch file. The terminal output is as follows:

```
egc@egc-VirtualBox: ~/Workspace/repvoting/ADMCensus
[INFO] Compiling 25 source files to /home/egc/Workspace/repvoting/ADMCensus/target/classes
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 3.209s
[INFO] Finished at: Sat Dec 20 10:51:53 CET 2014
[INFO] Final Memory: 16M/97M
[INFO] -----
egc@egc-VirtualBox:~/Workspace/repvoting/ADMCensus$ clear

egc@egc-VirtualBox:~/Workspace/repvoting/ADMCensus$ svn diff > /home/egc/Patches/patch_URL.patch
```

Figura 11: Aplicar parche

Tras el símbolo > se añade la ruta, en la cual se generará el fichero *patch*, y el nombre indicado para dicho fichero. En la siguiente imagen se puede ver la modificación realizada, concretamente se ha cambiado la URL:

<http://localhost:8080/ADMCensus/census/prueba.do>

Por:

<http://http://localhost/auth/index.php>.

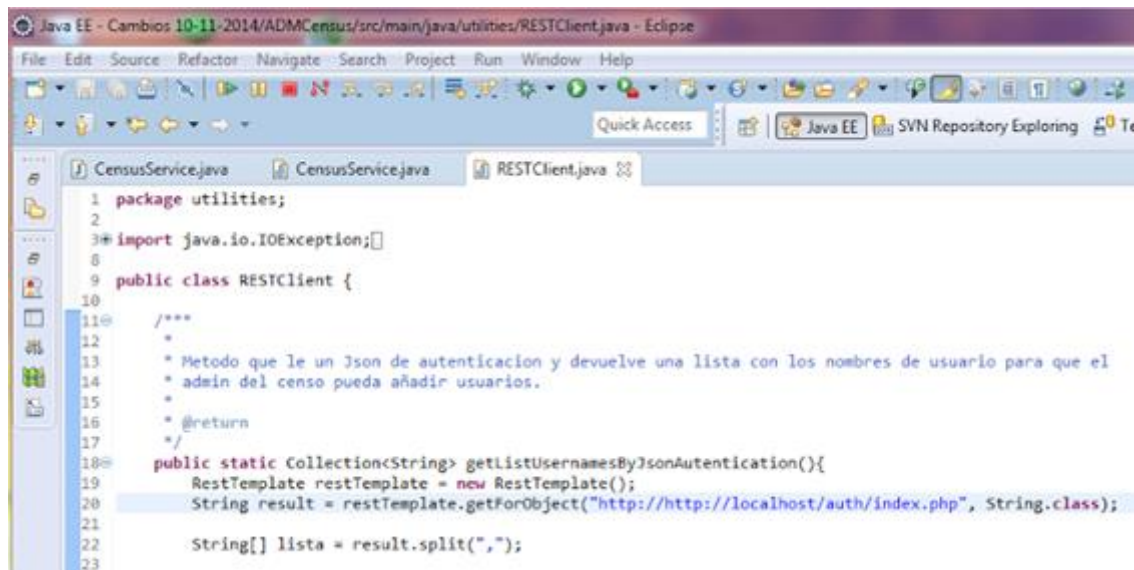


Figura 12: Cambio en el código

En esta otra imagen se muestra el formato con el cual se ha generado el *patch* en el fichero indicado en el comando ejecutado.

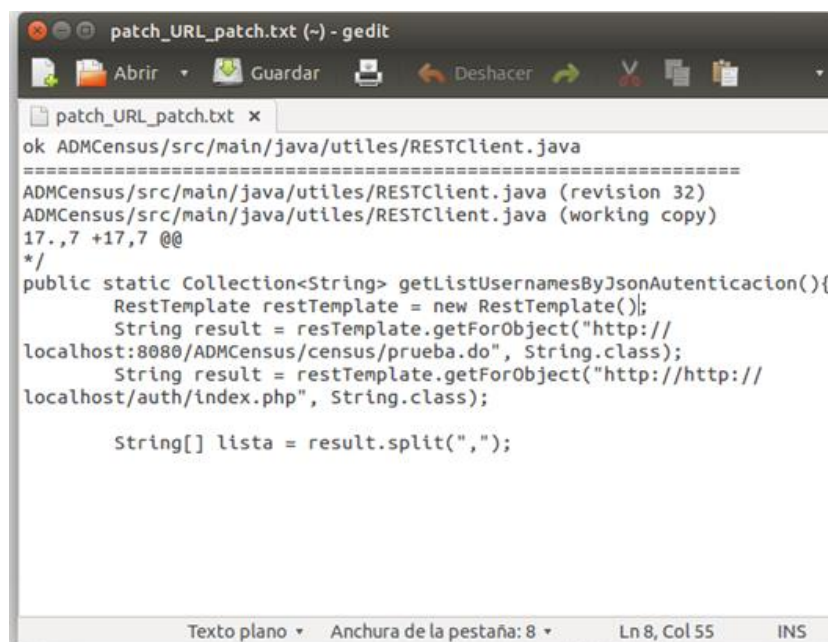


Figura 13: Fichero patch

Para aplicar un *patch* recibido hay que situarse en el directorio en el que se encuentre el proyecto, o como en el caso anterior, en la rama donde se desee aplicar el *patch*. Una vez situados ejecutar el siguiente comando en el terminal:

```
patch -p0 -i rutaDondeGuardarFichero/nombrePatch.patch
```

5.3.3. LECCIONES APRENDIDAS

Los *patch* son necesarios para el correcto desarrollo de un proyecto software, gracias a este apartado podremos ver una correcta realización de estos para la actualización del producto, y para mantener versiones estables del producto. Además del uso de parches, se ha ampliado el conocimiento de utilidades que ofrece la herramienta Subversion, tales como la creación de parches.

5.4. ROLES

5.4.1. PROBLEMA

Dado que en todo proyecto software se trabaja de forma grupal, la cooperación entre los integrantes de dicho grupo puede ser caótica si no se toman las medidas oportunas.

Para ello se asignan una serie de roles para establecer la responsabilidades a la hora de gestionar el proyecto. En el apartado de gestión del código ocurre exactamente lo mismo. Dependiendo de la magnitud del proyecto, será necesaria la asignación de más personal para la gestión de código, además de la mayor pluralidad de roles para el control adecuado del desarrollo del producto.

5.4.2. SOLUCIÓN PROPUESTA

En un primer momento se pensó que para el desarrollo del código sería necesario un grupo de varios desarrolladores, sin embargo, una vez contemplado el alcance de nuestro subsistema se optó por un grupo compuesto por dos desarrolladores.

A pesar de la existencia de sólo dos desarrolladores del subsistema, los roles principales para gestionar el código han sido los siguientes:

Administrador: Es la persona encargada de aceptar los cambios que han producido conflictos. Tras hablar con la persona que ha realizado el cambio en la funcionalidad y como consecuencia generar dicho conflicto, será este quién permita la subida final al repositorio. También es el encargado de realizar los *merge* que se requieran. Además desarrolla y realiza subidas como un desarrollador normal.

Desarrollador: Persona/s cuya función es realizar el código asignado y subirlo a la rama de trabajo correspondiente. En caso de haber conflicto se pondrá en contacto con el administrador para dar parte de ello y resolverlo conjuntamente.

5.4.3. LECCIONES APRENDIDAS

Creemos que la asignación de roles es básica para la optimización a la hora de realizar código fuente. La diversidad de tareas a la hora de realizar código fuente hace necesario que en el equipo de trabajo existan diferentes desarrolladores con distintas especializaciones, tales como analistas, programadores, testers, etc.

En nuestro caso, debido a la magnitud del proyecto, hemos sido capaces de comprobar la cantidad de desarrolladores necesarios y las responsabilidades de estos, asignando roles acorde a lo mencionado anteriormente.

5.5. POLÍTICA DE NOMBRE Y ESTILOS UTILIZADOS EN EL CÓDIGO FUENTE

5.5.1. PROBLEMA

Tanto la política de nombre como los estilos en la gestión de código fuente conllevan al establecimiento de patrones o reglas a la hora de su desarrollo. Cada proyecto puede tener su propia política, o mantener la misma política para todos los proyectos de una compañía.

La política de nombre es la que se encarga de establecer el nombre de variables, métodos, vistas, etc. De esta manera llegamos a un consenso por el cual todos los desarrolladores mantendrán una hegemonía a la hora de nombrar los diferentes elementos ya mencionados, facilitando su organización.

El estilo se centra en el formato que se tomará en la realización del código gestionando temas como la tabulación, el uso de minúsculas y mayúsculas, etc.

Otro tema a tratar es la estabilización de un estilo a la hora de realizar los *commits*, deben venir acompañados de un comentario claro y conciso con cierto patrón para evitar confusiones y/o discrepancias entre los desarrolladores del código.

5.5.2. SOLUCIÓN PROPUESTA

- Realizando *commit*: en la medida de lo posible, se ha intentado seguir el siguiente patrón para que el desarrollador que haga *update*, pueda identificar rápidamente los cambios que se está actualizando.
 - Diferenciar *commit* de arreglar fallos, y *commit* de nueva funcionalidad (interna o externa)
 - Tipo de mensaje en el *commit*: p.e: NombreRama - FAIL - LUGAR (Master - Fallo - Método crear censo); p.e: Master - FUNC - LUGAR (Master -Funcionalidad - API Puede borrar)
- Estilo:
 - Las variables se definen con el nombre del objeto a devolver y empezando con minúscula.
 - El nombre de la clase debe empezar siempre con mayúscula.

- Los servicios deben llamarse con la siguiente estructura: NombreEntidadService.
- Los repositorios deben llamarse con la siguiente estructura: NombreEntidadRepository.
- Los métodos se llamarán como la función que realicen en inglés, empezando con minúscula. Si el nombre es más de una palabra irán todas juntas siendo la primera en mayúsculas (lowerCamelCase). Ej:

```
public boolean updateUser(int censusId, String username){...}
```

- El nombre del proyecto y el de la base de datos deben ser el mismo.
- En las clases de dominio primero se definirán los atributos, a continuación los getter y setter de estos. En el caso de haber relaciones con otras entidades, se definirán dichas relaciones después de los getter y setter de los atributos de la propia clase.
- El código debe estar bien tabulado, como se muestra en la siguiente imagen:

```
/**
 * Metodo utilizado por cabina para actualizar el estado de voto de un usuario
 *
 * @param censusId
 * @param token
 */
public boolean updateUser(int censusId, String username) {
    boolean res = false;
    Assert.isTrue(!username.equals(""));
    Census c = findOne(censusId);
    HashMap<String, Boolean> vpo = c.getVoto_por_usuario();

    if (vpo.containsKey(username) && !vpo.get(username)){

        vpo.remove(username);
        vpo.put(username, true);
        res = true;
    }

    c.setVoto_por_usuario(vpo);
    save(c);

    return res;
}
```

Figura 14: Ejemplo de código fuente

- Estilo de las vistas: Para mostrar contenido en una página jsp, siempre usamos el comando `jstl:out` de la librería `jstl`. Otra buena práctica es, para no repetir código innecesariamente, crear tags (etiquetas) para que las llamadas sean más sencillas y las vistas estén menos sobrecargadas.
- Se debe añadir un comentario sobre los métodos donde se explique una breve descripción de la funcionalidad que realizan.

5.5.3. LECCIONES APRENDIDAS

Es necesario el seguimiento de patrones a la hora de programar para mantener un orden. Desarrollando el código basándose en una estructura de codificación común e intuitiva garantizamos que esta sea limpia y eficiente.

Además de la estructuración del código, la aplicación de estilo a la hora de realizar subidas al repositorio favorece la agilidad de la comunicación entre desarrolladores gracias a los comentarios suministrados por los *commits* realizados hasta ese momento.

5.6. EJERCICIO

“Realizar un Merge de una rama con el trunk, asignando posteriormente un tag.”

Resolución

Para poder dar solución al ejercicio propuesto, debemos tener en nuestro repositorio al menos una rama más aparte del *trunk* para que esto pueda ser posible.

Comentar, que esta solución se ha realizado haciendo uso de SVN, con una estructuración de dos ramas, la principal o *trunk* y la rama “Rama1” que será usada para realizar la acción de *merge*.

En primer lugar, comencemos por comprobar que lo desarrollado en la rama “Rama1” es totalmente correcto y por tanto no contendrá errores, ya que si ese es el caso daría ocasión a un conflicto, cosa que hay que intentar evitar para llevar a cabo unas buenas prácticas de la gestión del código fuente.

Una vez comprobado esto, procedemos a actualizar tanto el *trunk* como la rama en cuestión, además de asegurarnos de que nadie trabaja sobre dicha rama.

Tal y como se ha comentado antes, el proyecto como mínimo debe estar de la siguiente manera:

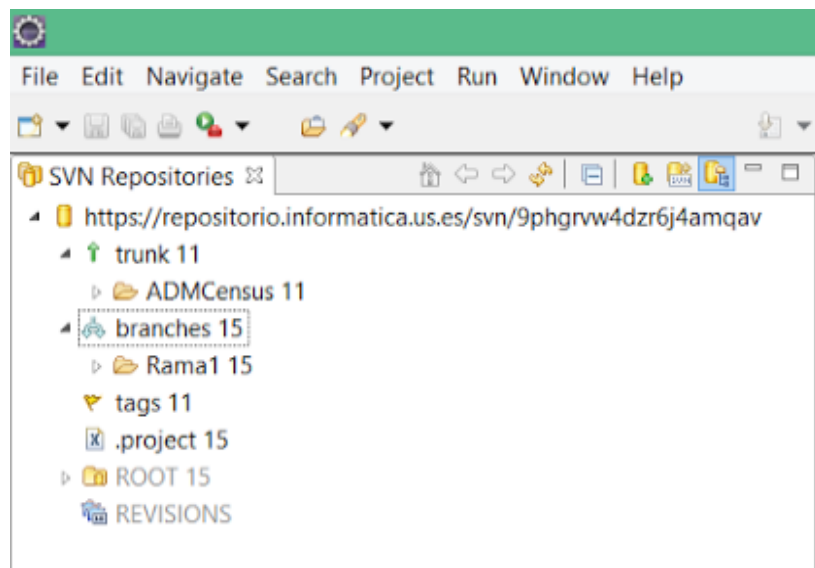


Figura 15: Estructura del repositorio

Ya teniendo la última versión de ambas ramas, procedemos a seleccionar la rama *trunk* para posteriormente realizar el *merge* en cuestión, mediante la opción de *merge* (*team/merge*).

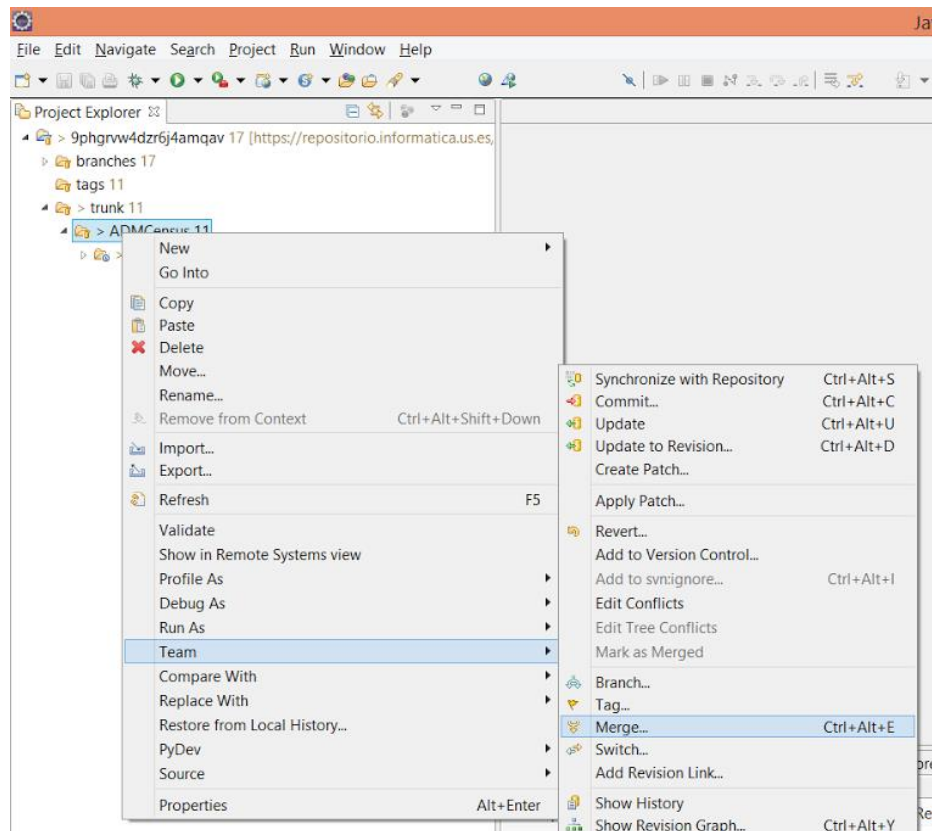


Figura 16: Menú team

A continuación, se muestra una ventana semejante a la que podemos ver en la siguiente imagen, la cual consiste en las distintitas configuraciones para ajustar valores tales como pueden ser en éste caso las ramas que van a verse involucradas.

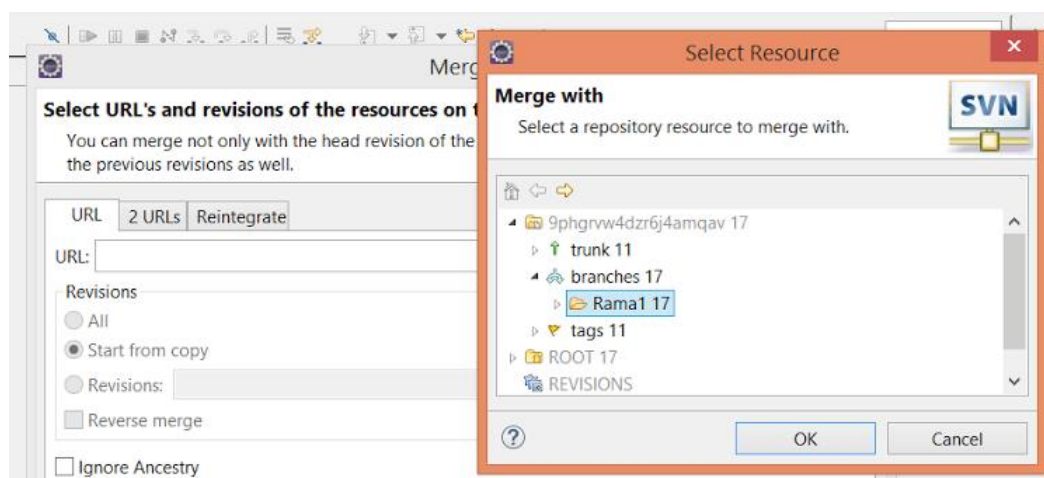


Figura 17: Rama con la que realizar el merge

Llegados a este punto, sólo nos queda realizar dicha acción, tras la cual comprobaremos, en el caso de no haber realizado una buena gestión, si surgen conflictos.

No sería necesario hacer *commit* ya que el *merge* realiza esta acción de automáticamente.

Ahora, procedemos a la creación de un *tag*, que permite el cierre de una versión estable de nuestro proyecto. Esto nos va a ayudar a poder volver a una versión anterior a la actual.

Primero procedemos a crear un *tag*, seleccionando la opción de crear Tag tal y como podemos comprobar en la imagen mostrada a continuación (Team → Tag).

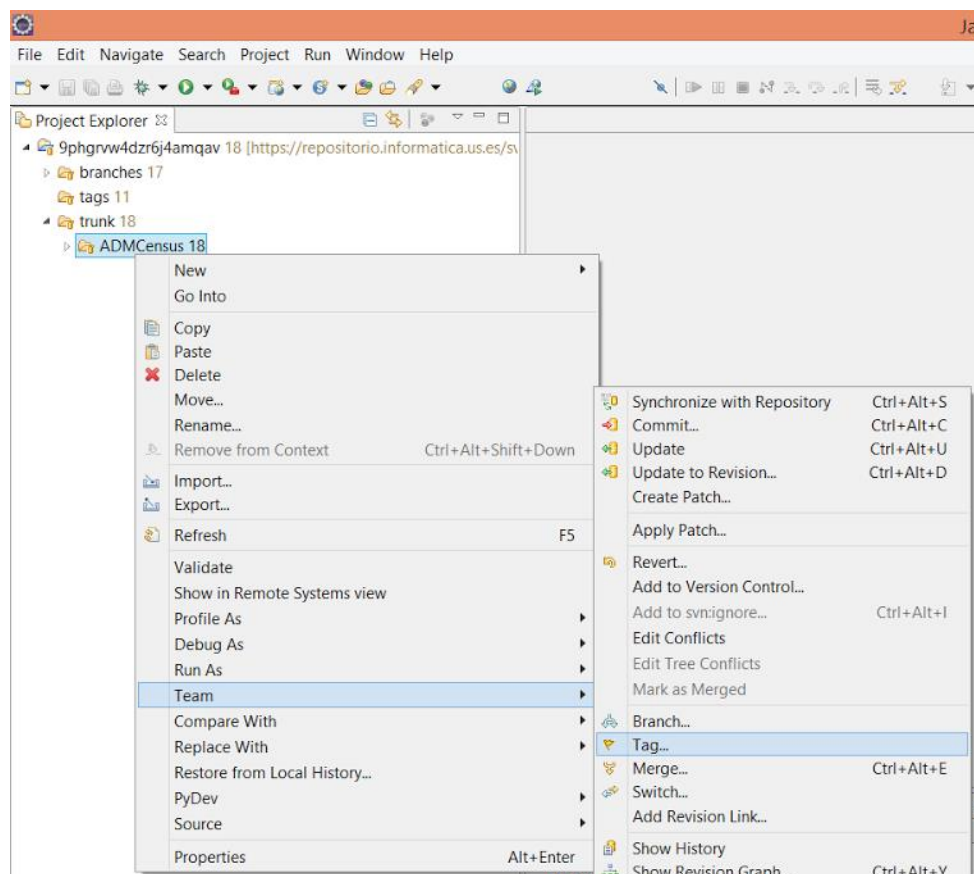


Figura 18: Menú Team

Llegados a este punto, visualizamos una ventana semejante a la mostrada a continuación, en la que debemos elegir el *tag* en el que queremos almacenar dicha versión de nuestro proyecto.

Además de esto, conviene dejar un comentario en el que se describa lo desarrollado hasta el momento, de modo que si en un futuro buscamos algo en concreto en nuestro software esto nos lo facilite.

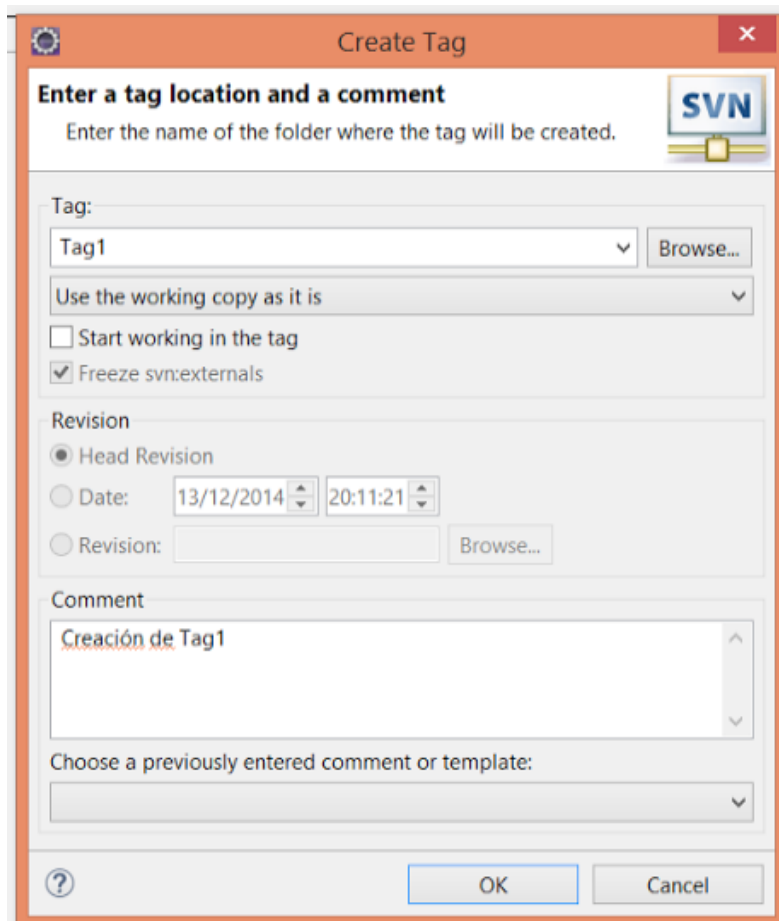


Figura 19: Creación de Tag

Finalmente, una vez realizado lo descrito anteriormente, la realización tanto del *merge* como del *tag*, ya habríamos concluido el ejercicio propuesto de manera satisfactoria.

6. GESTIÓN DE LA CONSTRUCCIÓN E INTEGRACIÓN CONTINUA

6.1. GESTIÓN DE LA CONSTRUCCIÓN

6.1.1. PROBLEMA

La gestión de la construcción consiste en convertir/construir software a partir del código fuente. En este punto se deben tomar determinadas decisiones que inciden directamente en la forma de realizar el código, por ejemplo: uso de repositorios, modularización del código, de qué forma se van a gestionar las dependencias, etc.

Podemos decir que Agora@US es un proyecto en el que la gestión de la construcción es importante, debido a la existencia de varios subsistemas y una fuerte relación entre muchos de ellos. Dichos subsistemas pueden encontrar problemas e incluso fracasar a la hora de integrarse, en el caso de que no se organicen correctamente.

6.1.2. SOLUCIÓN PROPUESTA

Uso de repositorios

Tanto los repositorios utilizados, como las decisiones tomadas para el uso de los mismos han sido explicados anteriormente en el apartado de Gestión del código fuente.

La URL del repositorio de SVN alojado en ProjETSII es la siguiente:

<https://repositorio.informatica.us.es/svn/6cn2fmqpc7fa429m2e9>

La URL del repositorio compartido alojado en GitHub es la siguiente:

<https://github.com/EGC-1415-Repositorio-compartido/repvoting.git>

En este último repositorio debemos tener en cuenta que se comenzó a poner el código funcional en la rama adminCensos, mientras que cuando se dio por finalizado el desarrollo se realizó un merge de dicho código en la rama master, a partir de dicho momento los pequeños cambios realizados para solventar problemas se han realizado sobre el repositorio de GitHub, abandonando el desarrollo en el repositorio de SVN.

División de funcionalidades

Para la construcción de nuestro subsistema se han dividido las funcionalidades en dos partes diferenciadas: internas y externas.

Las primeras funcionalidades en llevarse a cabo fueron las internas, es decir, los métodos CRUD: crear, listar, modificar y eliminar censos. Se tomó la decisión de realizar primero dichas funcionalidades, ya que eran menos susceptibles de cambios, al no ser proporcionados a otros subsistemas.

Posteriormente se realizaron las funcionalidades externas, para ello se ha generado una API con todos los métodos necesarios para obtener una comunicación exitosa con los subsistemas con los cuales nos comunicamos, o bien requieran de nuestro servicio. La decisión de aportar una API en formato Json se debe a que nos relacionamos con subsistemas que utilizan diferentes tecnologías, por lo que debíamos buscar una solución independiente de las mismas y se consideró que dicha solución cumplía dicho requisito.

Gestión de dependencias

Para solventar problemas de dependencias con librerías y APIs de terceros se ha utilizado Maven, debido a que dicho framework se integra con facilidad con el resto de tecnologías que usamos para el desarrollo de nuestro subsistema. Concretamente la versión elegida es Maven 3 mediante el cual se obtienen las librerías necesarias para la construcción a través de los archivos Pom.xml de los proyectos, y así poder gestionar las dependencias de una forma más sencilla.

Las dependencias del proyecto se pueden clasificar en dos grandes grupos:

- **Internas:** son librerías de terceros necesarias para nuestro subsistema, no son competencia de ninguno de los subsistemas que componen Agora@US. Por ejemplo proporcionamos datos en formato Json (previamente acordado con los demás subsistemas de Agora@us) y para ello utilizó la dependencia de *Codehaus*, en concreto de una librería que proporciona llamada *Jackson*, la cual provee de las llamadas para convertir un objeto Java a formato Json.
- **Externas:** son dependencias existentes con otros subsistemas de Agora@US, para poder cumplir una funcionalidad concreta. Se pueden distinguir dos grupos:
 - Consumidas: son funcionalidades, que por sus características, desarrolladas por otros subsistemas, pero que nuestro subsistema las ha necesitado para alguna determinada tarea (por ejemplo el registro de usuarios).
 - Producidas: son las funcionalidades que brindamos a otros grupos que necesitan de datos o acciones que se han desarrollado internamente (por ejemplo, el registro de usuarios que han votado o no)

Entorno de desarrollo

A la hora de comenzar el desarrollo se debe buscar algún repositorio en el que alojar nuestro código fuente, ya que cuando se trata de un equipo de desarrolladores, trabajar en local y después unir todo el código al final del desarrollo aumenta las posibilidades de fracaso a la hora de realizar la integración.

En este caso, el equipo decidió alojar su código fuente en el repositorio de ProjETSII proporcionado por la Escuela Técnica Superior de Ingeniería Informática.

Este repositorio se utilizará como un gestor de versiones, poder ramificar el desarrollo, tener *baselines*, etc. Como se puede ver en las siguientes imágenes la gestión del código fuente en primer lugar y el historial de versiones en la siguiente, vemos la gran utilidad que nos proporciona, debido a que si un equipo en el cual se está desarrollando la aplicación en local se estropea, el código queda a salvo en el repositorio.

Además, en el control de cambios, nos es de gran utilidad a la hora de subir archivos que ya han sido modificados, permitiendo comparar lo que se quiere agregar al repositorio con lo existente.

Sobre este tema de resolución de conflictos ya se ha explicado anteriormente en el apartado *Gestión del código fuente* como el grupo de desarrolladores tomó las decisiones de resolución.

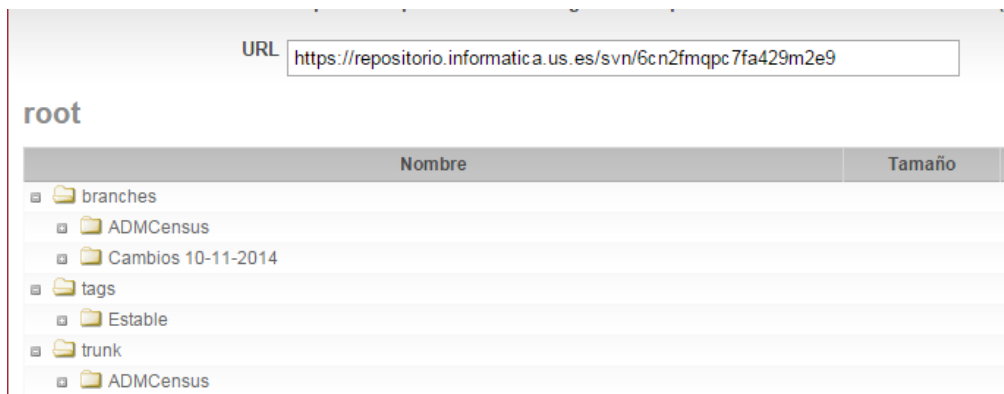


Figura 20: Vista del repositorio en ProjETSII

#		Fecha	
32	<input checked="" type="radio"/>	Domingo, 23 de Noviembre de 2014 13:28:29 +0100	
31	<input type="radio"/>	Domingo, 23 de Noviembre de 2014 13:12:56 +0100	
30	<input type="radio"/>	Domingo, 23 de Noviembre de 2014 13:08:09 +0100	Gl
29	<input type="radio"/>	Sábado, 22 de Noviembre de 2014 20:50:43 +0100	
28	<input type="radio"/>	Sábado, 22 de Noviembre de 2014 20:37:14 +0100	
27	<input type="radio"/>	Sábado, 22 de Noviembre de 2014 20:36:30 +0100	
26	<input type="radio"/>	Sábado, 15 de Noviembre de 2014 05:23:58 +0100	
25	<input type="radio"/>	Martes, 11 de Noviembre de 2014 21:12:32 +0100	
24	<input type="radio"/>	Martes, 11 de Noviembre de 2014 21:11:57 +0100	
23	<input type="radio"/>	Lunes, 10 de Noviembre de 2014 19:14:14 +0100	

[Ver diferencias](#)

[Ver todas las revisiones](#)

Figura 21: Cambios realizados en el repositorio (ProjETSII)

Este servidor está alojado en la nube por lo cual el equipo de desarrollo se despreocupa de eso.

Esta configuración del repositorio se utilizó hasta la primera integración con los demás subsistemas, una vez realizada esta integración el equipo migró los datos al repositorio compartido de GitHub.

Una vez descrito el repositorio que se utilizará, describiremos el entorno local de los desarrolladores.

Para la construcción del subsistema Creación y Administración de censos se ha basado en la tecnología JEE (Open Source).

Se ha desarrollado sobre un sistema operativo Ubuntu 14.04 de 64bits con una versión de Java 7, ya que al estudiar la compatibilidad de los frameworks utilizados, no se encontró ninguna incompatibilidad, y con el IDE de desarrollo Eclipse Luna (4.4.1).

Cada uno tiene un equipo portátil con la máquina virtual la cual contiene las herramientas necesarias para el trabajo, las cuales se detallan a continuación:

- Conectividad a internet: es totalmente necesaria debido a Maven, ya que utilizar este sin conexión a Internet, le quita todas las ventajas proporcionadas.
- Maven: herramienta para la construcción de software. En este caso, el equipo lo ha utilizado para la gestión de las dependencias y librerías del proyecto y para la creación de la estructura de la aplicación. Aunque tiene muchas funciones más, no han sido necesarias.
- Servidor Tomcat 7: servidor de aplicaciones muy extendido para el despliegue de aplicaciones basadas en JEE.

El equipo de desarrollo vió conveniente seguir el modelo de programación MVC (Modelo-Vista-Controlador) para el subsistema y para ello se apoyó en el framework Spring-MVC versión 3.2.

Como Sistema de Gestión de Base de Datos (SGBD) se utilizó MySQL 5.5 y la conexión con este lo hacía con Hibernate 4.

Para hacer el ORM entre Java y el SGBD se aprovechó las características de JPA 2.1 y su integración con Hibernate.

Una vez fueron conocidos todos los subsistemas con los que Creación/Administración de censos se comunicaba y se identificaron los servicios que debíamos ofrecer, evaluamos los cambios que se debían realizar. En el caso de que dicha comunicación afectara a nuestro modelo de datos, se volvería a repetir el ciclo ascendente anteriormente descrito. Sin embargo, en los casos en los que solo ha sido necesario realizar una comunicación mediante Json o modificar la lógica de la aplicación, se ha realizado una integración “descendente”, ya que no ha sido necesario realizar el ciclo ascendente completo.

La siguiente figura muestra el estado del entorno local de desarrollo tras haberlo configurado todas las herramientas necesarias, tanto para la integración interna como la externa.

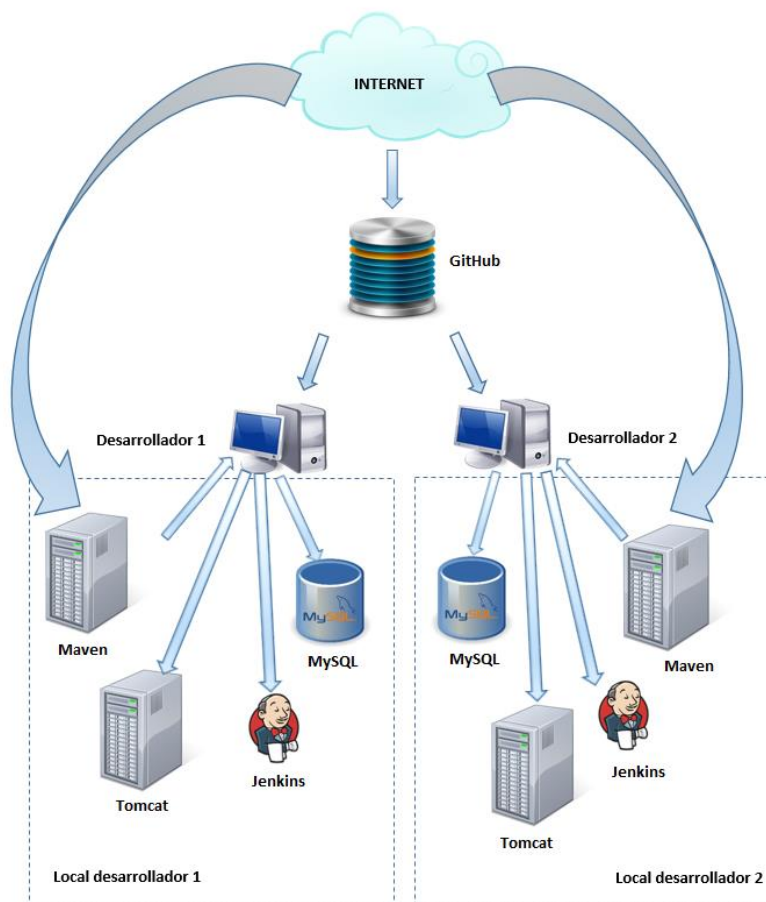


Figura 22: Entorno local de desarrollo

Descarga del código fuente

Como prerrequisitos de este paso, se debe tener un equipo con todas las tecnologías nombradas anteriormente, para así poder descargar el proyecto del repositorio y comenzar a trabajar sin más problema. Estas tecnologías necesarias se proporcionan en la máquina virtual entregada junto con esta memoria.

A continuación, vamos a describir los pasos para poder descargar en nuestro IDE eclipse el proyecto ADMCensus:

- Con eclipse arrancado, en un *Workspace* cualquiera, pulsamos Window -> Open Perspective -> Svn Repository Exploring. Aquí está listo para decirle que lea lo que hay en el repositorio.
- Ahora le damos a botón derecho new -> Repository Location... e introducimos nuestro usuario y contraseña que nos han proporcionado y la ruta del repositorio, en este caso es:

<https://repositorio.informatica.us.es/svn/6cn2fmqpc7fa429m2e9>

- Acto seguido, tendremos la estructura del repositorio en nuestro equipo:

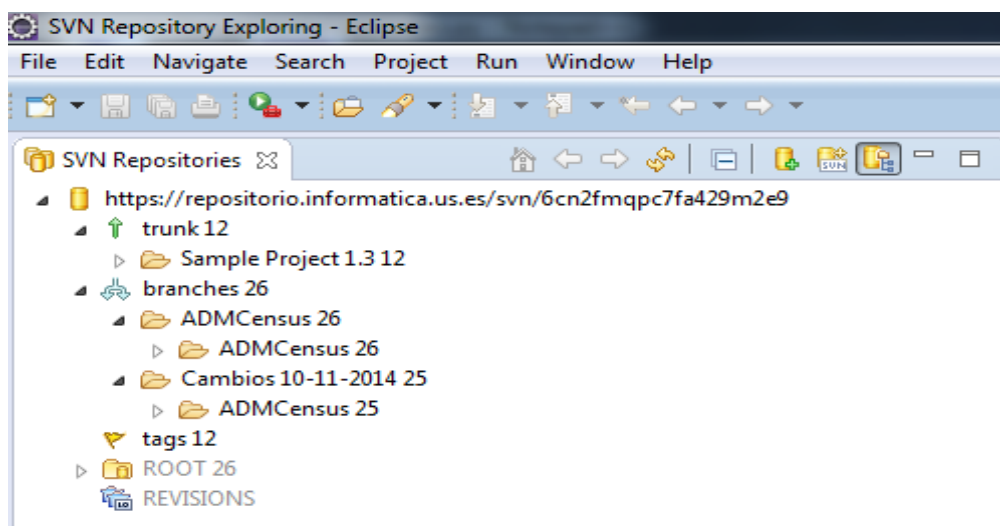


Figura 23: Estructura del repositorio

Aquí, podremos descargar la versión de rama que queramos, o bien si queremos una versión final para realizar pruebas, podremos descargar el trunk o el tag.

Importarlo mediante botón derecho sobre lo que queramos descargar (*trunk*, *branch* o *tags*) y *Check out*. Con esto, ya tendremos el proyecto en nuestro Workspace a falta de configurar la base de datos.

- La base de datos del proyecto se llama ADM Census y para que el proyecto funcione debemos tenerla creada.
- Por último, ejecutamos dos clases Java para poder tener datos funcionales. Las clases son las siguientes:
 - PopulateDatabase.java -> Crea la estructura de la base de datos. La debemos ejecutar como una clase java normal.
 - CreateCensus.java -> Creará usuarios y censos de pruebas -> La debemos ejecutar como un test JUnit.
- Ya tenemos lista la aplicación para agregar el proyecto al servidor Tomcat con botón derecho sobre Tomcat -> Add and Remove -> agregamos el proyecto -> Finish.

Construcción y ejecución

A la hora de probar nuestra aplicación en un entorno de desarrollo, tratándose de una aplicación Java, podemos hacerlo empaquetando el proyecto en un fichero WAR (Web Application Archive).

Gracias a Eclipse podemos generar dicho fichero de una forma cómoda, para ello seguiremos los siguientes pasos:

- Segundo botón sobre el proyecto
- Export
- WAR file
- Introducimos el nombre del fichero y destino

Como la base de datos tiene una estructura determinada, antes de desplegar el WAR en el Tomcat del entorno de producción, debemos crear la estructura de la base de datos. Para realizar dicha tarea, lo más cómodo es generar un script SQL en el entorno de desarrollo, mediante el cual crearemos la base de datos en el entorno de producción.

6.1.3. LECCIONES APRENDIDAS

A la hora de abordar la gestión de la construcción, al igual que en muchos otros puntos de la elaboración de un trabajo de este tipo, se debería tener claro desde el principio qué relaciones van a tener cada uno de los subsistemas con el resto, para tratar de realizar los menores cambios posibles, al mismo tiempo que disponer de vías de comunicación fluidas con los miembros de otros subsistemas.

Se debe tener en cuenta que el código elaborado no debe estar condicionado, ni mucho menos estar directamente ligado al entorno de desarrollo sobre el que se haya elaborado, es decir, se debe tratar de hacer lo más independientemente posible de la tecnología utilizada, por ejemplo proporcionando APIs.

Para la elaboración de futuros proyectos en los que haya comunicación con otros sistemas se propone que dicha comunicación se realice mediante APIs utilizando el formato JSON.

En cuanto a la elección del Sistema Operativo, la decisión de usar Ubuntu pensamos que ha sido acertada debido a la facilidad de integración con todas las herramientas que se han utilizado, tanto para desarrollo, integración continua, etc.

6.2. INTRODUCCIÓN A LA INTEGRACIÓN

6.2.1. INTEGRACIÓN INTERNA

6.2.1.1. PROBLEMA

La integración continua surge para poder detectar fallos en el desarrollo de un proyecto cuanto antes, realizando integraciones de los componentes que lo forma. Podemos decir que la integración es el proceso en el que se unen los componentes que forman un sistema. Dicha integración se ha de realizar cada cierto tiempo, descargando el código fuente de cada uno de los componentes, compilándolo en el caso de ser necesario, ejecutando pruebas y generando informes.

Llevar a cabo la integración continua requiere un gran esfuerzo, debido a que cada componente se integrará tras la aprobación del resto de componentes con los que esté relacionado. Sin embargo, si realizásemos la integración de todos los componentes al finalizar su desarrollo, podría resultar caótico.

Antes de decidir qué herramienta de integración continua elegir, debemos tener en cuenta qué tecnologías estamos utilizando, los costes del uso de dichas herramientas, si soporta el repositorio que utilicemos, etc.

6.2.1.2. SOLUCIÓN PROPUESTA

Estrategias de integración

A la hora de llevar a cabo la integración continua podemos elegir entre dos estrategias:

- **Integración por fase:** consiste en combinar todos los componentes a la vez tras finalizar la fase de diseño, a este tipo de estrategia también se le conoce por el nombre de “integración Big Bang”. Este tipo de estrategia tiene la desventaja de que los errores encontrados son difíciles de aislar, por tanto su solución es más laboriosa. Podría resultar útil en proyectos pequeños, ya que no se necesita demasiado tiempo a la hora de formar el producto final.

- **Integración incremental:** consiste en combinar los componentes progresivamente, es decir, se integra un componente y tras aprobar dicho ensamblaje se integra otro componente hasta obtener el producto final tras la integración de todos los componentes. La ventaja de esta estrategia reside en la capacidad de detectar, aislar y corregir los errores. La desventaja reside en el tiempo que se puede llegar a tardar a la hora de integrar por completo los componentes de un producto, debido a que se lleva un plan sistemático.

Tras analizar las ventajas y desventajas de las distintas estrategias, se optó por realizar una integración incremental a pesar de ser más costoso en tiempo, pero más fiable y sistemático en cuanto a la detección y solución de posibles errores.

Dentro de la integración incremental podemos distinguir tres formas de llevarla a cabo: ascendente, descendente y sandwich.

Tanto para los métodos internos como para la realización de la API para comunicarnos con los demás subsistemas, se ha seguido una integración incremental ascendente, es decir, comenzando con el modelo de dominio del subsistema asignado, creación de la base de datos, repositorios para acceder a los datos necesarios, los servicios con la funcionalidad interna requerida, las vistas y los controladores.

¿Cómo se ha realizado la integración?

Al comienzo del desarrollo del proyecto toda la integración interna se realizaba apoyándonos sobre el repositorio elegido, en nuestro caso SVN. Para ello se utilizaba el plugin anteriormente mencionado que incluye Eclipse.

Además del repositorio mencionado, se ha utilizado Maven para la automatización de tareas a nivel de código: compilar, empaquetar, verificar, instalar dependencias, limpiar archivos temporales, etc. El uso de esta tecnología se ha realizado tanto a través del plugin que proporciona Eclipse como por comandos en consola.

A medida que se fueron conociendo tecnologías, se fueron automatizando las tareas de integración continua, para ello se utilizó la herramienta Jenkins. Además del uso de dicha herramienta, como ya se ha comentado anteriormente, una vez se realizó la primera sesión de integración, se abandonó el repositorio de SVN para seguir realizando cambios sobre el repositorio de Git.

Cada desarrollador se ocupó de una función distinguiendo dos roles:

- Métodos internos: [Antonio Juan Amador Salmerón](#)
- API externa: [Guimar Fernández de Bobadilla Brioso](#)

Maven

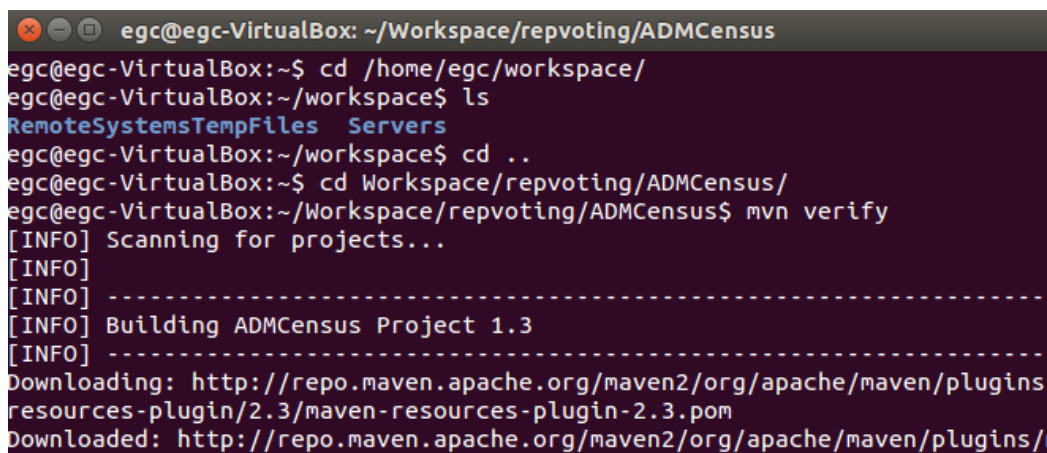
Se puede considerar la primera herramienta de integración utilizada, mediante su uso podemos verificar que los cambios descargados del repositorio no interfieran en el proyecto y podamos seguir trabajando.

A parte de para gestionar las dependencias como ya se ha comentado anteriormente, se ha utilizado esta herramienta para ejecutar ciclos de trabajo, pero para ello debemos situarnos en el mismo directorio en el que se encuentra el fichero pom.xml de nuestro proyecto.

Mediante la ejecución de metas (compile, clean,...) se pueden realizar diversas tareas que nos ayudan en el desarrollo del proyecto. Además nos permite la ejecución automatizada de test unitarios JUnit, mediante los cuales se comprueba la consistencia del proyecto.

Para ejecutar tests de JUnit, utilizamos la meta `mvn test`, indicándole el nombre y la ruta de la clase y este nos devolverá el resultado de los test ejecutados, correctos y fallidos.

En la siguiente imagen se muestra una verificación del proyecto tras haber realizado *update*.

A terminal window titled 'egc@egc-VirtualBox: ~/Workspace/repvoting/ADMCensus'. The user navigates to the workspace directory, lists files, and then runs 'mvn verify'. The output shows Maven scanning for projects, building the ADCensus Project 1.3, and downloading the maven-resources-plugin 2.3 from the Apache Maven repository.

```
egc@egc-VirtualBox: ~/Workspace/repvoting/ADMCensus
egc@egc-VirtualBox:~$ cd /home/egc/workspace/
egc@egc-VirtualBox:~/workspace$ ls
RemoteSystemsTempFiles  Servers
egc@egc-VirtualBox:~/workspace$ cd ..
egc@egc-VirtualBox:~$ cd Workspace/repvoting/ADMCensus/
egc@egc-VirtualBox:~/Workspace/repvoting/ADMCensus$ mvn verify
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building ADCensus Project 1.3
[INFO] -----
[INFO] Downloading: http://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-resources-plugin/2.3/maven-resources-plugin-2.3.pom
[INFO] Downloaded: http://repo.maven.apache.org/maven2/org/apache/maven/plugins/
```

Figura 24: Verificación del proyecto

Entre otros comandos que nos permite ejecutar Maven, podemos destacar el que nos genera el WAR de nuestro proyecto, ya que ha sido uno de los que hemos utilizado.

```
mvn war:war
```

Jenkins

Entre las herramientas de integración continua más utilizadas, podemos destacar algunas como Bamboo, Hudson, Continuum o Jenkins. Dichas herramientas permiten ejecutar tareas, apoyándose en otras como pueden ser Ant o Maven para realizar compilaciones, pruebas e informes del código.

En nuestro caso se ha optado por Jenkins, ya que se trata de una herramienta muy utilizada en la actualidad, con una gran comunidad de usuarios, soporta los repositorios utilizados mediante plugins, se puede montar en un servidor local, permite utilizar de una forma sencilla gran cantidad de herramientas, es OpenSource, además de otras ventajas.

Jenkins: Configuración

Podemos decir que la instalación de Jenkins sobre un entorno Linux, concretamente Debian es sencilla, como podemos ver a continuación, basta con incluir el repositorio de Jenkins, actualizar nuestro repositorio local e instalarlo.

```
deb http://pkg.jenkins-ci.org/debian binary/  
  
sudo apt-get update  
  
sudo apt-get install jenkins
```

El único cambio significativo que se ha realizado durante dicha instalación ha sido el puerto, ya que utiliza por defecto el 8080, siendo este utilizado por el Tomcat de Eclipse para las pruebas de desarrollo. Para resolver dicho conflicto se le asignó a Jenkins el puerto 8090.

Para que Jenkins realice sus funciones debemos tener las tecnologías Java y Maven instaladas en la máquina en la que vayamos a ejecutar dicha herramienta, además debemos instalarle el plugin de Git para poder manejar el repositorio.

Jenkins: Automatización

Para llevar a cabo la automatización del proyecto mediante Jenkins podemos optar por dos opciones, realizando la integración cada vez que se realice un *pull* en el repositorio, o bien ejecutándolo cada determinado periodo de tiempo.

En nuestro caso se ha optado por la segunda opción, ya que para poder llevar a cabo la primera debemos tener las claves públicas y privadas del repositorio, al tratarse de otros miembros los que lo han creado no disponemos de las mismas.

A continuación se muestra el comando mediante el cual le indicamos a Jenkins que debe ejecutar cada día la integración continua de nuestro sistema.

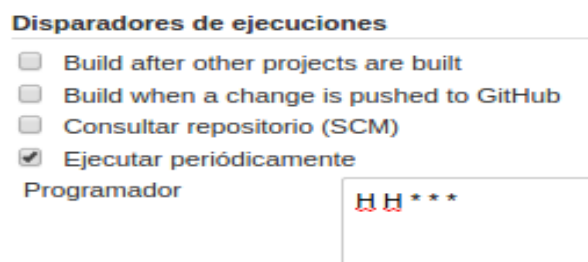


Figura 25: Ejecución periódica de Jenkins

La decisión de ejecutarlo cada día ha sido para realizar pruebas y porque se consideró que es un tiempo adecuado para detectar errores a tiempo.

Jenkins: Notificaciones de error

En caso de que una tarea falle, podemos automatizar la notificación a los desarrolladores, esto podemos hacerlo en la configuración del proyecto, dentro de otras acciones, añadir notificación. En la siguiente imagen podemos ver cómo se crea una notificación:

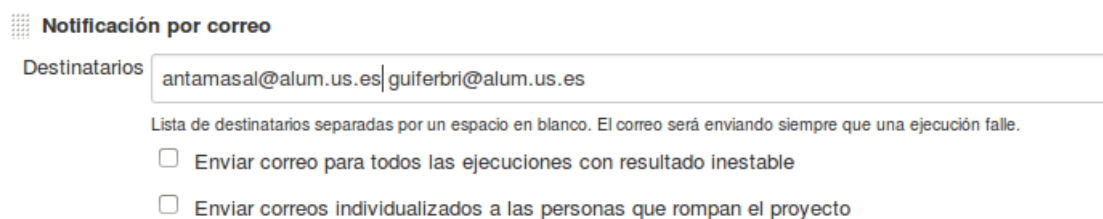


Figura 26: Notificaciones de correo

Aparte de esto, también podríamos almacenar el resultado y generar informes con los mismos.

Generación automatizada de documentación

Tras la primera revisión del proyecto y haber asistido a las presentaciones de otros grupos, se decidió utilizar una herramienta de generación de documentación de forma automática. Dichas herramientas se utilizan para asegurarnos de que el código fuente generado está correctamente comentado, es decir, no nos dejamos métodos o funciones importantes sin comentar. Generar dicha documentación puede ser muy importante a la hora de retomar un proyecto, o bien para que otro desarrollador pueda entender con mayor facilidad nuestro código.

En nuestro caso al tratarse de un proyecto Java, la mayoría de herramientas se basan en el estándar Javadoc para realizar dicha labor, entre las herramientas o plugins podemos destacar JAutodoc (plugin para eclipse) o Doxygen.

Finalmente se decidió elegir Doxygen porque no es dependiente del lenguaje, ya que soporta C, C+, Java, Python, entre otros.

Antes de comenzar a utilizar dicha herramienta debemos instalarla, para ello ejecutaremos el siguiente comando en el terminal de Ubuntu.

```
sudo apt-get install doxygen
```

Una vez hecho esto ejecutamos el comando **doxygen** y este creará un archivo de configuración en el directorio actual. En dicho archivo entre otras cosas podemos definir el título de la documentación, el lenguaje a documentar, etc.

Para utilizar esta herramienta con Jenkins disponemos de dos opciones, instalando el correspondiente plugin o bien mediante comandos. En nuestro caso, se ha optado por hacer uso de Doxygen mediante comandos en Jenkins, para ello debemos añadir una línea en la tarea que descarga y compila nuestro proyecto.

Doxygen necesita un archivo de configuración para saber que documentación generar, como interpretar ciertos comentarios, donde guardar la documentación, etc.

Para crear dicho archivo, ejecutamos en la ruta que queramos crear el archivo el siguiente comando:

```
doxygen
```

Una vez configurado el archivo, nos ubicamos en la carpeta donde estén las clases a comentar y ejecutamos el siguiente comando.

```
var/lib/Jenkins/jobs/EGC/workspace/census/ADMCensus/src/main/java/services  
doxygen ruta_fichero_configuracion
```

Para que Doxygen genere la documentación de forma automática debemos seguir el estándar Javadoc (ver artículo de [Wikipedia](#)) a la hora de insertar los comentarios en el código.

Como veremos en el [EJERCICIO 2](#) la documentación generada por Doxygen es en formato HTML.

Revisión del código

Consiste en medir la calidad del código elaborado, mediante herramientas que facilitan esta tarea. Esta calidad se puede regir bajo diversas reglas o normativas (como por ejemplo Madeja en la Junta de Andalucía). La finalidad de realizar dicha tarea es la de informar acerca de la complejidad, duplicidad, legibilidad o falta de comentarios en el código fuente. La gran ventaja que nos aporta este tipo de herramientas es que nos facilita la tarea de mejorar nuestro código. Como desventaja podríamos indicar que el programador debe adaptarse a los patrones utilizados.

En nuestro caso hemos utilizado la herramienta SonarQube ya que se puede integrar en Jenkins con facilidad, y es una herramienta muy extendida.

Para la instalación de SonarQube, como podemos ver a continuación, debemos incluir el siguiente paquete en el repositorio de Ubuntu.

```
deb http://downloads.sourceforge.net/project/sonar-pkg/deb binary/
```

Una vez incluido dicho paquete, debemos actualizar el repositorio de Ubuntu e instalar SonarQube con los siguientes comandos:

```
sudo apt-get update  
sudo apt-get install sonar
```

Tras ejecutar estos comandos ya tendríamos instalado SonarQube, para poder utilizarlo debemos iniciarlo mediante el siguiente comando.

```
sudo service sonar start
```

SonarQube en su instalación por defecto se arranca en el puerto 9000, por lo que si queremos acceder al mismo debemos hacerlo mediante la siguiente URL:

<http://localhost:9000>

Por defecto trae integrada una pequeña base de datos para almacenar los resultados de código que realice, aunque nos permite la posibilidad de cambiar dicha base de datos por otra que deseemos. En nuestro caso no ha sido necesario cambiar dicha base de datos al tratarse de un proyecto pequeño.

Para indicarle a SonarQube que debe analizar un proyecto, como en nuestro caso se trata de un proyecto Java, podemos hacerlo mediante la herramienta Maven, como se puede ver a continuación:

```
mvn sonar:sonar
```

Para automatizar dicho proceso, debemos incluir el comando anterior en la tarea que Jenkins vaya a ejecutar.

Tras ejecutar la tarea de Jenkins, si accedemos a la URL local de SonarQube anteriormente mencionada, en la pestaña “*All Projects*” (como podremos ver en el EJERCICIO 2) podemos ver la instancia de nuestro proyecto, en nuestro caso ADMCensus, si accedemos a este podremos ver el análisis del código que ha realizado SonarQube.

Principios de la integración continua:

En base a los 10 principios de Martin Fowler para hacer una integración continua perfecta, evaluaremos cuales de éstos cumple la integración de nuestro subsistema:

✓ Mantener un único repositorio de código fuente: sí, todo el código se encuentra en ProjETSII, repositorio anteriormente descrito y en el compartido de GitHub.

✓ Automatizar la construcción del proyecto: sí, con la herramienta Jenkins (detallada anteriormente) conseguimos una automatización del proyecto.

✓ Hacer que la construcción del proyecto ejecute sus propios tests: sí, utilizando la herramienta de JUnit, se automatizan los test cada vez que se hacía un *mergeo* para comprobar su correcta funcionalidad.

X Entregar los cambios a la línea principal todos los días: no, puesto que nos parece excesivo dicho *mergeo*. El equipo decidió llevar cambios a la rama principal una vez por semana o al finalizar una funcionalidad completa.

✓ Construir la línea principal en la máquina de integración: sí, el equipo lo primero que realizó fue una línea base del proyecto para que todas las ramas extendiera de la principal, asegurando así un ramificado con la coherencia de funcionalidad adecuada.

✓ Mantener una ejecución rápida de la construcción del proyecto: sí, ya que tanto el despliegue mediante el fichero *.war* ó en la máquina de desarrollo, el tiempo máximo para construir el proyecto es de 7-9 minutos.

✓ Probar en una réplica del entorno de producción: sí, el equipo tiene una máquina de pre-producción con las herramientas instaladas simulando las que el cliente tendría.

✓ Hacer que todo el mundo pueda obtener el último ejecutable de forma fácil: sí, pues el equipo proporcionará un fichero .war con el cual solo lo tendrá que desplegar con el servidor de aplicaciones Tomcat 7 y el proyecto estará preparado para su utilización.

✓ Publicar qué está pasando: si, el equipo en cada *commit* o *merge* del repositorio, explicaba lo realizado en comentarios

✓ Automatizar el despliegue: aunque al comienzo del desarrollo se comenzó a realizar de una forma manual, tras conocer la herramienta de Jenkins se realizó con esta.

6.2.1.3. LECCIONES APRENDIDAS

Desde el comienzo del proyecto se debería tener claro qué repositorio se va a utilizar, ya que en nuestro caso se decidió migrar de SVN a GIT con lo cual se perdió tiempo en hacer dicha migración, además del tiempo invertido en trasladar el historial de versiones de un repositorio a otro.

Para mejorar el desarrollo del proyecto se deben marcar unos hitos al comienzo del proyecto, revisables durante el desarrollo del mismo, de esta forma se agilizaría el desarrollo, pruebas, integración, etc.

En el caso de proyectos pequeños y con pocos desarrolladores, como nuestro caso, pensamos que el uso de herramientas como Jenkins, SonarQube o Doxygen no tiene demasiada utilidad. Sin embargo, si se deciden utilizar este tipo de herramientas, lo mejor es hacer uso de ellas desde el comienzo del proyecto, ya que de esta forma la curva de aprendizaje es más suave.

La notificación de errores de forma automática, que nos aporta Jenkins, puede resultar molesta en fases tempranas del desarrollo, ya que puede saturarnos de mensajes. Por lo cual consideramos que se automaticen dichas notificaciones en fases del desarrollo avanzadas.

En cuanto a la herramienta Maven, nos ha sido de mucha utilidad para la gestión de librerías externas, puesto que si tenemos que llevarnos el proyecto de un entorno a otro, no tendríamos que preocuparnos de las dependencias del proyecto.

6.2.2. INTEGRACIÓN EXTERNA

6.2.2.1. PROBLEMA

La integración externa surge cuando queremos que dos o más sistemas se unan para crear uno solo que englobe a tantos subsistemas como sea necesario.

Esta integración debe ser independiente al tipo de tecnología usada en cada sistema, forma de trabajar, etc.

Esto da pie a múltiples estrategias de integración y varias formas de trabajar aunque todas persiguiendo el correcto funcionamiento del sistema general.

Para esto, es aconsejable utilizar herramientas, como Jenkins, para automatizar este tipo de tareas y que te indique cuando la integración ha fallado, porque y a causa de que subsistema.

6.2.2.2. SOLUCIÓN PROPUESTA

Tras la propuesta de creación de un repositorio común por parte de un miembro del grupo de otro subsistema, en concreto Autenticación, nuestra propuesta para integrarnos con los distintos subsistemas con los que estamos relacionados y de forma general para el resto, teniendo en cuenta los principios de integración de Martin Fowler es la siguiente:

✓ Mantener un único repositorio de código fuente → Se cumple ya que se utiliza el repositorio común, en el cual cada subsistema tiene una rama asignada en la que trabajar. Dicho repositorio se encuentra alojado en Git en la siguiente dirección: <https://github.com/EGC-1415-Repositorio-compartido/repvoting>

✓ Automatizar la construcción del proyecto → Gracias a la herramienta de Jenkins, conseguimos automatizar la construcción de los proyectos implicados.

✓ Hacer que la construcción del proyecto ejecute sus propios tests → Cada subsistema realizará test unitarios para comprobar la correcta funcionalidad y detectar posibles errores.

X Entregar los cambios a la línea principal todos los días → No se cumple la realización de “push” cada día, pero sí siempre que se añada una nueva funcionalidad válida o se realicen cambios.

✓ Construir la línea principal en la máquina de integración → En una máquina, ir haciendo los *merge* de manera secuencial, es decir, cada rama por separado. De esta forma en dicha máquina se irá probando cada combinación de las ramas hasta completar la construcción, que quedará alojada en la máquina.

✓ Mantener una ejecución rápida de la construcción del proyecto → Gracias a Jenkins, tras preparar un script y configuración de este, la ejecución de todos los proyectos se hace de manera rápida y fácil.

✓ Probar en una réplica del entorno de producción → Mediante una máquina virtual, se simulará en el entorno donde se hará uso de la aplicación Agora@US para hacer pruebas de rendimiento, fiabilidad y funcionalidad de la aplicación en un estado estable y “final”. Entendiendo como final que responde correctamente a la funcionalidad requerida a falta de pruebas en el entorno donde se usará.

X Hacer que se pueda obtener el último ejecutable de forma fácil → No se cumple debido a la multitud de tecnologías distintas con las que está construido cada subsistema, no habrá un único ejecutable.

✓ Publicar qué está pasando → Se cumple ya que todos los desarrolladores tienen acceso a cada subsistema y pueden ver los fallos, comentarios de las subidas, etc., en definitiva, se tiene un estado actualizado del proyecto en general.

✓ Automatizar el despliegue → Cada subsistema generará scripts y artefactos de forma que para realizar la integración en los distintos entornos (desarrollo, pruebas y producción) se realice el despliegue de forma sencilla con Jenkins.

Entorno de integración

Cada subsistema deberá llevar lo necesario para que su proyecto arranque y permita integrarse con los otros subsistemas con los que esté relacionado. En nuestro caso, Creación/Administración de Censos, será necesario la base de datos MySQL 5.5, servidor Tomcat e IDE de desarrollo Eclipse.

Una vez se haya puesto todo en la máquina, la cual será la utilizada como máquina de integración (5º principio de Martin Fowler: *Construir la línea principal en la máquina de integración*), se procederá a integrar cada subsistema de forma ordenada y coherente. No se podrá integrar un nuevo subsistema hasta que el anterior no haya sido probado y comprobado el correcto funcionamiento.

Orden de integración (general)

El orden de integración propuesto es, en nuestra opinión, según la complejidad de integración, dejando para el final los subsistemas que no requieren integración y en primer lugar el subsistema el cual necesita integrarse o es integrado por una gran cantidad de subsistemas. Es decir, se intentará montar la base funcional de la aplicación como la autenticación, creación de votaciones y censos y por último se dejará la parte más visual como el subsistema de visualización. Con este orden se pretende dejar la aplicación tras la integración con una mínima funcionalidad.

[Autenticación](#)

[Creación/Administración de censos](#)

[Creación/Administración de votaciones](#)

[Cabina de votación](#)

[Almacenamiento de votos](#)

[Sistema de modificación de resultados](#)

[Verificación](#)

[Recuento](#)

[Frontend de Resultados](#)

[Visualización de resultados](#)

[Deliberaciones](#)

Orden de integración para Creación/Administración de censos

[Autenticación](#)

[Creación/administración de votaciones](#)

[Cabina de votación](#)

[Deliberaciones](#)

Estética de la aplicación

Como cada grupo tendrá su estilo, podríamos votar el que más guste y adaptar todos los subsistemas a ese. En caso de que no se haya desarrollado ningún estilo, podrán votar igualmente para poder aplicarlo al suyo, de esta forma, los subsistemas no desentonarán.

Forma de integración

La forma real de trabajar ha sido la siguiente:

Cada subsistema ha creado una rama en el repositorio Git anteriormente indicado y a medida que su código esté finalizado, se realizarán *merges* a la rama *master*.

Como ya estamos en la integración externa, se ha realizado, testeado y actualizado en rama (*adminCensos*) en Git y a continuación se llevaba los cambios a la rama *master*. Esta actualización se ha realizado mediante el uso de Eclipse a través del plugin añadido a dicho IDE. En la siguiente imagen se muestra un ejemplo de lo que acabamos de explicar. El *Commit* es para la actualización de un menú del proyecto y se realizarían los siguientes pasos: hacer clic con el botón derecho sobre el proyecto (o si es un solo archivo el modificado también valdría) → Team → **Commit** y aparecerá la ventana siguiente para indicar el archivo a actualizar y un comentario sobre el cambio realizado.

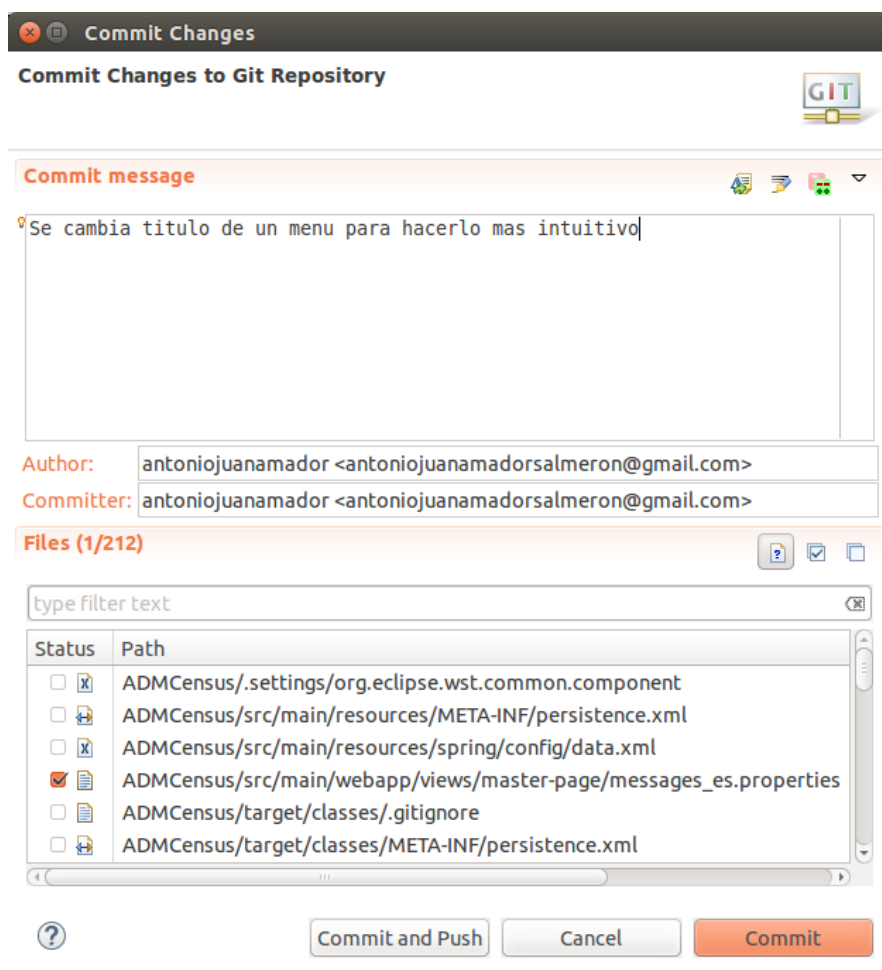


Figura 27: Commit en local

Esto habrá realizado la actualización en el repositorio local. Para llevar los cambios al repositorio común hay que realizar un *push* de manera similar al *commit*: hacer clic con el botón derecho sobre el proyecto (o si es un solo archivo el modificado también valdría) → Team → **Commit and Push**. Hacer clic en “Ok” y si no ha habido ningún problema, como podría ser la generación de un conflicto, se habrá subido el código y estará listo para ser utilizado por otros subsistemas.

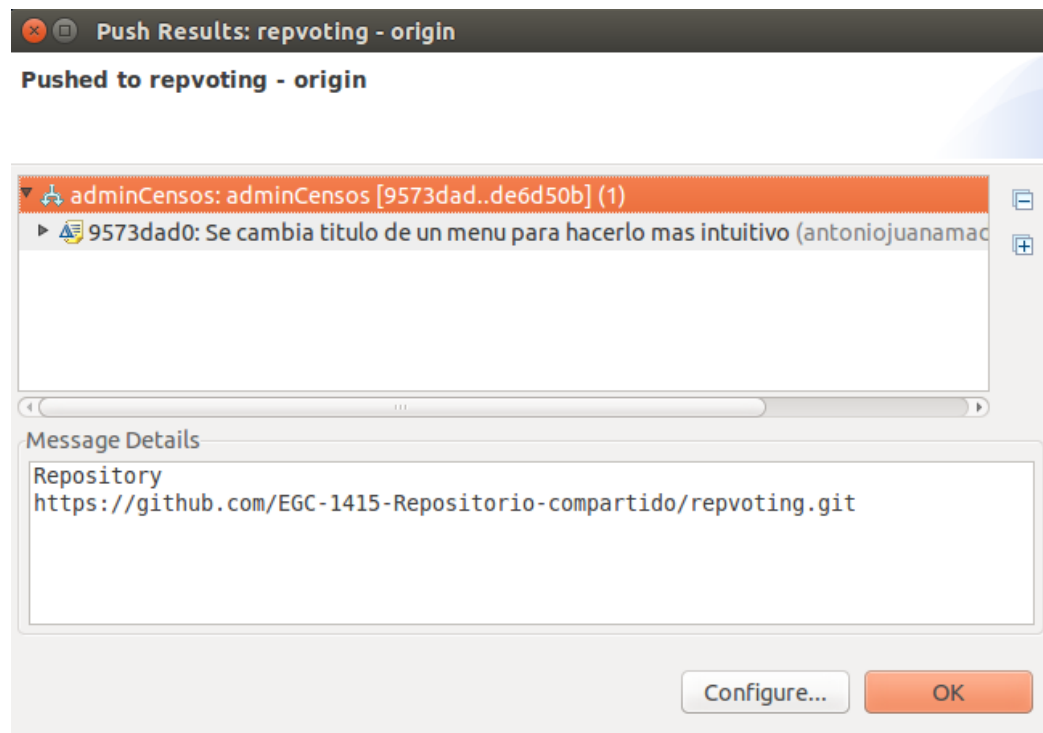


Figura 28: Commit and Push

6.2.2.3. LECCIONES APRENDIDAS

Integrar tantos subsistemas engloba muchísimas tareas (descargar código del repositorio, generación de paquetes, despliegue en servidor, etc.) por lo que nos decidimos a utilizar Jenkins. Con esta herramienta, solo teníamos que hacer una configuración inicial y a partir de ahí, podíamos automatizar cada cierto tiempo su ejecución de todas las tareas programadas.

La decisión de hacer la comunicación externa mediante APIs nos parece acertada, debido a que los resultados se devuelven en un formato estándar, en este caso JSON, lo cual hace que sea independiente de las tecnologías de otros subsistemas evitando así tener que preocuparnos de incompatibilidades tecnológicas.

A la hora de realizar un sistema y dividirlo en varios subsistemas, debe ser primordial definir las funcionalidades de cada grupo desde un primer momento, ya que si no lo hacemos, podemos llegar a conflictos en un nivel de desarrollo avanzado, siendo los costes de la solución elevados.

Cuando se han realizado las integraciones, han salido a la luz muchos problemas en los que las causas han sido la despreocupación por parte de algunos grupos de facilitar la integración con los grupos que se relaciona.

La falta de comunicación ha sido causa de muchos problemas, porque creemos que para que tantas partes puedan integrarse con los mínimos problemas posibles, debe haber un canal de comunicación fluido.

Por último creemos que sería de utilidad que un grupo o subsistema se encargase de realizar tareas de coordinación para coordinar al resto de grupos en exclusiva, es decir, facilitar vías de comunicación e integración, solventar determinados conflictos, etc.

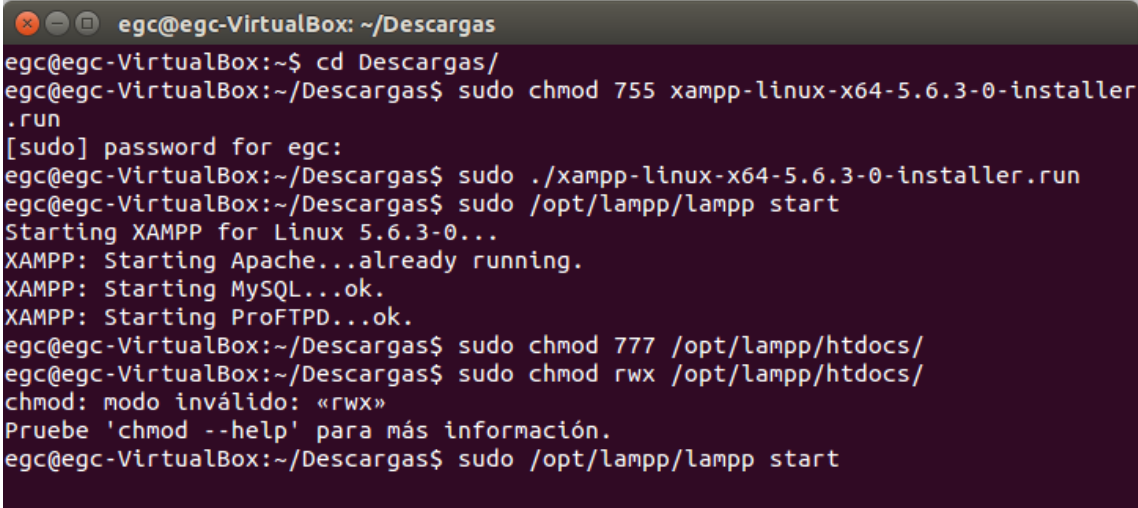
6.2.3. EJERCICIO 1

“Realizar la integración con el subsistema Deliberaciones, para poder realizar algún tipo de cambio en el código, haciendo uso de la máquina virtual proporcionada. Comprobar que se ha realizado correctamente.”

Resolución

El primer paso es preparar en el entorno de trabajo como en el apartado anterior (*Gestión del código fuente*) se indica. Es decir, iniciar MySQL (instalado con Xampp) y abrir Eclipse. Una vez realizado esto nos encontraremos algo similar a la siguiente imagen (en el caso de no haber integrado aún ningún subsistema)

Nota: para iniciar el servicio Xampp completo (Apache, Tomcat y MySQL) basta con poner el siguiente comando en un terminal: `sudo /opt/lampp/lampp start`



```
egc@egc-VirtualBox: ~/Descargas
egc@egc-VirtualBox:~$ cd Descargas/
egc@egc-VirtualBox:~/Descargas$ sudo chmod 755 xampp-linux-x64-5.6.3-0-installer.run
[sudo] password for egc:
egc@egc-VirtualBox:~/Descargas$ sudo ./xampp-linux-x64-5.6.3-0-installer.run
egc@egc-VirtualBox:~/Descargas$ sudo /opt/lampp/lampp start
Starting XAMPP for Linux 5.6.3-0...
XAMPP: Starting Apache...already running.
XAMPP: Starting MySQL...ok.
XAMPP: Starting ProFTPD...ok.
egc@egc-VirtualBox:~/Descargas$ sudo chmod 777 /opt/lampp/htdocs/
egc@egc-VirtualBox:~/Descargas$ sudo chmod rwx /opt/lampp/htdocs/
chmod: modo inválido: «rwx»
Pruebe 'chmod --help' para más información.
egc@egc-VirtualBox:~/Descargas$ sudo /opt/lampp/lampp start
```

Figura 29: Iniciar servicio Xampp

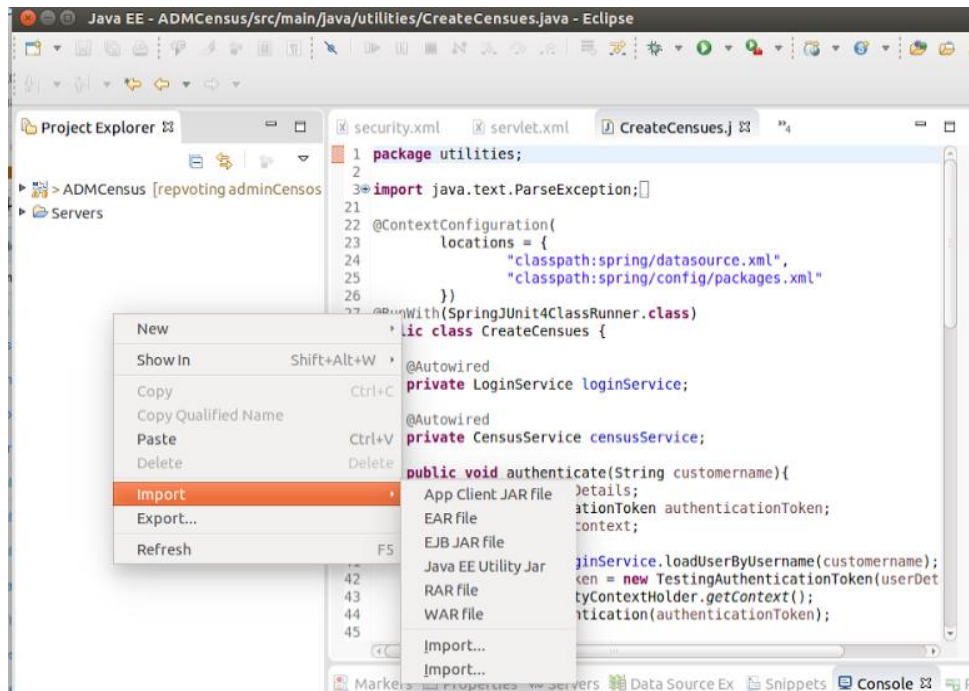


Figura 30: Menú contextual de la ventana Project Explorer

Para integrar el subsistema *Deliberaciones* lo primero que hay que realizar es descargarnos el proyecto del repositorio común donde está alojado. Para ello, en Eclipse, importamos el proyecto desde Git (File → Import → Git → Projects from Git)

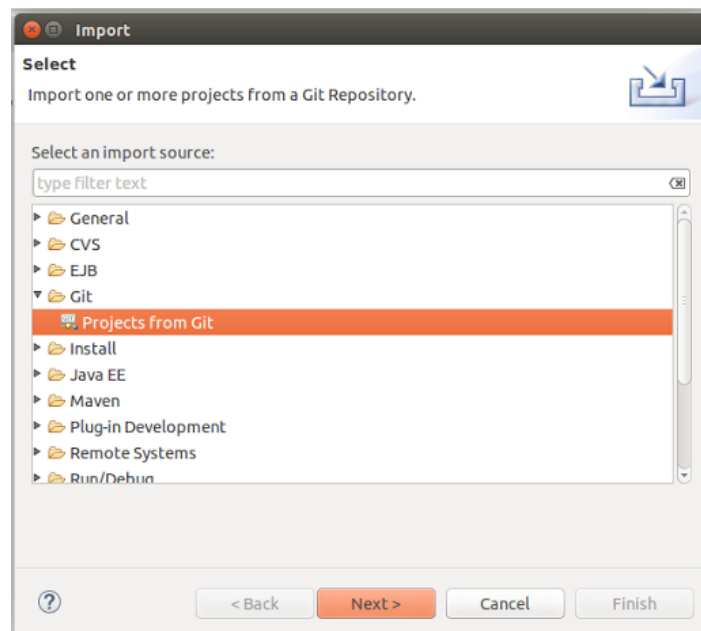


Figura 31: Ventana import

Hacemos clic en “Next” y a continuación en “Clone URI”

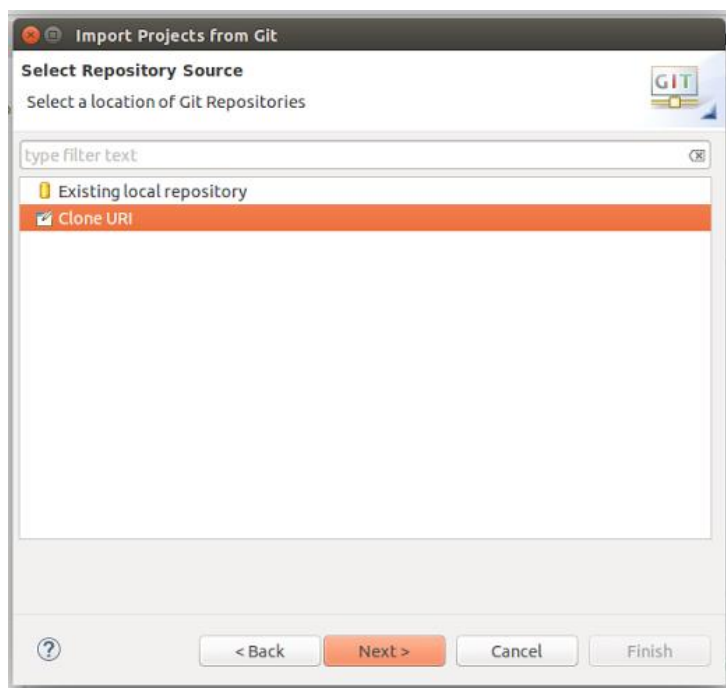


Figura 32: Ventana Git

En el menú que nos aparecerá hay que indicar la URL del repositorio: <https://github.com/EGC-1415-Repositorio-compartido/repvoting.git> y el Usuario y Contraseña con los cuales estamos registrados en GitHub para autenticarnos.

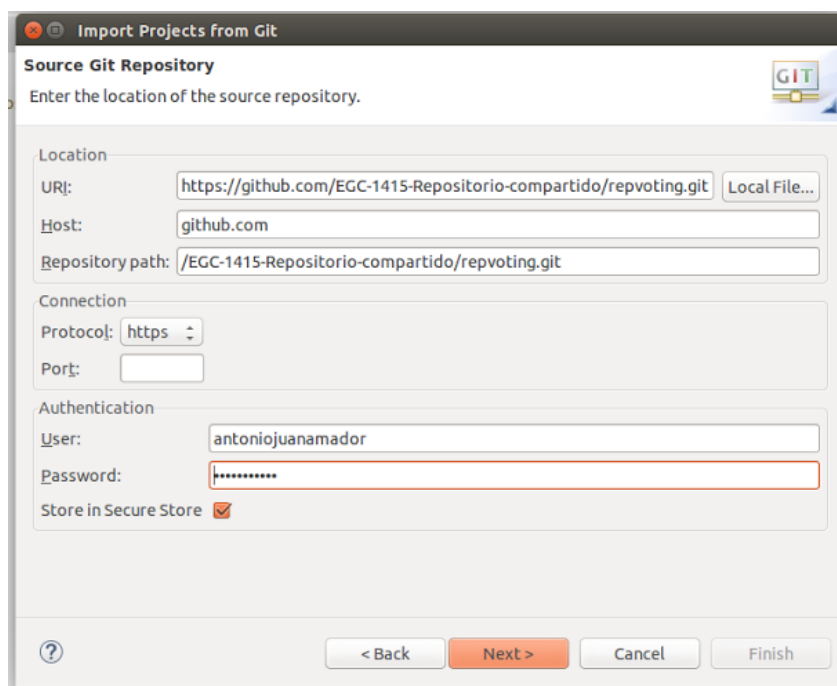


Figura 33: importar proyecto Git

Hacemos clic en “Next” y aparecerá un nuevo menú similar al siguiente. En dicho menú están todas las ramas que se encuentran en el repositorio indicado. Volvemos a hacer clic en “Next”.

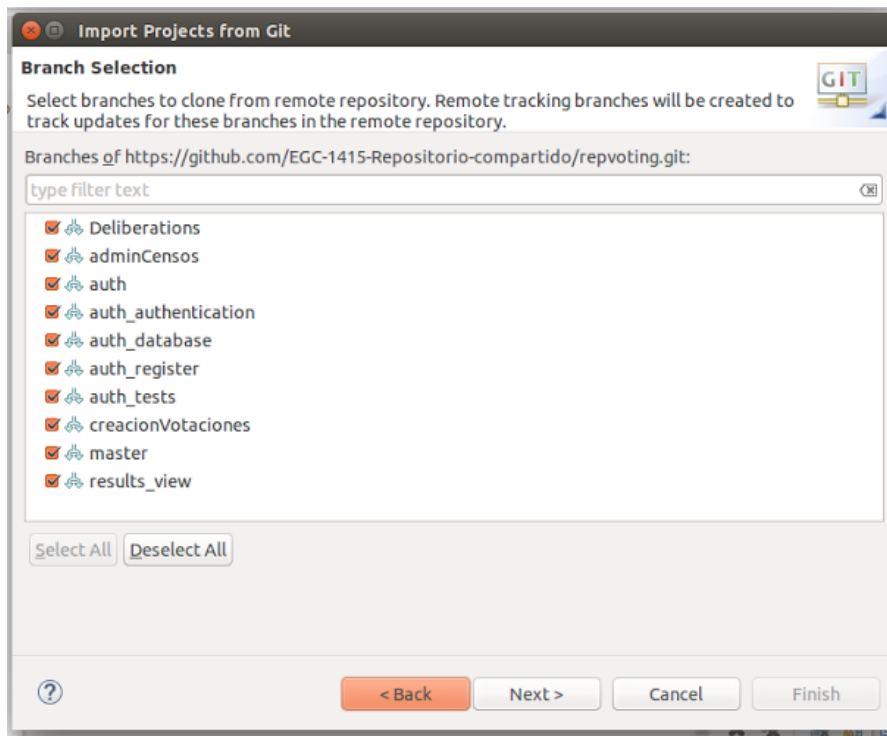


Figura 34: Vista de todas las ramas de un repositorio

En el siguiente menú seleccionamos la rama *Deliberations* en “Initial branch” y “Next”.

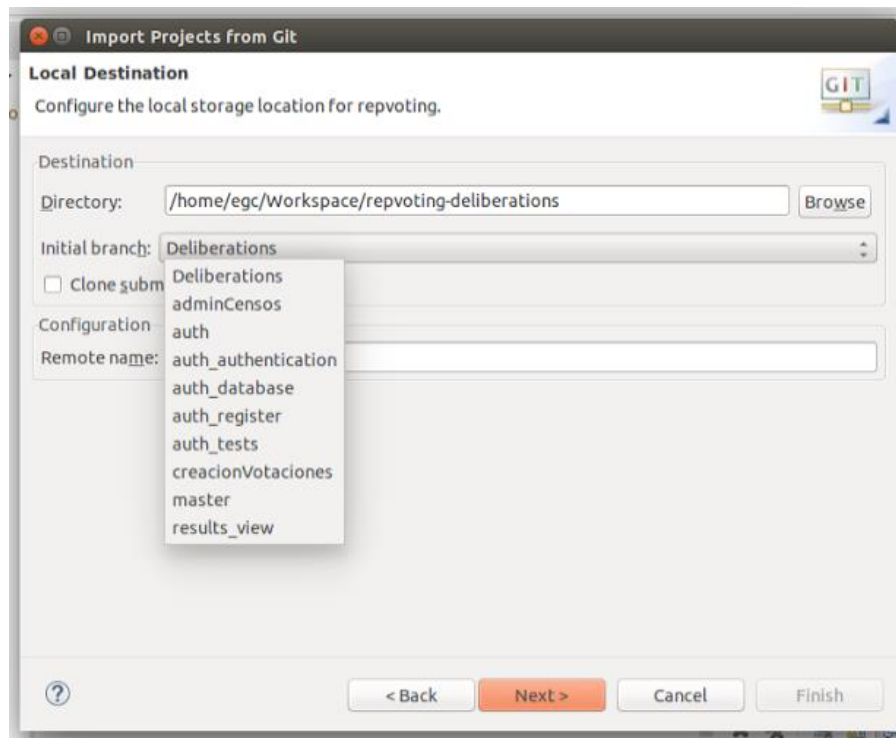


Figura 35: Selección de la rama a importar

Hacemos clic en “Import existing projects” lo cual indica que es un proyecto ya existente, y volvemos a hacer clic en “Next”.

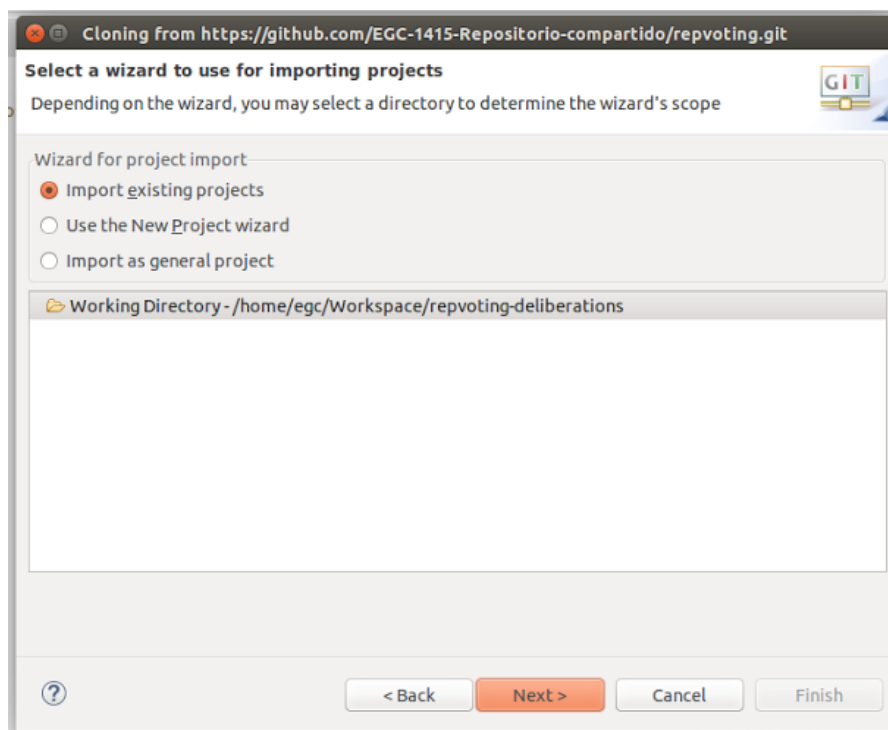


Figura 36: Importar proyecto existente

Aparecerá un nuevo menú para confirmar que el proyecto seleccionado y la rama son las correctas. Hacer clic en “Next” para continuar y ya estará el proyecto en tu *WorkSpace*.

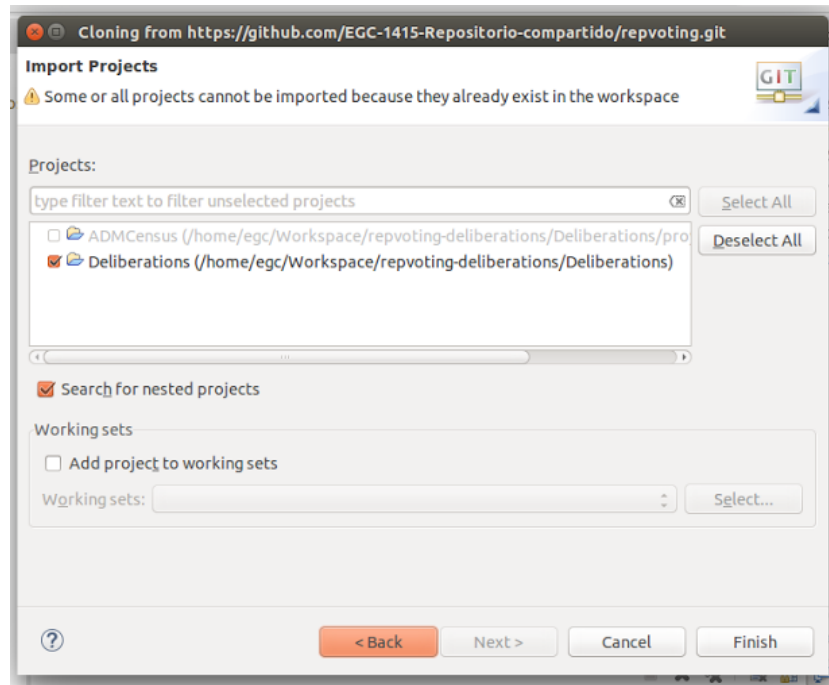


Figura 37: Confirmar proyecto a importar

Tras realizar estos pasos tu *Workspace* debe quedar similar a la siguiente imagen:

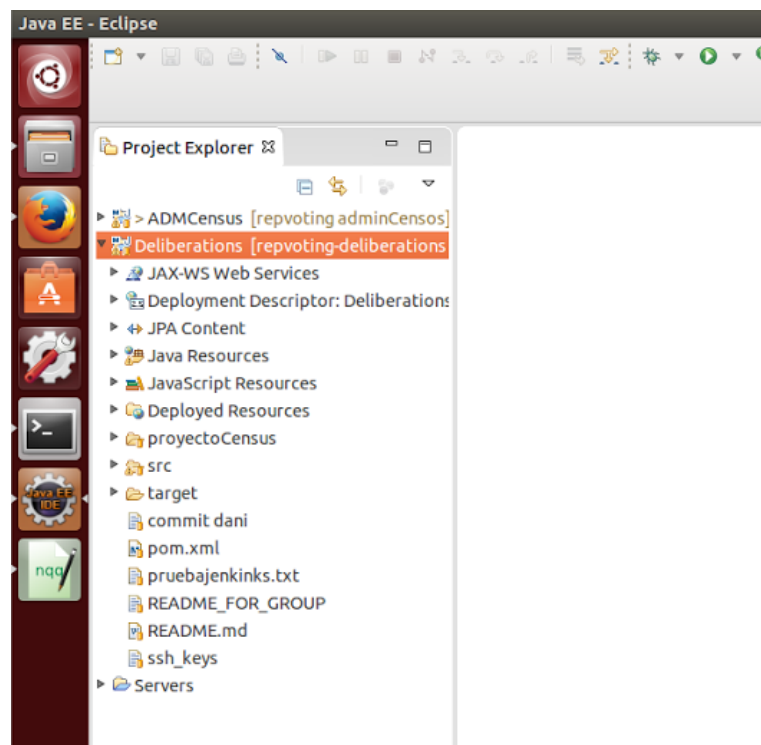


Figura 38: Vista del workspace

Para la segunda parte del ejercicio, comprobar que el subsistema *Deliberaciones* está integrado correctamente con *Administración/Creación de censos* (ADM Census), habrá que arrancarlo y confirmar que las llamadas entre ambos se realizan de la forma esperada. Es necesario los siguientes pasos previos a arrancar el servidor Tomcat:

Crear base de datos llamada *Deliberaciones*

Popular la base de datos: abrir el archivo `PopulateDatabase.java` (`src\main\java\utilities`) y ejecutar como Java Application (`PopulateDatabase.java` → `Run as Java Application`). Esto creará la estructura de la base de datos del proyecto y persistirá algunos objetos si los tiene.

Si todo ha ido bien, podemos arrancar el servidor Tomcat y realizar las llamadas requeridas entre ambos subsistemas de forma exitosa.

6.2.4. EJERCICIO 2

“Crear una tarea en Jenkins para automatizar la integración, se deberá realizar diariamente, con el subsistema de Creación y Administración de Censos:

- *Desplegar el proyecto en Tomcat 7 (desde la rama Master)*
- *Generar documentación con Doxygen de las clases CensusServices.java*
- *Analizar el código con SonarQube.”*

Resolución

- Lo primero que vamos a hacer es automatizar la integración del subsistema, para ello creamos una nueva tarea como proyecto de estilo libre y le asignamos un nombre, en nuestro caso se llamará ADMCensus.

Nombre de la Tarea

☒ **Crear un proyecto de estilo libre**
Esta es la característica principal de Jenkins, la de ejecutar el proyecto combinando cualquier tipo de repositorio o podrá tanto compilar y empaquetar software, como ejecutar cualquier proceso que requiera monitorización.

☐ **Crear un proyecto maven**
Ejecuta un proyecto maven. Jenkins es capaz de aprovechar la configuración presente en los ficheros POM, reduciendo la configuración.

☐ **Crear un proyecto multi-configuración**
Adecuado para proyectos que requieran un gran número de configuraciones diferentes, como testear en múltiples entornos.

☐ **External Job**
Este tipo de tareas te permite registrar la ejecución de un proceso externo a Jenkins, incluso en una máquina remota. Para más información consulta esta [página](#).

☐ **Copiar una Tarea existente**
Copiar desde

OK

Figura 39: Creación del proyecto en Jenkins

Tras haber creado la tarea, procedemos a configurar el repositorio de GitHub.

Configurar el origen del código fuente

☐ Ninguno
☐ CVS
☐ CVS Projectset
☒ Git

Repositories

Repository URL:

Credentials:

Branches to build

Branch Specifier (blank for 'any'):

Figura 40: Configuración de repositorio Git en Jenkins

Como el fin de utilizar Jenkins es automatizar las tareas, vamos a escribir las líneas de código que nos permitan hacer lo descrito en el enunciado.

```
#Seccion 1-----
git checkout master
git pull

#Seccion 2-----
cd /var/lib/jenkins/jobs/Ejercicio\ integracion\ externa/workspace/census/ADMCensus
mvn clean
mvn validate
mvn compile war:war

#Seccion 3-----
rm /var/lib/tomcat7/webapps/ADMCensus.war
rm -r -f /var/lib/tomcat7/webapps/ADMCensus
cd /var/lib/jenkins/jobs/Ejercicio\ integracion\ externa/workspace/census/ADMCensus/target/ADMCensus-1.3.war /var/lib/tomcat7/webapps/ADMCensus.war

#Seccion 4-----
mvn sonar:sonar

#Seccion 5-----
cd /var/lib/jenkins/jobs/Ejercicio\ integracion\ externa/workspace/census/ADMCensus/src/main/java/services
doxygen /home/egc/DocumentacionDoxyfile/fichero_config/config
service tomcat7 restart
```

Figura 41: Código de tarea en Jenkins

Vamos a analizar las líneas escritas anteriormente:

- La primera sección es la encargada de descargarse el código de GitHub.
- La segunda sección navega hasta la carpeta del proyecto y ejecuta comandos de Maven para verificar y compilar en un WAR el proyecto, en este caso ADCensus.
- La tercera sección copia ese WAR creado por Maven y sustituye el que hay actualmente en Tomcat 7.

- d. Para analizar la calidad del código hacemos uso de SonarQube mediante el comando Maven que vemos en la cuarta sección. SonarQube reconoce que el proyecto es de tipo Maven, por lo que este comando debe ejecutarse en la raíz donde se encuentre el archivo de configuración pom.xml.
- e. Por último, en la quinta sección, navegamos hasta la carpeta que nos piden generar la documentación y ejecutamos el comando para que Doxygen genere la documentación de la clase solicitada. Para que genere la documentación acorde a lo que configuramos al instalar la herramienta, en el comando de ejecución, debemos indicarle donde está el archivo de configuración.
- f. Reiniciamos Tomcat7 para que actualice los cambios y listo.

Una vez creado el script encargado de ejecutar las tareas pedidas, vamos a automatizar la ejecución. Para ello debemos incluir en nuestra tarea el siguiente comando:

```
H H * * *
```

Nota: Si no queremos esperar a que se ejecute de forma automática, podemos darle a construir ahora y vemos los resultados obtenidos.

Como resultado final, si accedemos a la URL del proyecto (localhost:8080/ADMCensus) debemos ver lo siguiente:



Figura 42: Subsistema Creación y Administración de Censos funcionando

Para visualizar los resultados aportados por SonarQube, debemos acceder a la URL de SonarQube (localhost:9000) y acceder al proyecto que se ha creado, mostrándose lo siguiente:

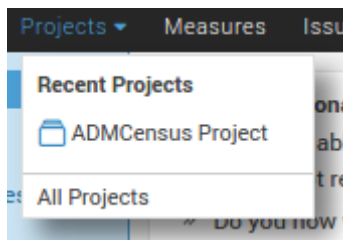


Figura 43: Selección de proyectos en SonarQube

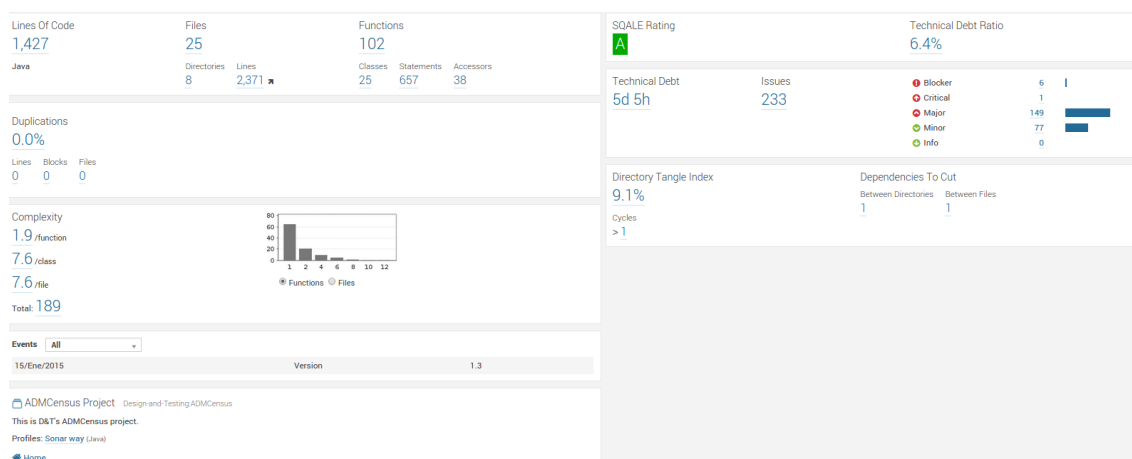


Figura 44: Análisis del código de ADMCensus

Para poder ver la documentación autogenerada por Doxygen accederemos a la siguiente carpeta:

```
/home/egc/DocumentacionDoxyfile/html/index.html
```

Visualizaremos la documentación generada de la siguiente forma.

```
Census services.CensusService.create ( int    idVotacion,
                                         String username,
                                         String fecha_inicio,
                                         String fecha_fin,
                                         String tituloVotacion
                                         )      throws ParseException
```

Método que crea un censo a partir de una votacion

Parameters

idVotacion	Identificador de la votacion
username	Nombre de usuario que ha creado la votacion
fecha_inicio	Fecha de inicio para la votacion
fecha_fin	Fecha de fin para la votacion
tituloVotacion	Cadena de texto con el titulo de la votacion

Returns

census

Exceptions

ParseException

Figura 45: Documentación generada por DoxyGen

7. GESTIÓN DEL CAMBIO, INCIDENCIA Y DEPURACIÓN

7.1. GESTIÓN DE CAMBIOS

7.1.1. PROBLEMA

Cuando hablamos de gestión del cambio, nos referimos a las acciones a llevar a cabo en los momentos en los que surgen cambios que tienen como consecuencia un impacto en el desarrollo de un determinado proyecto.

Por tanto, este es un proceso que debe quedar bien definido ya que el desarrollo del sistema depende mucho de la buena o mala realización del mismo. Se dice esto porque dichos cambios deben quedar bien analizados y registrados, habiendo seguido unas pautas preestablecidas; o incluso deben ser comunicados a la persona o al grupo correspondiente para poder así evitar retrasos en el desarrollo del sistema.

7.1.2. SOLUCIÓN PROPUESTA

Esta fase va de la mano de depuración, ya que la depuración consta de la aplicación de cambios para solventar incidencias.

En el momento que aparece un cambio a efectuar, la primera acción es analizar dicho cambio para llegar a contemplar los efectos tanto negativos como positivos para el desarrollo. Este análisis el cual será el que determine si los cambios son factibles, dependiendo de su impacto, utilidad y beneficios; consiste en identificar y estudiar el cambio a llevar a cabo teniendo en cuenta de que esto puede repercutir a fases anteriores al desarrollo, como puede ser por ejemplo la fase de análisis de requisitos.

Podemos clasificar los cambios en 3 tipos, dependiendo de su impacto:

- Si el impacto de dicho cambio es **crítico**, se realizará con la mayor prioridad buscando la estabilidad en el proyecto. Entendemos cambio crítico aquel que vaya en contra de alguna regla de negocio.
- En cambio, si dicha modificación es de nivel **medio**, la fecha de ejecución dependerá en función de otros cambios con mayor prioridad. Entendemos cambio medio aquel que afecta a una regla de negocio pero no va en contra de ella.
- Por último nos encontramos con los cambios **leves**, es decir, de muy poco impacto. Estos suelen ser los que se llevan a cabo en último lugar. Entendemos como cambios leves aquellos cambios que afectan tanto al diseño como a la navegabilidad del sistema.

Una vez realizado el análisis si dichos cambios son factibles, se lleva a cabo el reporte y la realización de estos. La asignación de cambios se otorga al desarrollador que esté trabajando o se haya encargado del requisito funcional sobre el cual se tiene que efectuar un determinado cambio. Este desarrollador es el responsable de los cambios realizados en el código y de informar por medio de los *commit* al repositorio de las modificaciones realizadas. De esta forma se pueden controlar todos los cambios realizados en el proyecto.

En cuanto a los roles para dicha gestión, tenemos a los propios desarrolladores, los cuales serán los encargados de ejecutar un cambio determinado en el código; al gestor de pruebas que será el encargado de analizar los cambios a efectuar y en último lugar tenemos al jefe de proyecto, cuya función es la de organizar al equipo, analizar y reportar junto con dicho gestor estos cambios.

Finalmente, toda la información importante sobre los cambios a efectuar recogida en un documento a modo de registro, para poder llevar un control sobre estos.

Recogiendo de este modo la persona encargada de realizar un determinado cambio, la acción realizada junto con la fecha y hora del momento en que se llevó a cabo y finalmente un identificador, el cual será un número entero que se verá incrementado por cada nuevo registro.

[identificador - fecha y hora – acción - persona encargada]

7.1.3. LECCIONES APRENDIDAS

Como conclusión, para mantener una buena práctica de la gestión del cambio, debemos mantener un registro de los cambios realizados durante el ciclo de vida del desarrollo del proyecto, para así mantener un control mediante al cual se pueden resolver futuros conflictos.

También es importante definir bien el impacto de los cambios, de esa forma podremos asignar más recursos a los cambios con mayor prioridad. Esto se consigue categorizando correctamente los cambios entrantes.

7.2. GESTIÓN DE INCIDENCIAS

7.2.1. PROBLEMA

Las incidencias aportan información relevante e importante sobre un error encontrado en el desarrollo o ejecución del sistema, por lo tanto es claro que debe haber una forma eficaz y rápida de reportar estas incidencias a los desarrolladores para poder ser resueltas con la mayor celeridad posible.

Con un buen sistema para la gestión de incidencias es difícil que un error se pase por alto, y por lo tanto aumente las probabilidades de éxito del proyecto.

7.2.2. SOLUCIÓN PROPUESTA

Inicialmente la gestión de incidencias se realiza verbalmente reportando los errores al grupo de trabajo en el cual se encuentra un determinado error. El grupo de trabajo afectado resuelve dicho error y reporta una solución a través de una aplicación de mensajería instantánea. A medida que la integración con los distintos grupos avanza, se utiliza como herramienta de gestión, ProjETSII, en la cual se realizan reportes mediante documentos o imágenes para explicar con un mayor detalle donde se encuentra dicho error. Finalmente, una vez solucionado dicho error, se comunica a través de ProjETSII que dicha incidencia ha sido solucionada y por tanto, donde se va a depositar el código con el error ya solucionado.

Sin embargo, estas técnicas para comunicar las incidencias se muestran ineficaces, especialmente la forma verbal, ya que no queda un registro escrito de la nombrada incidencia y complica su resolución, por lo tanto se deben buscar alternativas.

Pero antes de encontrar una mejor solución a las nombradas se debe esclarecer la información que un registro de incidencias debe tener para poder abordarlas de forma correcta.

1. Registro de incidencia:

La admisión y registro de la incidencia es el primer paso necesario y el más importante para poder ejecutar una correcta gestión de la misma.

Las incidencias pueden provenir de diversas fuentes tales como usuarios, gestión de aplicaciones, el Centro de Servicios o el soporte técnico, entre otros.

El proceso de registro debe realizarse inmediatamente, pues resulta mucho más costoso realizarlo a posteriori, además de que se corre el riesgo de que la aparición de nuevas incidencias demore indefinidamente el proceso en cuestión.

Cabe destacar que es muy importante aportar toda la información posible a la descripción de la incidencia ya que es la que va a ayudar a localizar el error reportado y por tanto solventarlo con una mayor brevedad.

Esto consiste en introducir un asunto de la incidencia breve y conciso, y una descripción en la que se detalle todo lo que ha ocurrido para encontrarse con un determinado error.

Se aconseja introducir:

- Los datos personales, tales como nombre, apellidos y correo electrónico.
- El entorno de desarrollo en el que ha surgido dicho error, en el caso de que el sistema se encuentre en desarrollo.
- El sistema bajo el que se está desarrollando o corriendo el sistema.
- El momento en el que esto sucedió.
- Las acciones que hicieron aparecer el error.
- Los mensajes que tanto el sistema como el entorno de desarrollo puede mostrar.
- Y por último, a ser posible, una imagen que refleje lo descrito en este apartado.

2. Clasificación de incidencias:

Al registrar una incidencia, debemos proceder a la clasificación de este. Esta tiene como objetivo principal el recopilar toda la información posible necesaria para que posteriormente pueda ser utilizada para la resolución de dicha incidencia.

El proceso a seguir para realizar la clasificación debe implementar, al menos, los siguientes pasos:

- **Categorización:** se asigna una categoría (que puede estar a su vez subdividida en más niveles) dependiendo del tipo de incidente o del grupo de trabajo responsable de su resolución. Se identifican los servicios afectados por el incidente.
- **Establecimiento del nivel de prioridad:** dependiendo del impacto y la urgencia se determina, según criterios preestablecidos, un nivel de prioridad. Este nivel será el que determine el tiempo en el que se llevará a cabo la revisión de una determinada incidencia.
- **Asignación de responsable:** se debe atribuir a un responsable, el cual será el portavoz del grupo de trabajo el cual se encargará de informar y organizar el tratamiento, y por tanto la resolución de una determinada incidencia.
- **Monitorización del estado y tiempo de respuesta esperado:** se asocia un estado al incidente (por ejemplo: registrado, en progreso, suspendido en prueba, resuelto, cerrado)

Estos son los pasos a seguir para reportar una incidencia adecuadamente, debe registrarse y a continuación categorizarse para facilitar todo lo posible la tarea del equipo de desarrollo.

Con esto aclarado se pueden abordar ahora distintas herramientas para la gestión de incidencias que se adapten a las necesidades del proyecto, en este caso tenemos dos propuestas:

1. BugTracker:

Se propone la herramienta *Zoho BugTracker*, la cual podemos encontrar accediendo al siguiente enlace:

<https://www.zoho.com/bugtracker/>

Esta es una aplicación que ha resultado tanto sencilla como útil e intuitiva, además puede decirse que está diseñada para poder asegurar una correcta gestión de incidencias.

Esto conlleva una mejora en el aseguramiento de la calidad del software y asistencia tanto a programadores como a cualquier otra persona involucrada en el desarrollo de un determinado proyecto. Además de que puede ser de ayuda en cuanto al seguimiento de los defectos de software, incluso en los sistemas informáticos ya desarrollados.

En primer lugar para utilizar dicha herramienta hay que registrarse en el sistema. Una vez creada la cuenta hay que crear un proyecto, en el que reportaremos las incidencias.

Tenemos que agregar a los componentes de dicho proyecto, los cuales también deben de estar registrados en el sistema. Agregar componentes es una tarea muy sencilla ya que la interfaz de dicha herramienta se asemeja mucho a una red social.

Para realizar incidencias en el menú lateral seleccionaremos “*Errores*”. Nos aparecerá una ventana para introducir una nueva incidencia. Cuando pinchamos en dicho enlace aparecerá una imagen como la mostrada a continuación, en la cual podemos observar los distintos campos a rellenar para reportar dicha incidencia:

Prueba EGC

Descripción del problema

B *I* U abc x_2 x^2 F TTTT

Se permite un máximo de 10 archivos por carga

Arrastre los archivos o añada los documentos adjuntos aquí...

Añadir seguidores

Otra información

<p>¿Es reproducible?</p> <div style="border: 1px solid #ccc; padding: 5px;">Siempre ▼</div>	<p>Clasificación</p> <div style="border: 1px solid #ccc; padding: 5px;">Otros errores ▼</div>
<p>Señalar</p> <div style="border: 1px solid #ccc; padding: 5px;">Interno ▼</div>	<p>Asignar a</p> <div style="border: 1px solid #ccc; padding: 5px;">Sin asignar ▼ (?)</div>
<p>Acontecimiento importante relacionado</p> <div style="border: 1px solid #ccc; padding: 5px;">Ninguno ▼</div>	<p>Fecha de vencimiento</p> <div style="border: 1px solid #ccc; padding: 5px;"> </div>
<p>Módulo</p> <div style="border: 1px solid #ccc; padding: 5px;">Prueba EGC ▼</div>	<p>Gravedad</p> <div style="border: 1px solid #ccc; padding: 5px;">Mayor ▼</div>

Guardar y añadir nuevo
Guardar
Cancelar

Figura 46: Nueva incidencia en BugTracker

En primer lugar, tenemos un campo para dar una descripción detallada de la incidencia.

En segundo lugar tenemos una opción que nos permite subir los ficheros en los cuales se ha encontrado la incidencia.

Tras esta, nos encontramos la opción mediante la cual podemos asignar dicha incidencia a alguno de los componentes del proyecto. En breves, comentaremos con mayor detalle esta opción.

Por último, vemos una sección en la que podemos introducir información adicional. Como por ejemplo:

Si es reproducible, qué clasificación contiene el error, que gravedad tiene, que modulo se ve afectado, etc.

Como hemos dicho anteriormente, vamos a centrarnos en el procedimiento que debe realizar la persona a la que se le ha asignado la incidencia.

En primer lugar automáticamente se le envía un correo electrónico indicando la asignación de dicha tarea, además, como podemos ver a continuación, dentro de la herramienta aparecerá una notificación.

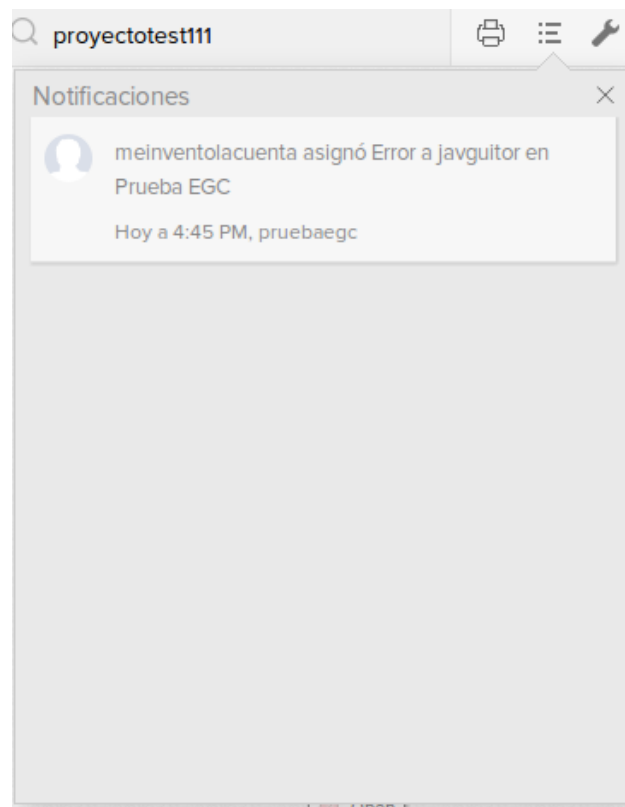


Figura 47: Notificaciones en BugTracker

En la lista de errores aparecerá este nuevo error y nos mostrará en una primera instancia la información de la tarea. Esta información puede ser configurada para que muestre lo que el usuario vea oportuno.



Figura 48: Listado de errores en BugTracker

Esta persona podrá ir modificando el estado de la incidencia para informar al resto de integrantes, el estado en que se encuentra, como podemos ver a continuación:

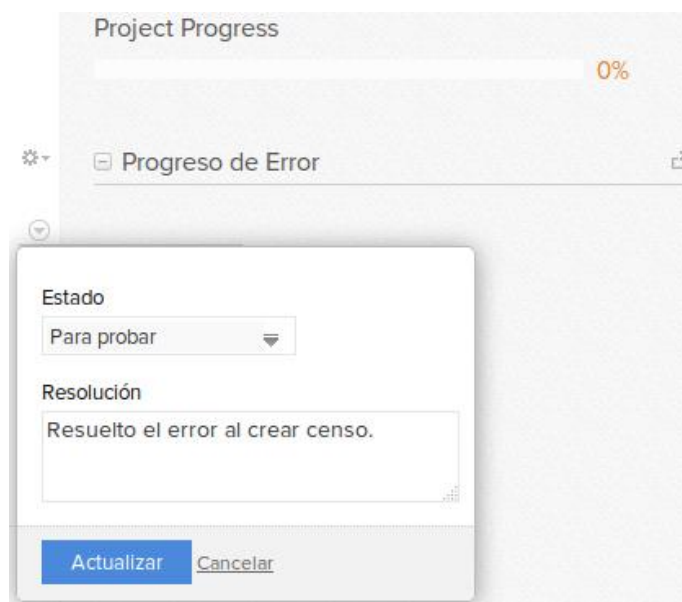


Figura 49: Estado de una incidencia en Bugtracker

Cuando el resto de integrantes vea la notificación de que el error ha cambiado de estado se le realizarán las pruebas oportunas para comprobar que todo funciona correctamente.

En el caso en que el error no haya sido solucionado completamente se volvería a repetir dicho proceso. Si por el contrario el error se ha solucionado, se procede al cambio del estado del error a cerrado.

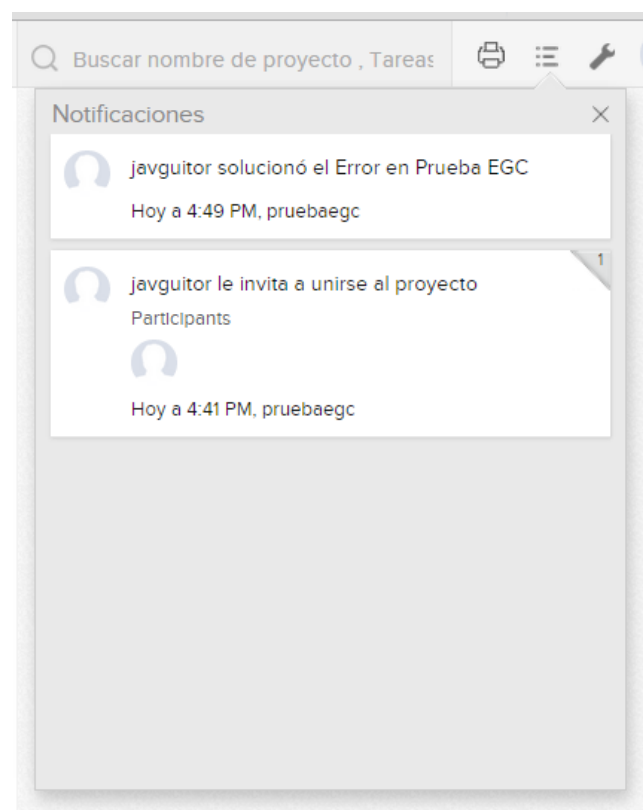


Figura 50: Notificación de error solucionado

Como podemos observar se trata de una herramienta bastante útil y sencilla a la hora de reportar errores, además de aportar una funcionalidad extra que permite realizar un seguimiento del proyecto, y comprobar en tiempo real cuántos errores existen y en qué situación se encuentran, como podemos observar a continuación.

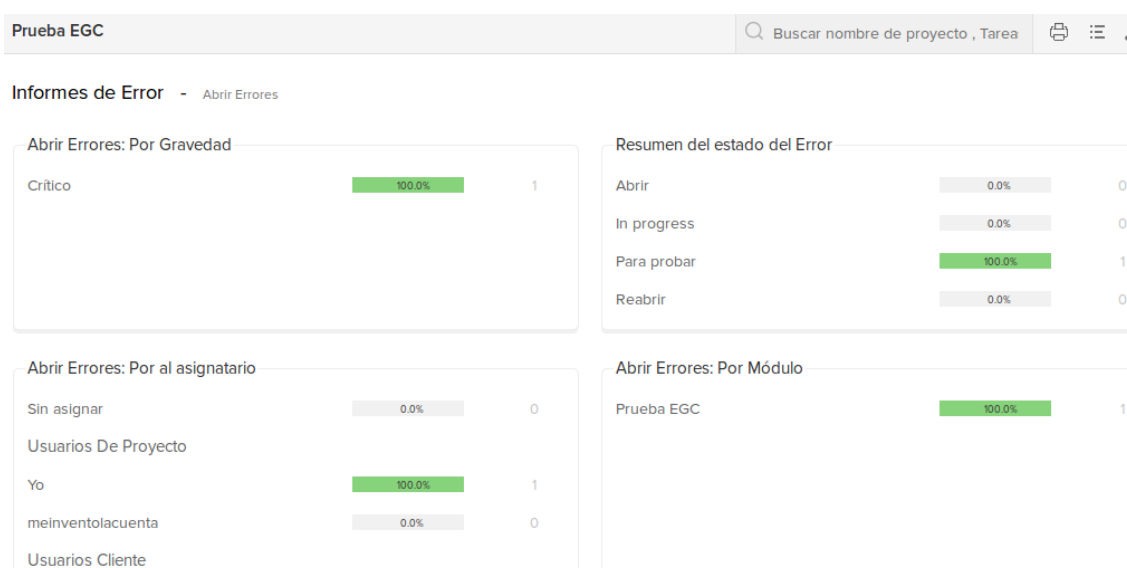


Figura 51: Informe de errores

2. GitHub:

Finalmente, se acordó utilizar como herramienta para reportar incidencias los *Issues* aportados por GitHub, ya que tenemos un repositorio compartido con los distintos subsistemas:

<https://github.com/EGC-1415-Repositorio-compartido/repvoting/>

La decisión de utilizar esta herramienta se debe a la sencillez de reportar la incidencia. Es decir, los distintos subsistemas van a utilizar la herramienta de GitHub para poder integrarse con el resto de subsistemas, y poder así, obtener la última versión del código. Por tanto, si durante la integración hubo algún problema es más cómodo reportar dicha incidencia y poder asignarla a la o las personas involucradas en el sistema.

Para crear una nueva incidencia utilizando la funcionalidad de GitHub, hay que poseer una cuenta de usuario, además de tener un proyecto subido a dicho repositorio. Para ello, dentro de la rama en la cual queramos realizar la incidencia, seleccionamos la opción *Issues* en el menú lateral.

Una vez realizado esto, se nos abrirá un nuevo menú en el cual podremos introducir el nombre de la incidencia, la persona a quien se le va a asignar la incidencia, un campo donde poner de manera detallada la descripción de la misma y por último podremos etiquetarla con una serie de categorías predefinidas.

A la persona a la que se le ha asignado la incidencia recibirá un correo electrónico. En dicho correo se le indicará donde se ha encontrado el error y qué especificaciones tiene.

Cuando la persona asignada resuelva dicha incidencia reporta su solución a la persona que creó la incidencia. Marcará por tanto la incidencia por cerrada. Cuando esto se realiza, se le envía un correo electrónico a la persona que creó la incidencia, para que en el caso en el que no esté utilizando la aplicación en el momento de la resolución, sepa que se ha resuelto.

La principal ventaja que presenta dicha herramienta para el reporte de incidencias, es que se asemeja mucho con una red social.

7.2.3. LECCIONES APRENDIDAS

Realizar una correcta gestión de incidencias desde el comienzo del proyecto es muy importante, ya que una gestión inapropiada (especialmente en la forma de comunicación de estas incidencias) puede desembocar en errores costosos de reparar en fases tardías del proyecto.

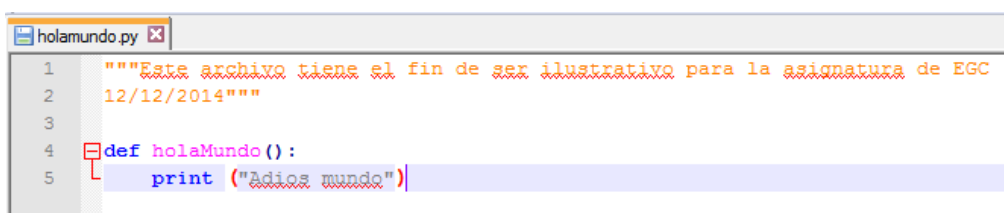
A su vez es importante familiarizar a los miembros del equipo con las buenas prácticas de la gestión de incidencias e inculcarles a todos la importancia y beneficios de esta para que sea más eficiente y productiva, y por tanto menos costoso para el proyecto.

7.2.4. EJERCICIO 1

“Realización de una incidencia por parte del “usuario 1” al “usuario 2” debido a un error en el fichero holamundo.py” haciendo uso de BugTracker

Resolución

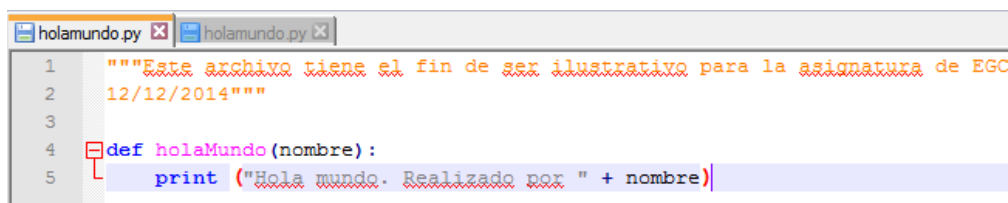
Para realizar el ejercicio el “usuario 1” será javguitor quien envía la incidencia al “usuario 2”, en este caso guio.fbb, adjuntándole el fichero holamundo.py, el cual contiene lo siguiente:



```
holamundo.py
1  """Este archivo tiene el fin de ser ilustrativo para la asignatura de EGC
2  12/12/2014"""
3
4  def holaMundo():
5      print("Adios mundo")
```

Figura 52: Fichero con incidencia por resolver

El usuario guio.fbb, tras resolver la incidencia, el archivo debe quedar de la siguiente forma:



```
holamundo.py
1  """Este archivo tiene el fin de ser ilustrativo para la asignatura de EGC
2  12/12/2014"""
3
4  def holaMundo(nombre):
5      print("Hola mundo. Realizado por " + nombre)
```

Figura 53: Resolución esperada de la incidencia

Javguitor realiza la incidencia en el menú lateral seleccionando “Errores”, como se ha descrito anteriormente, este menú posee una serie de opciones las cuales nos permitirán especificar la incidencia.

La incidencia creada por javguitor le llega a guio.fbb mediante una notificación. Este usuario abre dicha notificación y comprueba el error y sus detalles como se muestra en las siguientes imágenes. Ambas aparecen en la misma vista.

Ejercicio EGC

Informado por javguitor el 12-19-2014 03:11 PM

Realización de una incidencia por parte del "usuario 1" al "usuario 2" debido a un error en el fichero holamundo.py

[Comentarios](#)[Documentos ad...](#)[Resolución](#)[Activida...](#)

Archivos adjuntos :

[holamundo.py](#)


 Arrastre los archivos o añada los documentos adjuntos aquí...

Figura 54: Incidencia por resolver

×

PE2 Seguir

Nombre del proyecto :
Prueba EGC

Asignar a :
guio.fbb

Fecha de vencimiento :
12-20-2014 12:00 AM [borrar](#)

Estado

Abrir ▼

Gravedad

Crítico ▼

Hito asociado

Ninguno ▼

Módulo

Prueba EGC ▼

Clasificación

Mejora ▼

¿Es reproducible?

Not applicable ▼

Señalar :
Interno

Figura 55: Detalles de la incidencia

En la imagen situada en segundo lugar se ven las especificaciones de la incidencia creada por javguitor.

El usuario guio.fbb descarga el fichero adjuntado en la incidencia para resolverla. Realiza los cambios pertinentes y actualiza el estado de la incidencia adjuntando el fichero modificado.

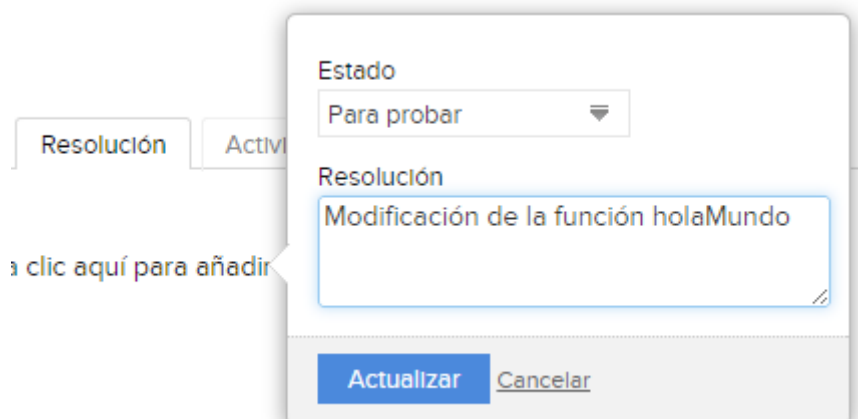


Figura 56: Cambio de estado en una incidencia

En este caso guio.fbb cambia el estado a “Para probar”. Una vez haya sido probado el nuevo fichero modificado, procederá a volver a cambiar el estado a “Cerrado”. Ya estaría resuelta la incidencia.

Los usuarios podrán comprobar el historial de cambios de la incidencia en el apartado “Actividades”. En la siguiente imagen se muestra el proceso completo realizado en el ejercicio desde que se creó la incidencia hasta que ha sido resuelta.

Ejercicio EGC

Informado por javguitor el 12-19-2014 03:11 PM

Realización de una incidencia por parte del "usuario 1" al "usuario 2" debido a un error en el fichero holamundo.py

Editar

Comentarios Documentos ad... Resolución Activida...

12-12-2014

ACTUALIZ..

Estado from Para probar to Cerrado
guio.fbb, 03:49 PM

AGREGADO

Documento adjunto from holamundo.py
guio.fbb, 03:47 PM

AGREGADO

Resolución to Modificación de la función holaMundo
guio.fbb, 03:45 PM

AGREGADO

Estado from Abrir to Para probar
guio.fbb, 03:45 PM

CREADO

Error archivado
javguitor, 03:11 PM

AGREGADO

Documento adjunto from holamundo.py
javguitor, 03:11 PM

Figura 57: Historial de cambios en una incidencia

7.2.5. EJERCICIO 2

“Realización de una incidencia por parte del “usuario 1” al “usuario 2” haciendo uso de GitHub.”

Resolución

La resolución de este ejercicio es un caso real. A continuación se explica cómo el usuario davalvsil crea una incidencia debido a un problema en la integración con uno de los subsistemas de Agora@US, en concreto con el subsistema Creación de votaciones.

La creación de dicha incidencia se muestra en la siguiente imagen:

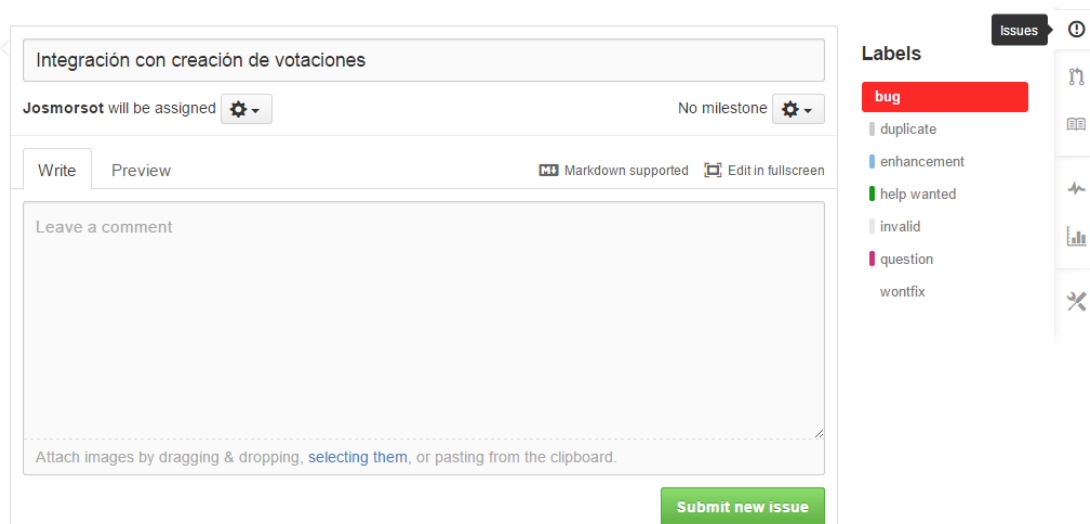


Figura 58: Creación de Issue en GitHub

Como podemos ver el título de la incidencia es “Integración con creación de votaciones”, ha sido asignada al usuario Josmorsot para que la resuelva y etiquetada como “bug”. Una vez creada debe quedar similar a esta imagen donde se puede ver que el estado de la incidencia es Abierto (Open). Además con un simple vistazo podemos observar los detalles de ésta.

Integración con creación de votaciones #2

Open davalvsil opened this issue just now · 0 comments

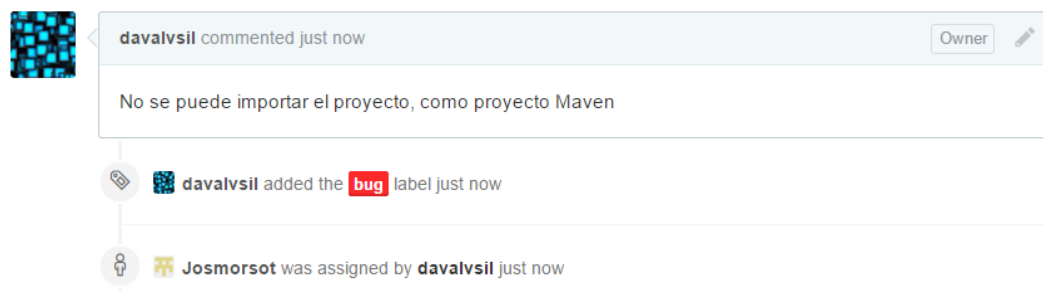


Figura 59: Detalles de incidencia

Una vez la persona asignada haya resuelto la incidencia será notificado al usuario creador (davalvsil) mediante 2 formas:

- Notificación en la propia herramienta → Al abrirla se verá la resolución y el cambio de estado de la incidencia. En este caso al haber sido solventada completamente el estado ha pasado a Closed.

Integración con creación de votaciones #2

Closed davalvsil opened this issue 40 minutes ago · 1 comment

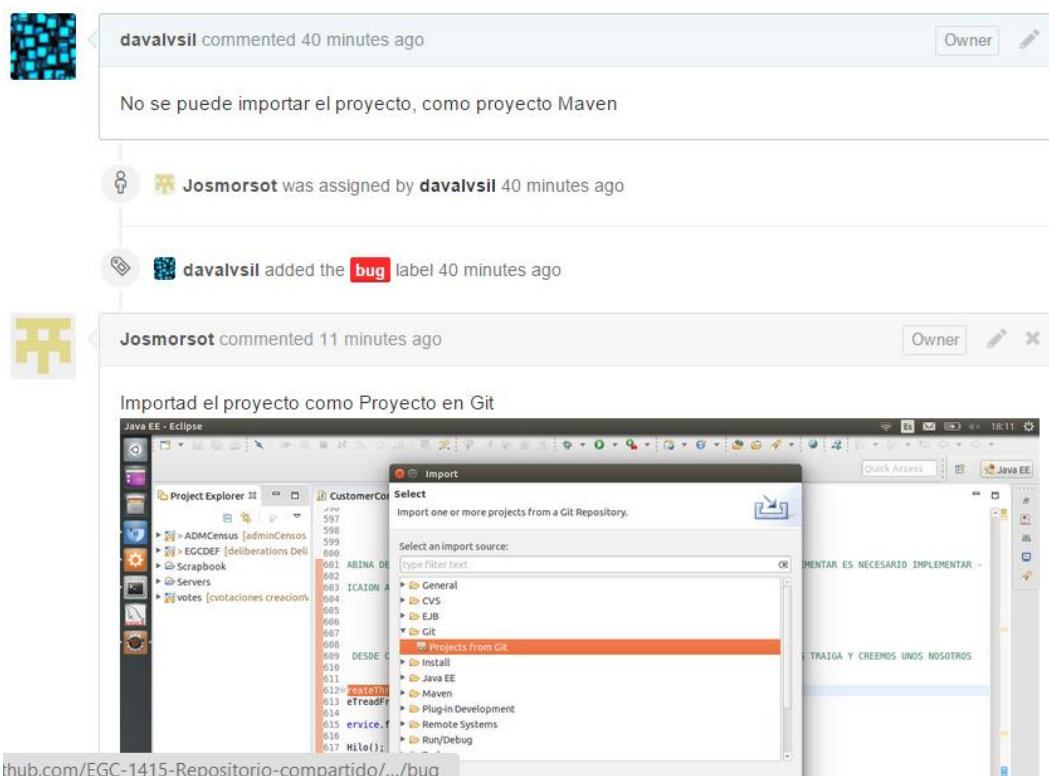


Figura 60: Notificación en GitHub de resolución

Notificación por correo electrónico → El usuario davalvsil ha recibido el siguiente correo electrónico con la resolución de la incidencia reportada.

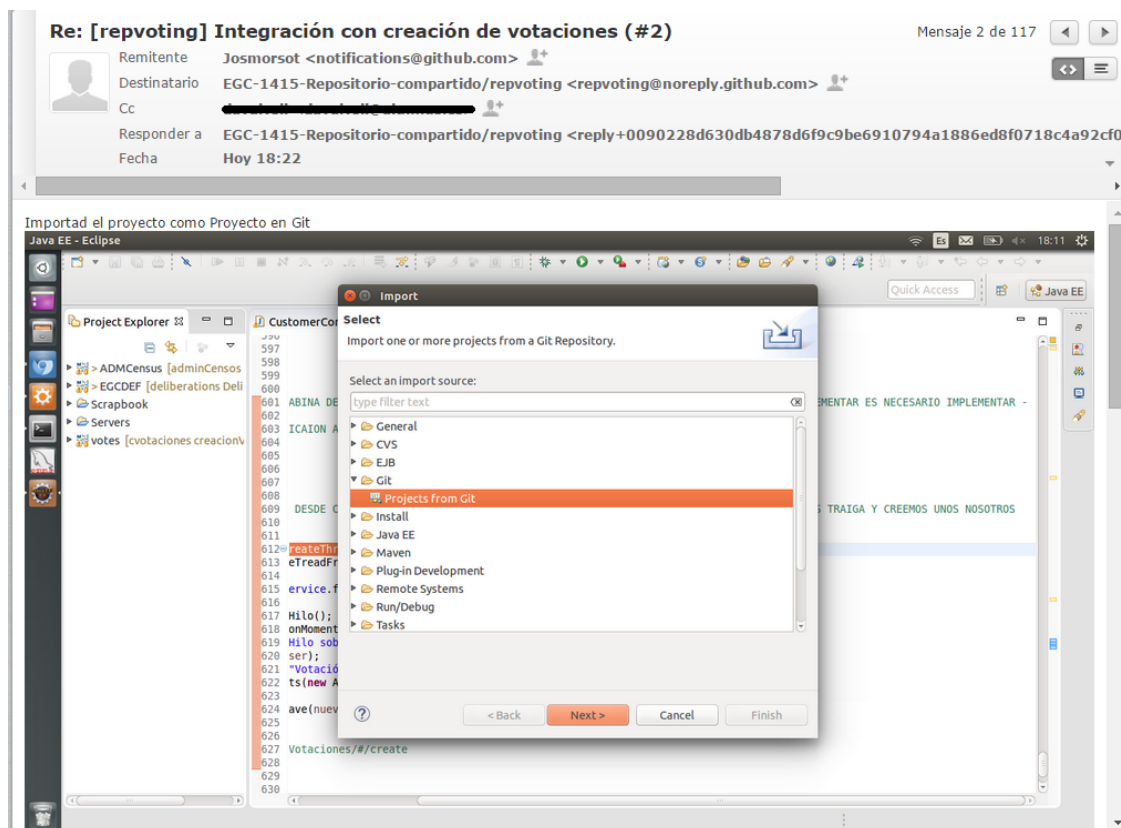


Figura 61: Notificación por email de resolución

Ya está la incidencia solventada.

Otra opción de GitHub es ver todas las incidencias registradas en el proyecto aunque no hayan sido asignadas al usuario registrado. En el caso del proyecto Agora@US estas son algunas de ellas:

<input type="checkbox"/>	8 Open	9 Closed	Author	Labels	Milestones	Assignee	Sort
<input type="checkbox"/>			Sección de despliegue en la memoria.	agreement			0
			#18 opened 13 hours ago by danayaher				
<input type="checkbox"/>			Error con la librería de verificación				2
			#17 opened 15 hours ago by abayta				
<input type="checkbox"/>			Definición del tipo VOTO (definitiva, por favor) ATENCIÓN CABINA				8
			#16 opened 15 hours ago by juanmartin1892				
<input type="checkbox"/>			Fecha para tener todo el código acabado	question			2
			#15 opened 15 hours ago by davalvsl				
<input type="checkbox"/>			Error al importar librería de Verificación en subsistema Recuento	bug help wanted			4
			#14 opened 17 hours ago by iyoque				
<input type="checkbox"/>			Dirección base de datos de Autenticación				1
			#11 opened a day ago by Josmorsot				
<input type="checkbox"/>			Verificación: longitud de la clave RSA	bug			7
			#10 opened 2 days ago by juamalosu				
<input type="checkbox"/>			Error al integrar Deliberaciones	bug			1
			#6 opened 5 days ago by alesanmed				

Figura 62: Registro de incidencias abiertas

7.3. DEPURACIÓN

7.3.1. PROBLEMA

Cuando hablamos de depuración hablamos de la fase que nos encontramos una vez registrada y clasificada una determinada incidencia, estudiada y analizada, para posteriormente realizar la búsqueda de esta.

Por tanto, hacemos referencia al proceso de encontrar e intentar reducir y solventar los errores o defectos que el software pueda contener.

7.3.2. SOLUCIÓN PROPUESTA

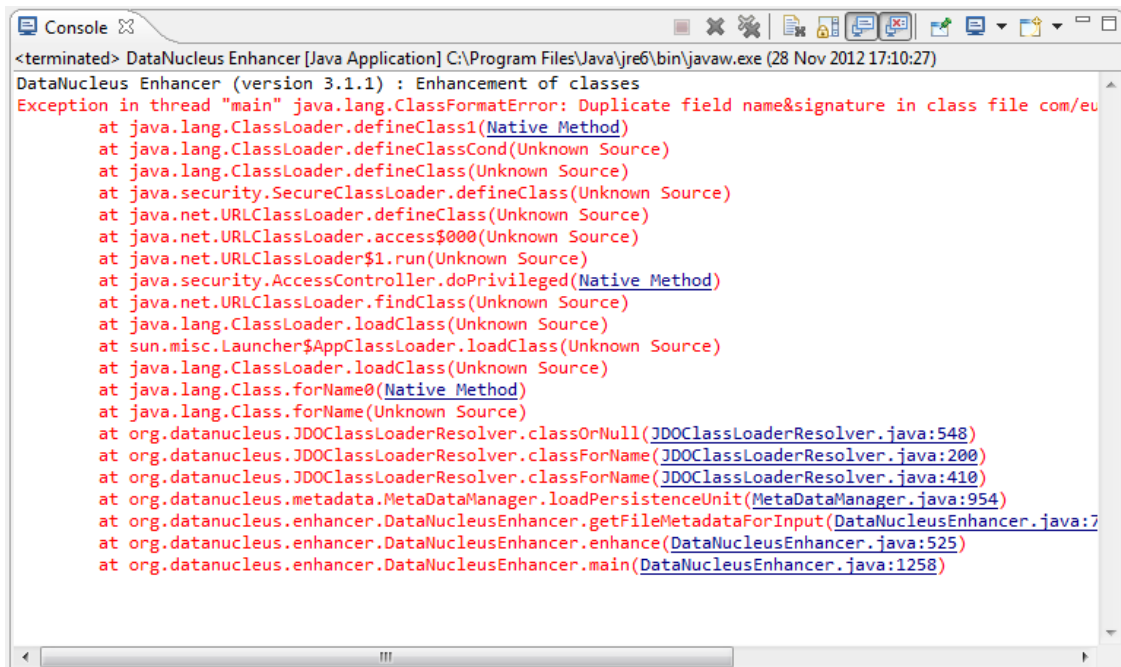
Se comienza depurando aquellas incidencias de mayor nivel de prioridad, es decir, las que reporten un error de mayor proporción que afecte en el funcionamiento del sistema.

Como primer paso para realizar la depuración, debemos comprobar de qué se trata la incidencia, para conocer con más detalle lo reportado. Esto nos ayudará a proporcionar varias hipótesis en las que basarnos para localizar nuestro objetivo. El tratamiento a llevar a cabo, como en muchas ocasiones, dependen del tipo de error y por tanto de la naturaleza de los mismos.

Por consiguiente, una vez analizadas las posibles hipótesis, el siguiente paso es el de aislar el problema, reduciendo el espacio de búsqueda del error descartando dichas hipótesis.

Por tanto, dados los diferentes métodos existentes para proceder a la depuración, podemos comenzar con intentar reproducir la incidencia en distintos escenarios, para comprobar que el problema no es del entorno de desarrollo y por tanto poder descartar esa opción.

En caso de que eso no resulte ser satisfactorio, procedemos a comprobar la traza mostrada por el sistema al ejecutar dicha excepción. La siguiente imagen muestra un ejemplo de ello:



```
<terminated> DataNucleus Enhancer [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (28 Nov 2012 17:10:27)
DataNucleus Enhancer (version 3.1.1) : Enhancement of classes
Exception in thread "main" java.lang.ClassFormatError: Duplicate field name&signature in class file com/eu
    at java.lang.ClassLoader.defineClass1(Native Method)
    at java.lang.ClassLoader.defineClassCond(Unknown Source)
    at java.lang.ClassLoader.defineClass(Unknown Source)
    at java.security.SecureClassLoader.defineClass(Unknown Source)
    at java.net.URLClassLoader.defineClass(Unknown Source)
    at java.net.URLClassLoader.access$000(Unknown Source)
    at java.net.URLClassLoader$1.run(Unknown Source)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(Unknown Source)
    at java.lang.ClassLoader.loadClass(Unknown Source)
    at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)
    at java.lang.ClassLoader.loadClass(Unknown Source)
    at java.lang.Class.forName0(Native Method)
    at java.lang.Class.forName(Unknown Source)
    at org.datanucleus.JDOClassLoaderResolver.classOrNull(JDOClassLoaderResolver.java:548)
    at org.datanucleus.JDOClassLoaderResolver.classForName(JDOClassLoaderResolver.java:200)
    at org.datanucleus.JDOClassLoaderResolver.classForName(JDOClassLoaderResolver.java:410)
    at org.datanucleus.metadata.MetadataManager.loadPersistenceUnit(MetadataManager.java:954)
    at org.datanucleus.enhancer.DataNucleusEnhancer.getFileMetadataForInput(DataNucleusEnhancer.java:7
    at org.datanucleus.enhancer.DataNucleusEnhancer.enhance(DataNucleusEnhancer.java:525)
    at org.datanucleus.enhancer.DataNucleusEnhancer.main(DataNucleusEnhancer.java:1258)
```

Figura 63: Ejemplo de excepción

Para ello, existen herramientas integradas en la mayoría de los entornos de desarrollo, las cuales reciben el nombre de depuradores. Los depuradores también ofrecen funciones tales como correr un programa paso a paso, es decir, pausar la ejecución del programa para examinar el estado actual en cierta instrucción especificada por medio de un punto de ruptura (*Breakpoint*) y poder visualizar los valores de las variables implicadas.

En la siguiente imagen muestra esto:

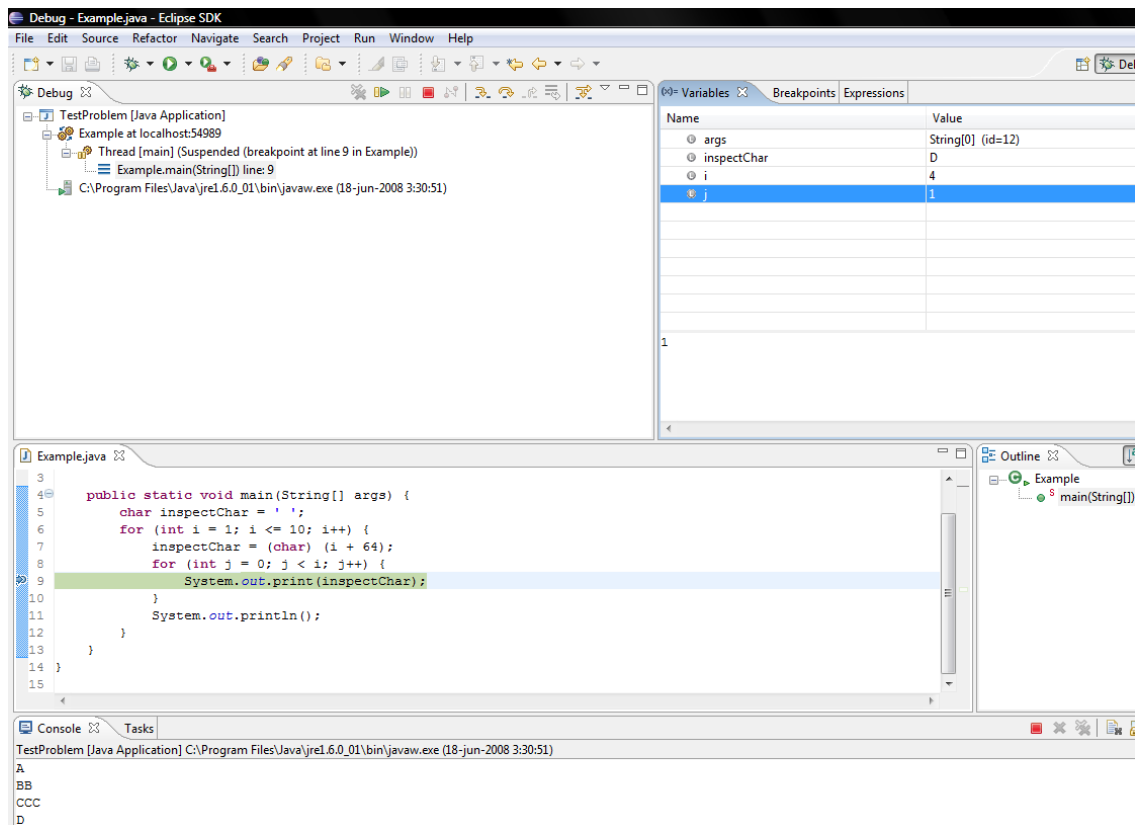


Figura 64: Modo depuración de Eclipse

De esta forma, si con una traza no es posible localizar el error, con este tipo de herramientas se puede indagar dentro del código y poder así detectar donde se encuentra el error observando los valores que van tomando las variables a lo largo de la ejecución.

Otro método de depuración es la comparación con ejercicios similares realizados de manera correcta. Es por ello que si se tiene una memoria de buenas prácticas o simplemente un proyecto similar, es posible encontrar la solución a dicho error comparando dichos proyectos.

No siempre el propio desarrollador que ha elaborado el código es capaz de encontrar el error, debido a que está predispuesto a que su código sea óptimo, por lo que hay casos en los que resulta de gran ayuda que sean localizados y posteriormente se proceda a solucionar los errores reportados, por desarrolladores ajenos al código en cuestión.

Una vez localizado el error, se procede al análisis de este y por consiguiente a la resolución del mismo.

Llegados a este punto, la fase de depuración queda casi concluida, ya que antes de darlo por finalizado se debería realizar una comprobación de si existen posibles errores que sean causados como consecuencia de la resolución de este, además de errores o fallos que sean similares a este que se puedan dar en un futuro.

Por último hay que mencionar el uso de la herramienta SonarQube, la cual, como se ha comentado anteriormente, se ha utilizado durante el ciclo de desarrollo del proyecto y que entre otras utilidades, en cuanto a depuración, tras los análisis realizados por esta, si hubiese algún error en el código analizado o el software tuviese algún defecto dicha herramienta es capaz de localizar la instrucción o las instrucciones en las que se produce esto.

7.3.3. LECCIONES APRENDIDAS

La depuración de las incidencias se debería realizar en un plazo de tiempo corto a la recepción de la misma pero siempre atendiendo a la prioridad establecida por esta, para evitar el aumento del coste para su resolución, además de que no siempre se debe ceder la responsabilidad de la ejecución de la misma al propietario del código a analizar ya que a veces esto puede resultar más costoso en tiempo.

Cabe destacar que esta fase resulta más complicada cuanto mayor es el tamaño del proyecto, por lo que se deberían seguir unas buenas prácticas de desarrollo para así evitar este tipo de contratiempos.

8. GESTIÓN DE DESPLIEGUE

8.1. PROPUESTA DE DESPLIEGUE

La siguiente propuesta de despliegue se ha realizado en común con los miembros de los grupos de Deliberaciones, Administración de votaciones y nuestro subsistema.

Una vez que todos los subsistemas hayan finalizado su código y se hayan realizado las pruebas pertinentes de integración con todos ellos, en distintos entornos de desarrollo, al no encontrar ningún fallo, se pasa a realizar el despliegue. 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 máquina 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)

Se parte de la base de que todo el código está en el repositorio compartido de GitHub y se podrán 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 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). Esta directiva hay que realizarla dentro de cada una de las ramas, donde se 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 la carpeta `var/lib/webapps/tomcat7`

Para los proyectos PHP, como Auth, se copia a la carpeta `/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 “passwrod”

Otros proyectos, como Visualización de Resultados (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.

Para el proyecto Python realizado por Cabina de Votación se debe crear una estructura de carpetas como la siguiente:

- Carpeta raíz: cabina-integracion
- 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 máquina 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 sería instalar esta librería en el repositorio Maven 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.

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.

9. MAPAS DE HERRAMIENTAS

9.1. MAPA DE HERRAMIENTAS PROPIO

Este es el mapa de herramientas correspondiente al desarrollo e integración de nuestro propio subsistema. Todas las herramientas aquí citadas se han utilizado en el desarrollo del subsistema “Creación y administración de censos”.

9.1.1. SISTEMA OPERATIVO

UBUNTU

En un principio el desarrollo iba a realizarse en Windows, pero tras hablar con los miembros de otro subsistema que realizarían el desarrollo en Ubuntu utilizando unos frameworks de este sistema operativo, se acordó desarrollar desde un primer momento en Ubuntu para evitar futuros problemas.

9.1.2. ENTORNOS

ECLIPSE

Utilizado para el desarrollo en lenguaje java con soporte para los servidores y frameworks escogidos para la realización del subsistema. Como en el anterior caso, el equipo se beneficia de una gran familiarización con el entorno y por lo tanto un mejor aprovechamiento y utilización de este.

9.1.3. LENGUAJES

JAVA

Al principio se planteó la posibilidad de hacer el desarrollo en Python, pero cuando el grupo fue consciente de los problemas que podíamos encontrarnos a la hora de realizar la integración, a la vez que de los problemas de aprendizaje de dicho lenguaje, ya que ninguno ha realizado desarrollos con cierto peso en este lenguaje, se optó por descartarlo.

La opción que se eligió fue Java, por la familiaridad que tenemos con dicho lenguaje y el conocimiento que tenemos sobre las posibilidades que tiene para poder realizar una integración. Este lenguaje es la base de la codificación y uso de los distintos frameworks, permitiéndonos el aprovechamiento de varios de estos.

9.1.4. SERVIDORES

APACHE

Servidor base para la utilización del motor de servlets de Tomcat, necesario puesto que ambos se presentan en combinación.

TOMCAT

Servidor utilizado para el despliegue del subsistema. Este despliegue se realiza en local y no en la web.

Ambos servidores se instalan bajo el sistema operativo Ubuntu y se integran con el entorno de desarrollo Eclipse.

9.1.5. BASE DE DATOS

MySQL

Este gestor de base de datos se instala sobre el sistema operativo Ubuntu y permite la creación y administración de una base de datos relacional para nuestro subsistema gracias a los frameworks utilizados.

Se decidió utilizar MySQL por la familiaridad que tenemos con dicho gestor de base de datos, a la vez que pensamos que habría menos problemas para integrar el resto de subsistemas ya que estos también utilizaban MySQL.

9.1.6. REPOSITARIOS

SVN

Esta herramienta de control de versiones utilizada en conjunto con Eclipse nos permite una sencilla gestión del código fuente de nuestro subsistema.

El servidor utilizado para alojar el repositorio es el que nos proporciona la propia escuela, ProjETSII.

9.1.7. FRAMEWORKS

SPRING

Este framework hace de soporte para el resto de herramientas utilizadas, instalado sobre el entorno de desarrollo Eclipse.

HIBERNATE

Herramienta de Mapeo objeto-relacional (ORM) utilizada para la implementación de la siguiente herramienta a describir.

JPA 2.1

Esta API nos proporciona ventajas en su utilización, como el uso del lenguaje propio JPQL, para la interacción con nuestra base de datos MySQL. Para su implementación se utiliza el framework anteriormente citado.

JSP

Basado en Java nos permite la creación de páginas web dinámicas y un mejor aprovechamiento de las herramientas anteriores. Instalado en conjunto con Spring sobre el entorno Eclipse al igual que el resto de frameworks.

9.1.8. HERRAMIENTAS DE ENTORNOS

APACHE-MAVEN

Herramienta del entorno de trabajo Eclipse que nos permite una gestión sencilla del proyecto Java ofreciéndonos un entorno unificado mediante un conjunto de plugins. En definitiva mejora el entorno de trabajo facilitando la instalación y uso de los frameworks elegidos para la realización de este proyecto.

9.1.9. HERRAMIENTAS DE INTEGRACIÓN

JENKINS

Servidor de integración continua utilizado para la integración de los diversos subsistemas del proyecto, entre algunas de las funcionalidades que se han utilizado cabe destacar el uso del plugin de Git, ejecución de scripts y despliegue automatizado. Dicha herramienta ha sido instalada sobre el sistema operativo Ubuntu.

SONARQUBE

Herramienta para el control de calidad del código, utilizada mayoritariamente para el apartado de depuración de nuestro sistema.

DOXYGEN

Herramienta para la generación automática de documentación del sistema.

9.1.10. REPRESENTACIÓN GRÁFICA DE LA RELACIÓN ENTRE LAS HERRAMIENTAS

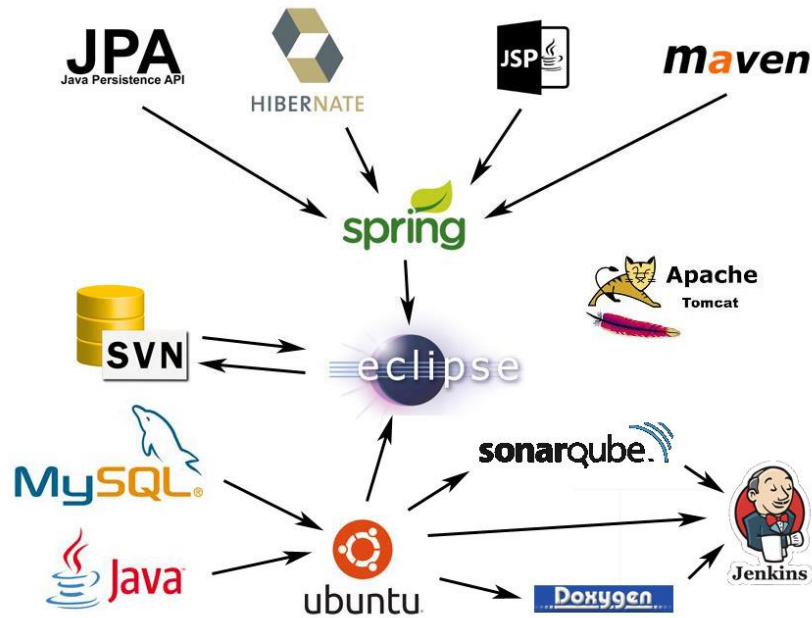


Figura 65: Mapa de herramientas propio

9.2. MAPA DE HERRAMIENTAS GENERAL

Este es el mapa de herramientas para el desarrollo e integración de todos los subsistemas del proyecto. Se incluyen todas las herramientas utilizadas por grupos externos a “Creación y administración de censos”.

9.2.1. SISTEMAS OPERATIVOS

LINUX

Toda la integración se ha realizado bajo sistemas operativos basados en Linux, ya que soporta todas las herramientas utilizadas en sus diferentes versiones además de beneficiarnos con el uso de software libre.

9.2.2. ENTORNOS

ECLIPSE

Utilizado para el desarrollo en lenguaje java de los distintos subsistemas con soporte para los servidores y frameworks escogidos por estos.

PYTHON VIRTUALENV

Herramienta para crear entornos de trabajo independientes. Sobre este funcionan todos los frameworks relacionados con el lenguaje Python.

9.2.3. LENGUAJES

JAVA

Este lenguaje es la base de la codificación y uso de los distintos frameworks, permitiéndonos el aprovechamiento de varios de estos.

HTML/PHP

Utilizados por varios subsistemas para la codificación. Se hacen uso de APIs y json para la comunicación con el resto de lenguajes.

PYTHON

Utilizado por un subsistema junto a un framework específico para este lenguaje.

9.2.4. SERVIDORES

APACHE

Servidor base para la utilización del motor de servlets de Tomcat, necesario puesto que ambos se presentan en combinación.

TOMCAT

Servidor utilizado para el despliegue del subsistema. La razón de que haya dos es que el que está relacionado con XAMPP se utiliza para el despliegue local, el instalado sobre el SO se utiliza para despliegue en la web.

XAMPP

Servidor utilizado para el despliegue de los subsistemas que utilizan PHP. Despliegue en local.

GUNICORN

Servidor para el despliegue local de los subsistemas basados en Python. Compatible con todos los frameworks utilizados y ligero en recursos.

9.2.5. BASE DE DATOS

MySQL

Este gestor de base de datos se instala sobre el sistema operativo y se relaciona directamente con el servidor XAMPP y permite la creación y administración de una base de datos relacional para nuestro subsistema gracias a los frameworks utilizados.

9.2.6. REPOSITORIOS

SVN

Esta herramienta de control de versiones utilizada como plugin en conjunto con Eclipse nos permite una sencilla gestión del código fuente de nuestro subsistema.

El servidor utilizado para alojar el repositorio es el que nos proporciona la propia escuela, ProjETSII.

GIT/GITHUB

Herramienta de control de versiones utilizada por distintos subsistemas para la gestión del código fuente. Además de instalarse sobre el sistema operativo se utiliza un plugin instalado sobre el entorno de desarrollo eclipse.

9.2.7. FRAMEWORKS

SPRING

Este framework hace de soporte para el resto de herramientas utilizadas, instalado sobre el entorno de desarrollo Eclipse.

HIBERNATE

Herramienta de Mapeo objeto-relacional (ORM) utilizada para la implementación de la siguiente herramienta a describir.

JPA 2.1

Esta API nos proporciona ventajas en su utilización, como el uso del lenguaje propio JPQL, para la interacción con nuestra base de datos MySQL. Para su implementación se utiliza el framework anteriormente citado.

JSP

Basado en Java nos permite la creación de páginas web dinámicas y un mejor aprovechamiento de las herramientas anteriores. Instalado en conjunto con Spring sobre el entorno Eclipse al igual que el resto de frameworks.

DJANGO

Framework utilizado en los subsistemas desarrollados en Python que permite un desarrollo rápido y limpio, no hay una comunicación directa con el resto de Frameworks.

REQUESTS

Librería HTTP licenciada de Apache2, simplifica y mejora la librería HTTP propia de Python.

DJANGO REST FRAMEWORK

Conjunto de herramientas para la creación de web API's de forma sencilla y rápida.

RSA

Framework para la encriptación y desencriptación, login y verificación de firmas digitales; y generación de claves.

9.2.8. HERRAMIENTAS DE ENTORNO

APACHE-MAVEN

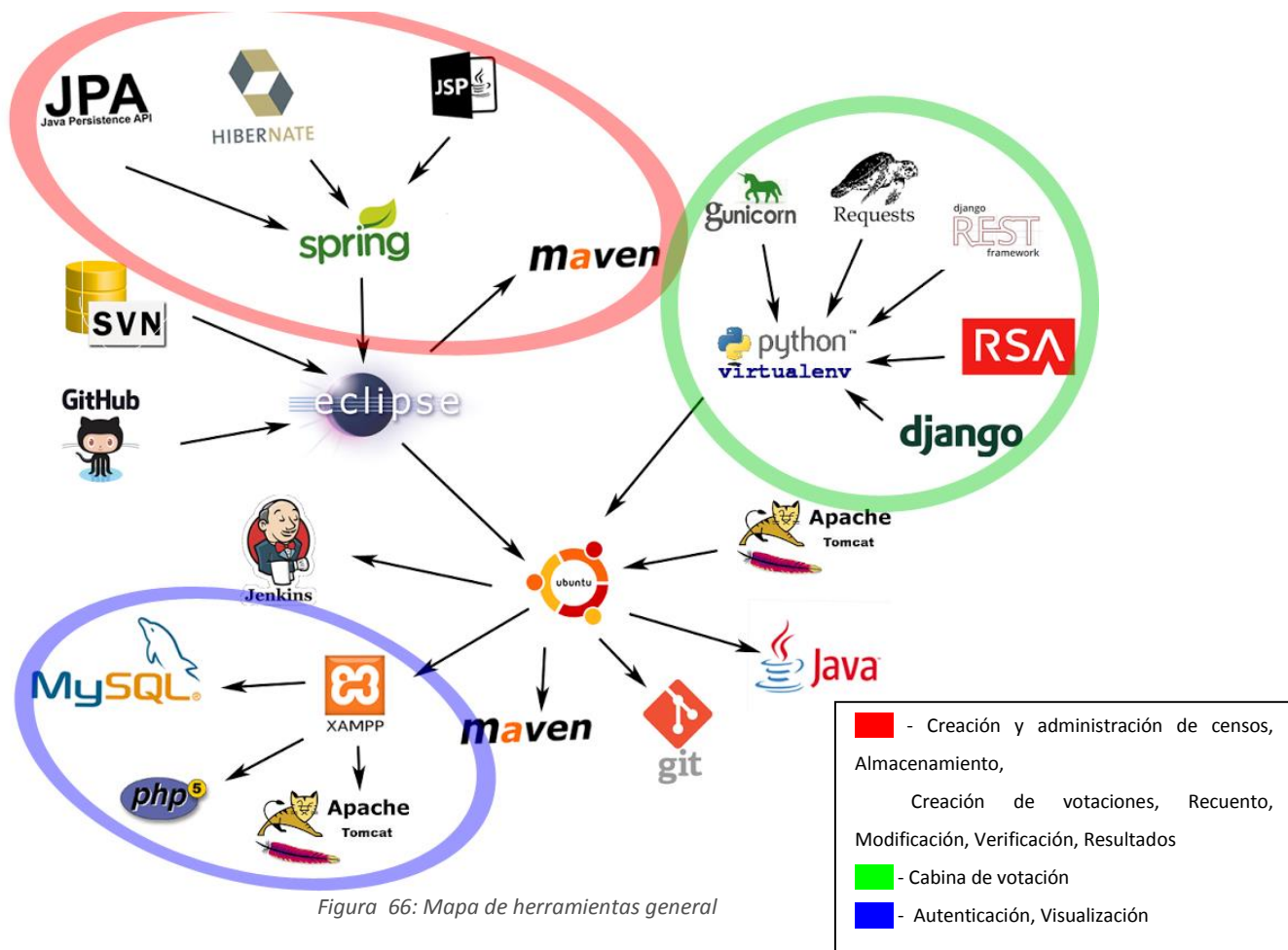
Herramienta del entorno de trabajo Eclipse que nos permite una gestión sencilla del proyecto Java ofreciéndonos un entorno unificado mediante un conjunto de plugins. En definitiva mejora el entorno de trabajo facilitando enormemente la instalación y uso de los frameworks (salvo Django) elegidos para la realización de este proyecto.

9.2.9. INTEGRACIÓN CONTINUA

JENKINS

Servidor de integración continua utilizado para la integración de los diversos subsistemas del proyecto, entre algunas de las funcionalidades que se han utilizado cabe destacar el uso del plugin de Git, ejecución de scripts y despliegue automatizado. Dicha herramienta ha sido instalada sobre el sistema operativo Ubuntu.

9.2.10. REPRESENTACIÓN GRÁFICA DE LA RELACIÓN ENTRE LAS HERRAMIENTAS



10. CONCLUSIONES

A través del proyecto propuesto en la asignatura hemos podido llevar a la práctica tanto los conceptos vistos en la teoría de Evolución y Gestión de la Configuración, como de asignaturas anteriores. Enfrentándonos a problemas reales y dándoles solución por nuestros propios medios, en la medida de lo posible.

Al tener que hacer un proyecto, en el que distintos subsistemas debían integrarse y estando cada uno de ellos realizados de formas distintas, hemos tenido que aprender a usar herramientas, que para algunos de nosotros eran desconocidas, como Git para poder manejar un repositorio compartido o Jenkins para poder realizar la integración.

Por último decir que aunque la comunicación interna de nuestro grupo se ha realizado de una forma fluida, han surgido muchos problemas a la hora de comunicarnos y poner en común con otros grupos algunas soluciones o problemas. No solo debido a la falta de comunicación, sino también a la falta de participación o interés de algunos miembros de otros grupos, debido a estos problemas ha habido cambios hasta el último momento repercutiendo en otros grupos que sí ponían de su parte.

BIBLIOGRAFÍA

Integración

<http://tratandodeentenderlo.blogspot.com.es/2009/09/integracion-continua.html>

<http://emanchado.github.io/camino-mejor-programador/html/ch08.html>

http://es.wikipedia.org/wiki/Integraci%C3%B3n_continua

Gestión del código fuente

<https://ariejan.net/2007/07/03/how-to-create-and-apply-a-patch-with-subversion/>

Gestión del cambio, incidencias y depuración

<https://www.zoho.com/bugtracker/>

<http://es.wikipedia.org/wiki/Depurador>

http://itilv3.osiatis.es/operacion_servicios_TI/gestion_incidencias/registro_clasificacion.php

<http://www.fing.edu.uy/inco/cursos/ingsoft/pis/proceso/MUM/treebrowses/disciplinas/gestionconf/indexGestionConf.htm>

En todos los apartados

Diapositivas de la asignatura

Trabajos consultados de otros años

- Grupo 1 - Chromium OS
- Grupo 5 - Django Framework
- Grupo 4 - Gestión de la configuración de AOSP

GLOSARIO DE TÉRMINOS

TÉRMINO	DEFINICIÓN
API	Conjunto de rutinas, protocolos y herramientas para la construcción de aplicaciones de software.
AUTOMATIZACIÓN	Ejecución de procesos con la mínima (o ninguna) intervención.
BASE DE DATOS	Conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.
BASELINE	Línea base para la construcción de una aplicación software.
BRANCH	División del repositorio para la realización de cambios en el código sin afectar a la rama principal o Trunk.
CENSO	Recuento de la población que tiene derecho a voto en unas determinadas elecciones.
COMMIT	Acción en un sistema de control de versiones para enviar cambios realizados en el código (ya sea en local o a servidor).
DESPLIEGUE	Consiste en poner el software en un entorno accesible, de manera que el cliente pueda acceder.
FRAMEWORK	Entorno software que proporciona una funcionalidad particular como parte de una plataforma para el desarrollo de aplicaciones, productos y soluciones software.
INCIDENCIA	Error detectado y documentado en el desarrollo de una aplicación software.
ISSUE	Unidad de trabajo para realizar una mejora en un sistema informático.

LIBRERÍA	Conjunto de soluciones software utilizadas para el desarrollo simplificado de aplicaciones.
PATCH	Modificación del código fuente para la corrección de errores, introducción de nuevas funcionalidades, actualizaciones, etc.
PLUGIN	Aplicación que, en un programa informático, añade una funcionalidad adicional o una nueva característica al software.
PUSH	Comando utilizado para subir cambios a un repositorio remoto.
MÁQUINA VIRTUAL	Software para ejecutar un SO externo con distintas herramientas en una máquina con un SO anfitrión.
REPOSITORIO	Depósito o archivo centralizado donde se almacena y mantiene información digital, habitualmente bases de datos o archivos informáticos.
SCRIPT	Programa simple usualmente almacenado en un archivo de texto plano para la realización de distintas tareas pequeñas dentro del contexto de una aplicación software.
SERVIDOR REMOTO	Combinación de hardware y software que permite el acceso remoto a herramientas o información que generalmente residen en una red de dispositivos.
TAG	Puntos de guardado en momentos importantes del desarrollo de un proyecto en un sistema de control de versiones.
TRUNK	Rama principal del sistema de control de versiones donde se encuentra el código principal de una aplicación.

WAR

JAR utilizado para distribuir una colección de JavaServer Pages, servlets, clases Java, archivos XML, librerías de tags y páginas web estáticas (HTML y archivos relacionados) que juntos constituyen una aplicación web.

ANEXOS

INSTRUCCIONES MÁQUINA VIRTUAL

Descarga de la máquina virtual

La máquina virtual aportada es Ubuntu 14.04.

El enlace en el que se está subiendo dicha máquina es el siguiente:

<https://dl.dropboxusercontent.com/u/53215631/entrega%20egc.rar>

Usuario y contraseña del sistema

Usuario: egc

Contraseña: egc

Al iniciar la máquina:

```
-1: iniciar xampp => sudo /opt/lampp/lampp start  
-2: iniciar tomcat=> sudo service tomcat7 start  
-3: iniciar sonar=> sudo service sonar start
```

Importante

Debido a que el tomcat que se utiliza para desplegar y el tomcat que se utiliza para el eclipse en desarrollo tiene el mismo puerto

Antes de arrancar el eclipse para desarrollar, deberemos iniciar un terminal y poner la siguiente sentencia:

```
sudo service tomcat7 stop
```

Con esto ya se podría iniciar el de desarrollo.

Para poder iniciar el desarrollo debemos iniciar xampp con el comando que se indica a continuación en un terminal.

Abrimos eclipse y arrancamos el servidor.

Si queremos probar el funcionamiento de nuestro subsistema, en el navegador Chrome, hay un marcador "agora@us" que sería la página de login.

usuario: danayaher

password: danayaher1

A continuación apuntamos rutas de utilidad para evaluación de la máquina.

```
ruta del eclipse: /home/egc/eclipse  
ruta de cabina: /home/egc/cabina-integracion  
ruta de proyectos php: /opt/lampp/htdocs/
```

Control de XAMPP

```
sudo /opt/lampp/lampp start,stop,restart
```

Control de Tomcat

```
sudo service tomcat7 start, stop, restart
```

Control de Jenkins

```
sudo service jenkins start, stop, restart
```

Control de Sonar

```
sudo service sonar start, stop, restart
```

Doxygen

La documentación de Doxygen se guarda en la siguiente ruta:

```
/home/egc/DocumentacionDoxyfile
```

Ejecución de Cabina

```
cd /home/egc/cabina-integracion  
  
sudo chmod +x run.sh  
  
sudo ./run.sh
```

Ejecución de Modificación

```
Java -jar  
/var/lib/jenkins/jobs/EGC/workspace/results/Modificacion/build/libs/modificacion-rest-service-1.1.0.jar --server.port=8181
```

Más información

Para más información consultar el archivo .txt situado en el escritorio de la máquina virtual.

En la barra de marcadores, hay accesos a las herramientas anteriormente nombradas