

Documento del proyecto

Grupo 10: Verificación

EVOLUCIÓN Y GESTIÓN DE LA
CONFIGURACIÓN

PARTICIPAN EN LA MEJORA

GARCÍA NIETO, DIEGO – **GESTOR DE LA CONFIGURACIÓN**



LEÓN RIEGO, JOSE MIGUEL – **DESARROLLADOR**
GESTOR DEL CÓDIGO
GESTOR DE CONFIGURACIÓN



MARTÍN MAROTO, SERGIO – **JEFE DE PROYECTO**
GESTOR DE LA CONFIGURACIÓN



PACHÓN JIMÉNEZ, ANDRÉS – **GESTOR DE LA CONFIGURACIÓN**
GESTOR DE INCIDENCIAS



SIERRA SILVA, SAMUEL – **GESTOR DE LA CONFIGURACIÓN**
GESTOR DE LA CALIDAD



UTRERA JAÉN, DANIEL – **GESTOR DE LA CONFIGURACIÓN**
REPRESENTANTE EN GITHUB GLOBAL



Identificación de los Roles

JEFE DE PROYECTO: persona encargada de velar por el cumplimiento del proyecto así como encargarse de firmar, para verificación, toda la documentación que se elabora. Administra el grupo de comunicación y ejerce de canal entre los distintos integrantes.

DESARROLLADOR: persona encargada de la producción de la implementación del código. También se encarga de la evaluación del impacto que supone producir un cambio en una versión funcional.

GESTOR DEL CÓDIGO: persona encargada de gestionar el código que se encuentra en un repositorio. Esta gestión incluye la creación de ramas, unificación de las mismas y controla el historial de subida de todo el ciclo de vida del proyecto.

GESTOR DE LA CALIDAD: persona encargada de velar por la calidad del software usando distintos perfiles de calidad para evaluar el código.

REPRESENTANTE EN GITHUB GLOBAL: persona que se encarga de representar a todo el equipo en el repositorio común de todos los subsistemas, dentro del servicio gratuito que proporciona GITHUB.

GESTOR DE LA CONFIGURACIÓN: persona que se encarga de gestionar los conjuntos de procesos destinados a asegurar la calidad de todo producto obtenido durante cualquiera de las etapas del desarrollo de un Sistema de Información.

GESTOR DE INCIDENCIAS: persona encargada de elaborar las políticas que se utilizan a la hora de gestionar las incidencias, errores o depuración, así como de velar por su cumplimiento y por el uso de buenas prácticas.

Contenido

1. Resumen:	5
2. Introducción	6
3. Subsistema de Verificación y sus funcionalidades	7
4. Gestión de Código Fuente:	8
4.1. Descripción	8
4.2. Caso Práctico	9
5. Gestión de Construcción e Integración Continua:	13
5.1. Integración Continua entre subsistemas	13
5.2. Integración continua en nuestro subsistema.	15
5.3. Caso práctico	16
5.4. Construcción.	19
5.5. Caso práctico	19
5.6. Integración de todos los subsistemas	26
6. Gestión de la calidad	27
6.1. Instalación y Configuración.	27
6.2. Configuración de perfiles de calidad e interpretación de métricas.	30
6.3. Caso práctico 1.	34
6.4. Caso práctico 2.	36
7. Gestión del Cambio, Incidencias y Depuración	39
7.1. Descripción	39
7.2. Caso práctico	41
8. Gestión de Liberaciones, Despliegue y Entregas.	48
8.1. Descripción	48
8.2. Caso práctico	48
9. Mapa de herramientas	52
9.1. Mapa de nuestro subsistema	52
9.2. Mapa general	53
10. Conclusiones:	54
11. Bibliografía	55
12. Anexos	56
12.1. Resumen de las Mejoras Realizadas	56

1. Resumen:

Para la asignatura Evolución y Gestión de la Configuración, se nos propuso la tarea de crear, entre todos, una pequeña versión del proyecto Ágora Voting, llamada Ágora US.

Para ello, los más de 80 alumnos de la asignatura nos dividimos en 11 grupos, y dividimos a su vez la aplicación que teníamos que desarrollar en 11 subsistemas, cada uno a desarrollar por un grupo de alumnos.

En este documento recopilamos todo el trabajo realizado durante el desarrollo de la asignatura y todo lo aprendido durante la misma.

En primer lugar, introduciremos el problema y la aplicación a desarrollar, centrándonos en el subsistema que nosotros debíamos programar e integrar con el resto.

Tras la introducción, detallaremos las diferentes políticas de gestión que hemos seguido para los diferentes aspectos de un proyecto informático, tales como:

- Gestión de código fuente: cómo se gestiona, qué políticas se siguen, cómo se resuelven los conflictos, etc.
- Gestión de la construcción y la integración: qué herramientas usamos para la construcción y cómo se intenta automatizar la integración entre los distintos subsistemas.
- Gestión del cambio, incidencias y depuración: qué política se sigue a la hora de documentar e informar a los demás miembros del equipo acerca de errores, necesidad de algún cambio o de depuración.
- Gestión de la calidad: qué herramientas utilizamos para mejorar y asegurar la calidad del código, qué métricas usamos para el estudio de la calidad, a qué aspectos del código les definimos umbrales, etc.
- Gestión de liberaciones, despliegue y entregas: qué archivos alojamos en el servidor remoto y el proceso que se sigue a la hora de desplegar dichos archivos.

Después de eso, daremos una visión global del mapa de herramientas que hemos utilizado a lo largo de todo el trabajo de esta asignatura.

Y finalmente, explicaremos algunas conclusiones a las que hemos llegado, o algunos detalles que hemos aprendido durante el desarrollo de este proyecto.

2. Introducción

Como hemos explicado anteriormente, para la realización de la aplicación Ágora US, nos basamos en la aplicación ya existente Ágora Voting. Para ello, estudiamos su arquitectura con el fin de entender qué es lo que la aplicación hace internamente y cómo dividirla en diferentes subsistemas:

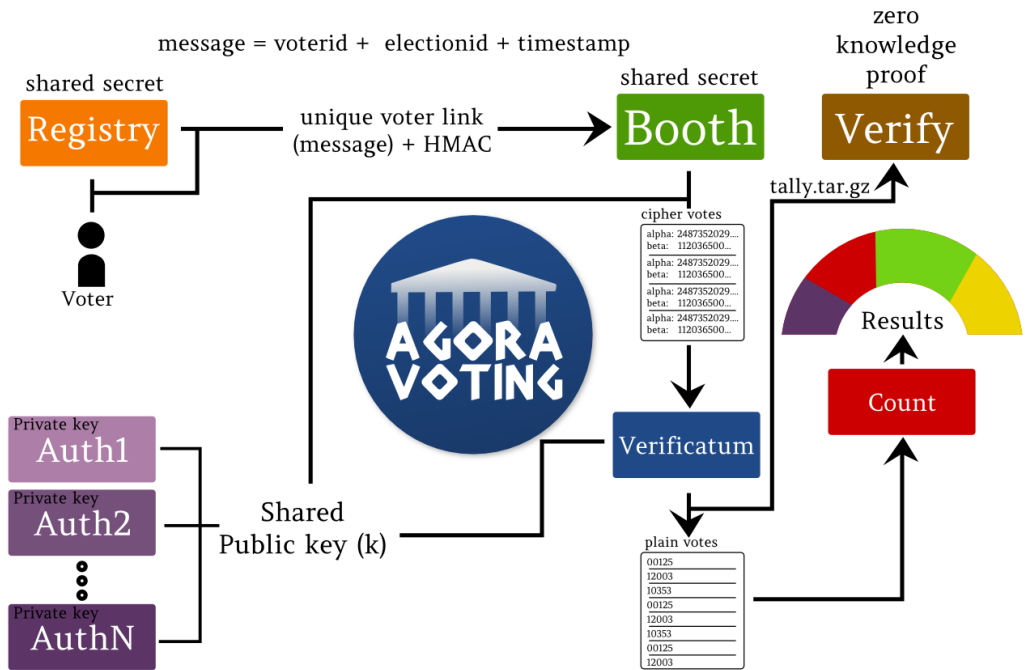


Ilustración 1. Arquitectura de Ágora Voting.

(https://1984.lsi.us.es/wiki-egc/index.php/Lista_de_proyectos_a_realizar_14-15)

Tras estudiar su arquitectura, se dividió la aplicación en 11 subsistemas:

Subsistemas
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

Cada uno de estos subsistemas serán desarrollados por diferentes grupos y deberán ser integrados durante el desarrollo de la asignatura.

3. Subsistema de Verificación y sus funcionalidades

A continuación vamos a explicar brevemente las funcionalidades del subsistema que hemos elegido. Nuestro grupo en concreto, decidió desarrollar el subsistema de Verificación, que se encarga de lo siguiente:

- Es la base de datos en la que se guardan las claves pública y privada para cada votación para que se puedan cifrar o descifrar los votos correctamente.
- Cuando se cree una votación, crearemos un par de claves pública/privada para dicha votación.
- Cada vez que se vaya a cifrar un voto, nosotros proporcionaremos la clave correcta para ello.
- Cuando se vaya a descifrar un voto, verificaremos que ese voto no ha sido alterado, y en el caso de que sea correcto, proporcionaremos la clave para su descifrado.

El subsistema deberá integrarse con aquellos subsistemas que necesiten cifrar o descifrar algún voto o aquel que necesite comprobar que un voto no haya sido modificado. Estos subsistemas son los siguientes:

- Creación de votaciones
- Cabina de votación
- Recuento
- Sistema de modificación de resultados

4. Gestión de Código Fuente:

4.1. Descripción

En primer lugar, cabe destacar que en este proyecto, debido al reducido tamaño en cuanto a líneas de código, se decidió al comienzo asignar a una persona para que ocupara el rol de desarrollador, cuyo cometido será el de elaborar la aplicación en sí. Debido a esto, por motivos evidentes, será el encargado también de gestionar el repositorio de código.

Dicho esto, al comienzo del proyecto, por desconocimiento de otras alternativas, se comenzó a utilizar un repositorio Subversion mediante la herramienta ProjETSII. Sin embargo, conforme la asignatura avanzó, vimos la necesidad de migrar el código a un repositorio Git con GitHub, debido tanto a la propia funcionalidad que ofrece en cuanto a gestión de código, como al hecho de que también es una herramienta muy útil para la comunicación.

La gestión de ramas que se seguirá es la siguiente: cuando sea necesario avanzar en el código de la aplicación, o simplemente realizar un cambio que ha sido reportado por otro grupo o por nosotros mismos (se explicará más adelante la gestión de cambios/incidencias), será el desarrollador el que decidirá, basándose en el impacto de dicho cambio, si es necesario crear una rama para ello, o simplemente modificar el tronco del proyecto directamente.

Por lo general, se creará una nueva rama cuando se trate de algo relacionado con utilizar una tecnología o librería que no hemos usado anteriormente ya que desconocemos, a priori, el impacto que supondrá. Generalmente, a no ser que se requiera colaboración de otro miembro del equipo, la rama se creará en el repositorio local y finalmente, si llega a una versión estable, se hará un merge con la rama master.

En cuanto al nombre de las ramas, se seguirá la siguiente política: se utilizará el nombre de la incidencia por la que se va a realizar el cambio, como por ejemplo “Hotfix issue #11”.

Cuando en una rama alcanzamos el objetivo por el cual esta fue creada, se procederá a realizar un merge con la rama principal para actualizar el proyecto. Si por lo que fuera no se consigue alcanzar el objetivo o nos damos cuenta de que no convienen estos cambios se desechará la rama. Si al realizar el merge surgen conflictos se mantendrá la versión de la cual realizamos el merge.

Cuando se realicen cambios sustanciales en el proyecto que merecen ser guardados en el repositorio principal se procederá a realizar un commit, este debe ir acompañado de una descripción detallada de los cambios que se han hecho, los archivos que han sido modificados y el porqué de las modificaciones.

Además, no sólo utilizamos GitHub para gestionar nuestro código, sino que entre todos los grupos se decidió crear un espacio común en la herramienta en la que gestionar el código de todos los subsistemas, de manera que siempre tuviéramos disponibles las últimas versiones de todos los subsistemas para facilitar la integración.

Con el fin de mantener actualizada nuestra rama del repositorio común, cada vez que se llegue a una versión estable de nuestra aplicación, se procederá a subirla al repositorio común.

4.2. Caso Práctico

En el caso práctico se va a exponer cómo se ha realizado una iteración en el código en la cual el cambio no era trivial. En esta iteración, se adaptará el proyecto para usar también el cifrado AES, pero sin sustituir el método de cifrado anterior, con lo cual, si la iteración finaliza exitosamente, se podría elegir un método de cifrado u otro.

En primer lugar, uno de los miembros del equipo crea un “Issue” en el repositorio de nuestro subsistema (<https://github.com/diegarnie/egc>) en el que se propone el cambio, tal como se aprecia en la imagen:

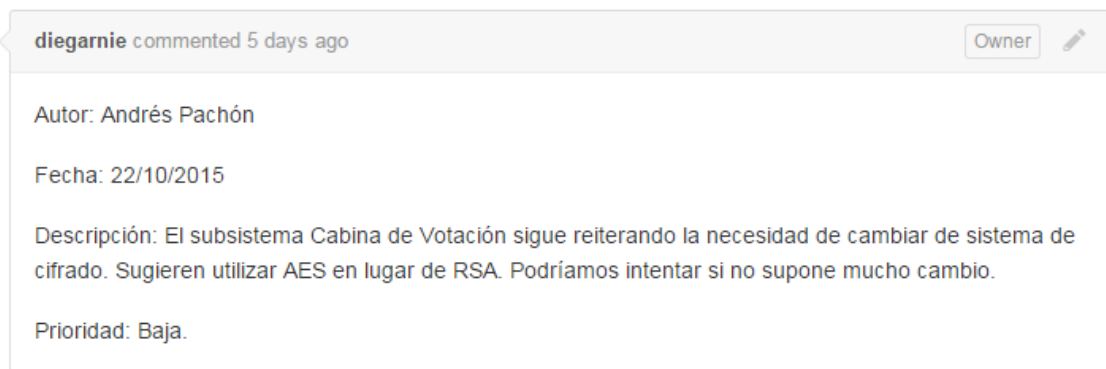


Ilustración 2. Creación de Issue

Una vez creado el “Issue”, el gestor de código pasa a evaluar el impacto, y la dificultad de realizar dicha mejora. Se decide crear una nueva rama, ya que el cambio es relativamente grande y existe cierta incertidumbre, ya que el gestor de código no es experto en criptografía y se desconocen los problemas que puedan surgir durante el desarrollo. Una vez decidido lo anterior, el gestor de código responde el “Issue” indicando la forma de proceder:

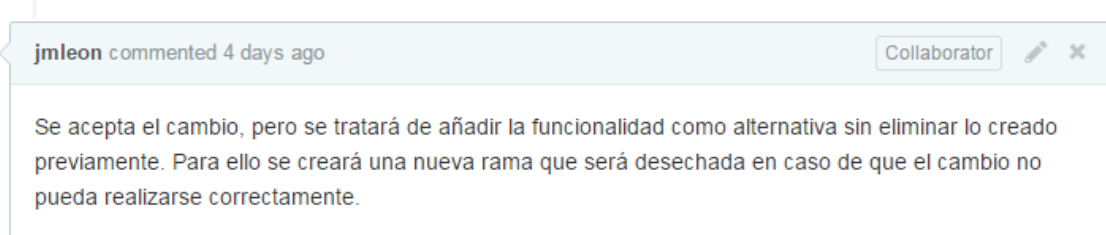


Ilustración 3. Respuesta al Issue

Una vez reflejado lo anterior en GitHub, el gestor de código pasa a realizar la mejora. Para ello, tal como se ha explicado, se crea una nueva rama cuyo nombre es “Issue#3” y se comienza el trabajo sobre dicha rama. Para la creación de la rama usando Eclipse, pulsamos sobre la rama master que aparece en la vista “Git Repositories” y seguidamente pulsamos sobre “Create branch”. Aparecerá una ventana como la de la siguiente imagen en la que únicamente hay que indicar el nombre de la rama y pulsar sobre “Finish”.

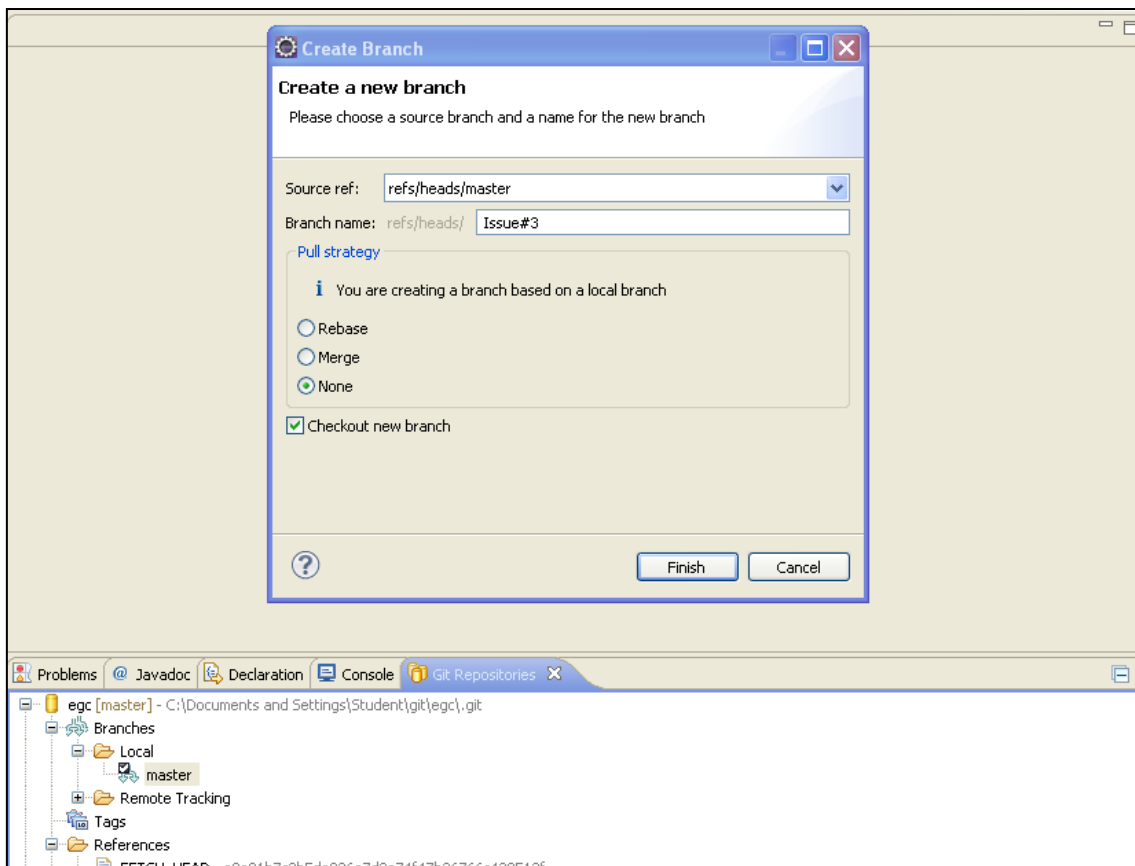
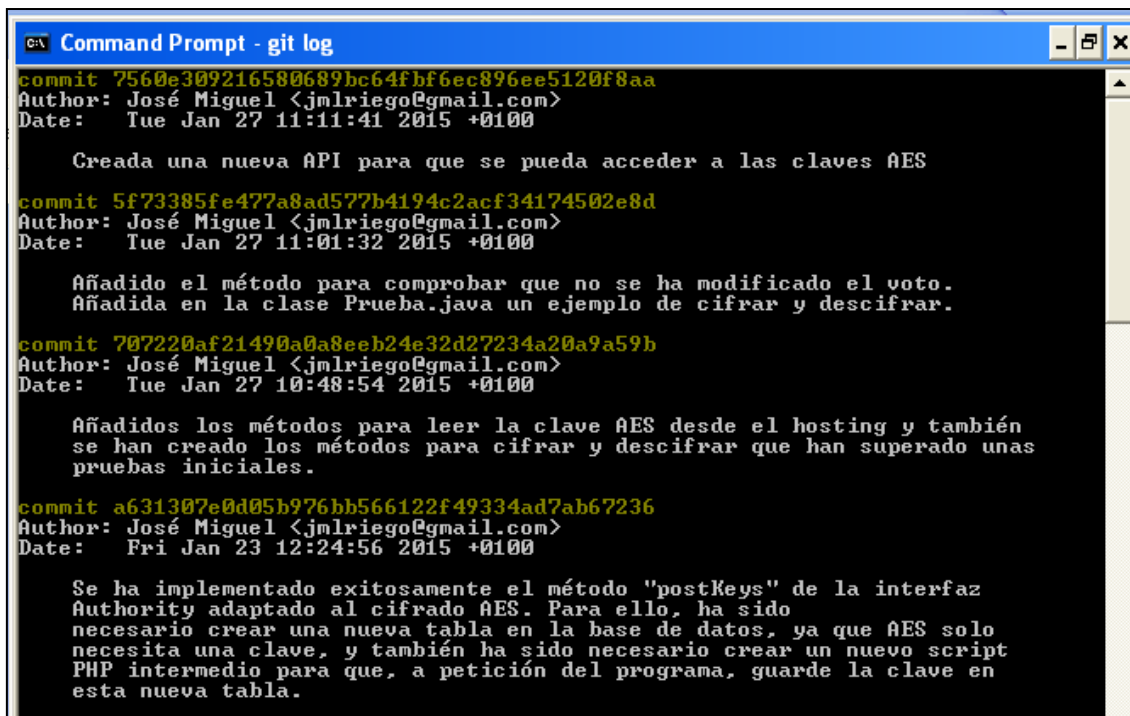


Ilustración 4. Creación de rama en Eclipse

Una vez creada la rama, se comienza con el trabajo. Para ello se ha dividido el trabajo en varias partes relativamente independientes y cada vez que se finaliza el trabajo de cada una de ellas se procedía a realizar un “commit” en el repositorio local. Para completar la mejora han sido necesarios cuatro commits sobre la rama “Issue#3”, que se muestran en la siguiente imagen:



```
Command Prompt - git log
commit 7560e309216580689bc64fbf6ec896ee5120f8aa
Author: José Miguel <jmlriego@gmail.com>
Date: Tue Jan 27 11:11:41 2015 +0100

    Creada una nueva API para que se pueda acceder a las claves AES

commit 5f73385fe477a8ad577b4194c2acf34174502e8d
Author: José Miguel <jmlriego@gmail.com>
Date: Tue Jan 27 11:01:32 2015 +0100

    Añadido el método para comprobar que no se ha modificado el voto.
    Añadida en la clase Prueba.java un ejemplo de cifrar y descifrar.

commit 707220af21490a0a8eeb24e32d27234a20a9a59b
Author: José Miguel <jmlriego@gmail.com>
Date: Tue Jan 27 10:48:54 2015 +0100

    Añadidos los métodos para leer la clave AES desde el hosting y también
    se han creado los métodos para cifrar y descifrar que han superado unas
    pruebas iniciales.

commit a631307e0d05b976bb566122f49334ad7ab67236
Author: José Miguel <jmlriego@gmail.com>
Date: Fri Jan 23 12:24:56 2015 +0100

    Se ha implementado exitosamente el método "postKeys" de la interfaz
    Authority adaptado al cifrado AES. Para ello, ha sido
    necesario crear una nueva tabla en la base de datos, ya que AES solo
    necesita una clave, y también ha sido necesario crear un nuevo script
    PHP intermedio para que, a petición del programa, guarde la clave en
    esta nueva tabla.
```

Ilustración 5. Commits realizados para la mejora.

En el primer commit, se crea el método que permite crear una clave de cifrado de 256 bits y guardarla en la base de datos subida a Hostinger. Para ello también ha sido necesario crear una nueva tabla en la base de datos que incluya las claves AES y crear un script para poder comunicar nuestra aplicación con la base de datos de Hostinger.

En el segundo commit, se crean los métodos para obtener desde la base de datos de Hostinger las claves que se han subido. También se crean dos métodos para cifrar y descifrar que nos permiten probar el correcto funcionamiento del algoritmo de cifrado.

En el tercer commit, se crea un método para comprobar que un voto no ha sido modificado. Esta comprobación se basa en que en el voto cifrado se ha añadido antes de cifrar el código hash del voto mediante el algoritmo md5. De este modo, el algoritmo de verificación descifra la cadena que contiene el voto y su hash, los separa y vuelve a calcular el hash, que debe coincidir con el que ya teníamos previamente. Los métodos que se han creado para cifrar y descifrar están adaptados para añadir el hash al final cuando se cifra y eliminarlo cuando se descifra, aunque mediante un parámetro se puede especificar que se cifre y descifre sin tener en cuenta el código hash, aunque en este caso no sería posible la verificación.

Por último, en el cuarto commit se crea un archivo PHP para que se pueda acceder a las claves a través de una petición HTTP con el siguiente formato:

[http://www.egcprueba.esy.es/getAESKeys.php?id=\[dataBaseID\]](http://www.egcprueba.esy.es/getAESKeys.php?id=[dataBaseID])

Así, proporcionando la id de la votación se obtiene un Json donde se muestra la clave de cifrado AES de esa votación.

Una vez realizado el trabajo descrito anteriormente, se procede a realizar un merge de la rama "Issue#3" con la rama "master". En eclipse debemos colocarnos sobre la rama "master", pulsar sobre el segundo botón del ratón, y después seleccionar "merge". En la ventana que aparece seleccionamos la rama "Issue#3" y pulsamos sobre "merge", como se muestra en la imagen:

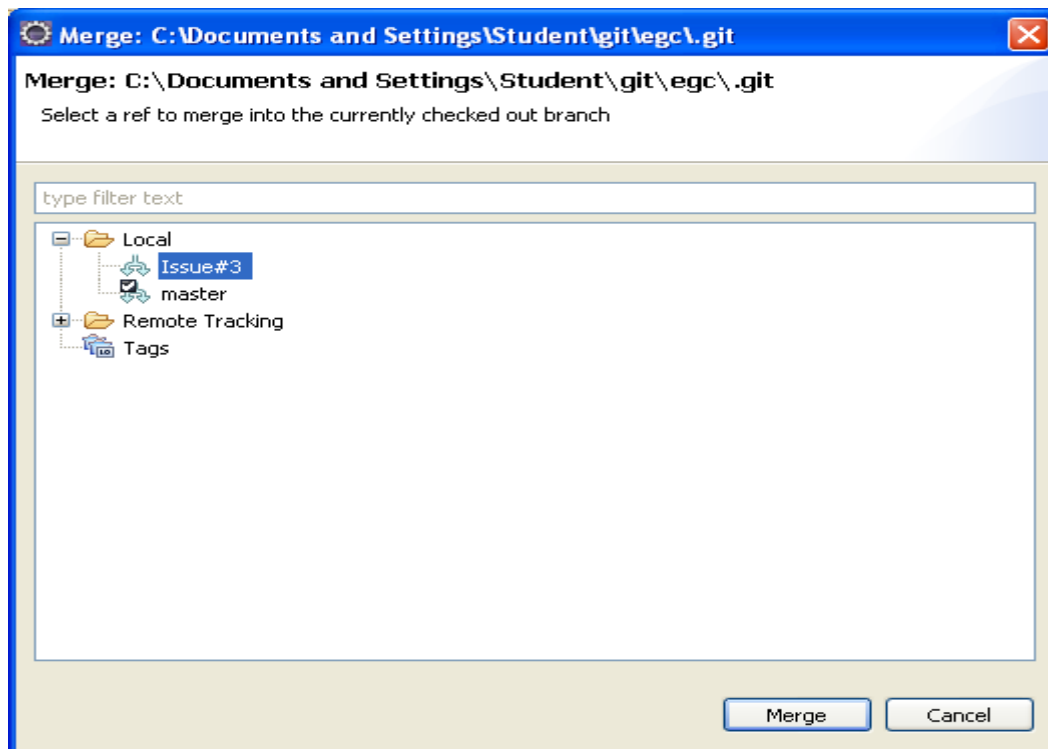


Ilustración 6. Realizando un merge en Eclipse

Por último, enviamos los cambios al servidor mediante la operación “push”:

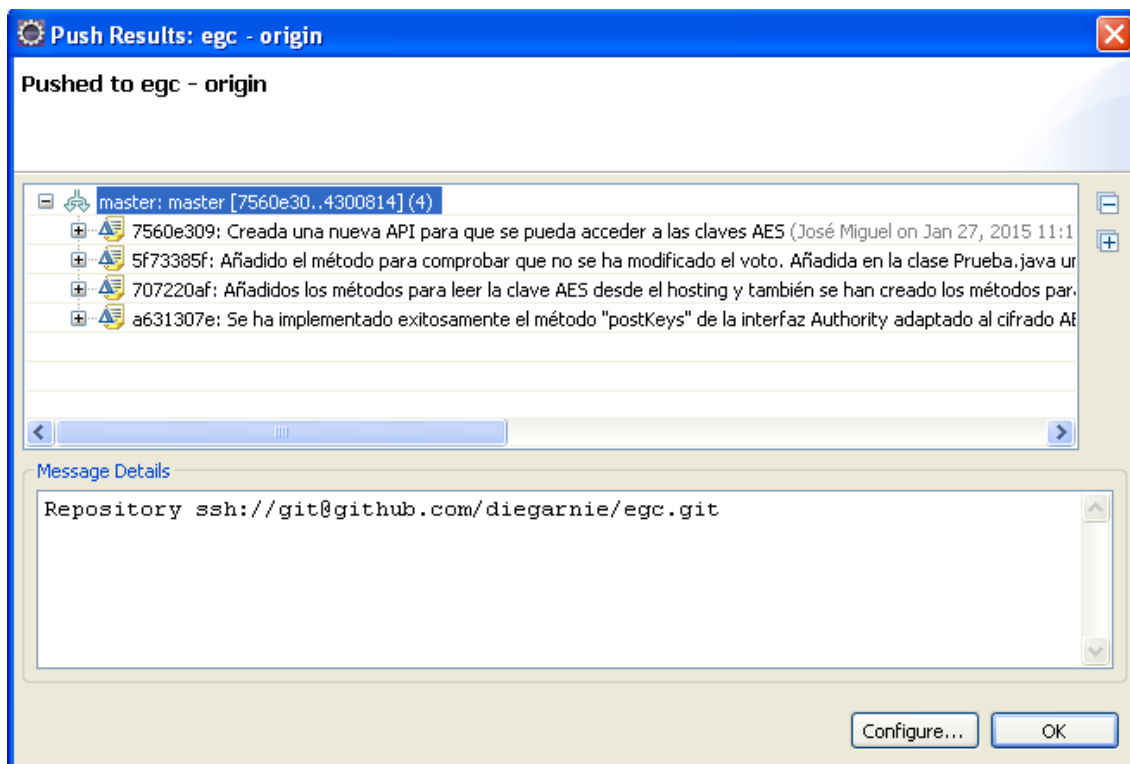


Ilustración 7. Resultado de la operación push.

En la anterior imagen se muestra el resultado de enviar los cambios al repositorio remoto. En este caso no ha sido necesario resolver ningún conflicto.

5. Gestión de Construcción e Integración Continua:

5.1. Integración Continua entre subsistemas

La integración se ha estado llevando a cabo de forma manual. Carecíamos de conocimientos en las herramientas de integración al empezar los talleres. No obstante se construyó un repositorio común para poder subir el código de cada subsistema con el fin de poder facilitar la compartición del trabajo.

Una vez supimos de las herramientas para facilitar la integración continua, decidimos no utilizarlas ya que la acomodación con las anteriores suponía un esfuerzo en tiempo y recursos al tratar de usar las nuevas.

La integración se ha estado realizando con un período de dos semanas aproximadamente. En cada taller un subgrupo de cada subsistema se reunía con los otros subgrupos de forma que acababan juntos al menos un representante de cada subsistema. Con esta forma de reparto, se podía poner en común el trabajo de cada subsistema para la integración del sistema total. Esto ayudaba a encontrar las incidencias y necesidades de cambio con los demás subsistemas, de forma que se pudiese dar una solución temprana a las posibles incompatibilidades.

Desde el primer momento, nuestra principal preocupación fue la de facilitar la integración de nuestro subsistema con el resto. Por ello, decidimos al inicio del proyecto utilizar Java plano para que cualquier subsistema pueda simplemente importar nuestra funcionalidad como un .jar.

De esta forma, nuestra integración con cada subsistema se hará de la siguiente forma:

- Con creación de votaciones: al crear la votación, llamará a nuestro método "postKey" pasando como parámetro el "id" de dicha votación (un String). Crearemos el par de claves para esa votación y lo guardaremos en nuestra base de datos.
- Con cabina de votación: la integración con cabina de votación es especial, ya que su sistema está basado en python. Ante la dificultad que reportaron al importar nuestro archivo jar, se creó una API para poder realizar la integración con este subsistema. Ésta API consta de un pequeño script PHP que solicita la clave pública de la votación cuyo 'id' ha sido pasado como parámetro y la devuelve en formato Json. Los datos son consultados de la misma base de datos en la que son almacenados a través de los métodos del jar, por lo que si se crea una nueva clave de cifrado desde código java será visible a través de la API desde el mismo momento de su creación. La dirección a través de la cual podemos invocar a la Api es la siguiente: <http://www.egcprueba.esy.es/getKeys.php?id={dabaBaseID}>.

El parámetro representa la id de la base de datos de la que queremos conocer la clave pública. La salida se devuelve en formato Json y tiene únicamente el campo "PublicKey" seguido de su valor.

- Con recuento y modificación: cuando necesiten descifrar un voto, primero deberán verificar que no ha sido modificado con el método “checkVote”, el cual devuelve una salida booleana (True en el caso de que el voto permanezca inalterado, y False si ha habido alguna alteración), consiguientemente llamarán a nuestro método “getPrivateKey” con la “id” de la votación en la que se encuentre (un String) y descifrarán usando dicha clave que les proporcionamos.

El modelo de integración que hemos llevado a cabo es el modelo basado en “Sandwich”

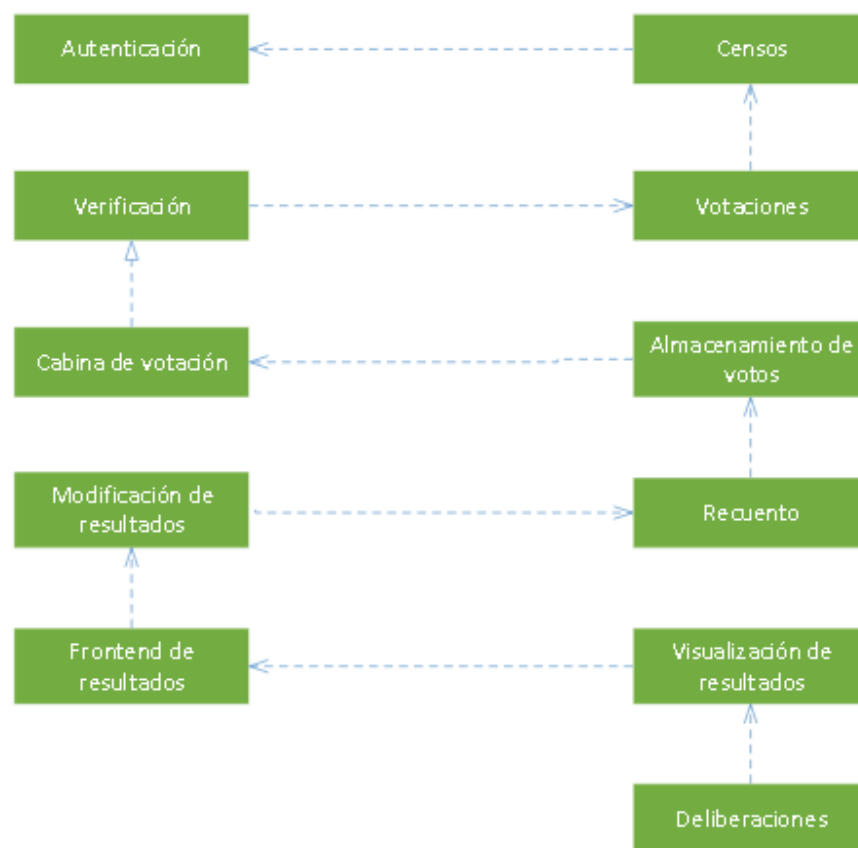


Ilustración 8. Integración modelo sandwich

5.2. Integración continua en nuestro subsistema.

En lo relativo a la integración continua dentro de nuestro subsistema, se usará la herramienta Jenkins junto con Maven para automatizar los test del proyecto, de modo que si algún desarrollador hace cambios que produzcan errores en los test, se detecte rápidamente, posibilitando una corrección rápida del código.

El proceso es el siguiente: **cada vez que se realice un nuevo aporte al repositorio en la rama master**, Jenkins descargará la última versión subida al repositorio Git del subsistema y procederá a la ejecución de los tests unitarios que incluya el proyecto, usando para ello la herramienta Maven. Maven generará un informe sobre la ejecución de los test, que usará Jenkins para mostrar los resultados de dichos tests:

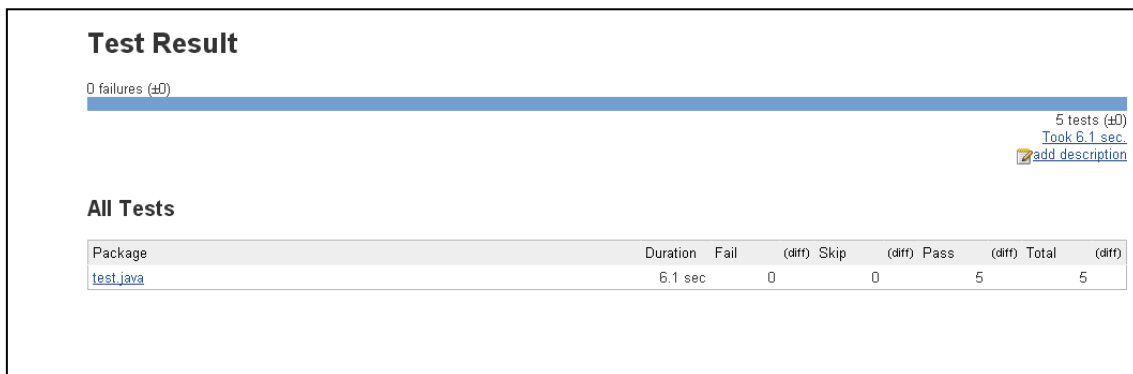


Ilustración 9. Resultados de la ejecución de tests

De este modo, cualquier cambio que impida que los test sean ejecutados correctamente, podrá ser detectado **rápidamente**, ya que Jenkins nos informaría de la situación.

Por otra parte, también mediante la herramienta Jenkins, se subirá automáticamente al repositorio de nuestro subsistema el archivo Jar generado a partir del código de la rama master. Somos conscientes de que esto puede ser una mala práctica, ya que el sistema de control de versiones no es eficiente trabajando con archivos binarios y además estos archivos pueden ser generados a partir del código que reside en el repositorio, pero queríamos disponer de un archivo Jar que esté siempre actualizado y accesible desde cualquier ordenador por cada miembro del grupo, para evitar posibles errores al usar un archivo Jar pensando que es la última versión.

De este modo, cada vez que la rama master cambie, Jenkins se ejecutará automáticamente y mediante Maven generará un archivo Jar que subirá a la rama master del repositorio si todos los test unitarios se ejecutan correctamente.

5.3. Caso práctico

Vamos a realizar dos casos prácticos, en el primero de ellos se realizará un ejemplo en el que añadimos un nuevo test a nuestro sistema de verificación y captamos un error gracias a la ejecución automática de Jenkins.

En el segundo ejemplo práctico, se va a realizar un cambio en la rama master y seguidamente se obtendrá el fichero Jar del repositorio que ha creado Jenkins, comprobando que está actualizado.

Comenzamos con el primer caso práctico. El escenario es el siguiente: un desarrollador del equipo hará un push al repositorio en el que hace fallar la correcta ejecución de uno o varios tests. Al cabo de un rato, el desarrollador accede a Jenkins para ver el estado del proyecto y observa que existe algún test que está fallando, con lo que investiga y observa que cometió un error en su último aporte al repositorio. Por último, el desarrollador corrige el error y observa que Jenkins ahora sí muestra la correcta ejecución de todos los tests.

Para recrear esta situación, en primer lugar vamos a subir al repositorio una nueva versión que provocará el fallo de algún test:

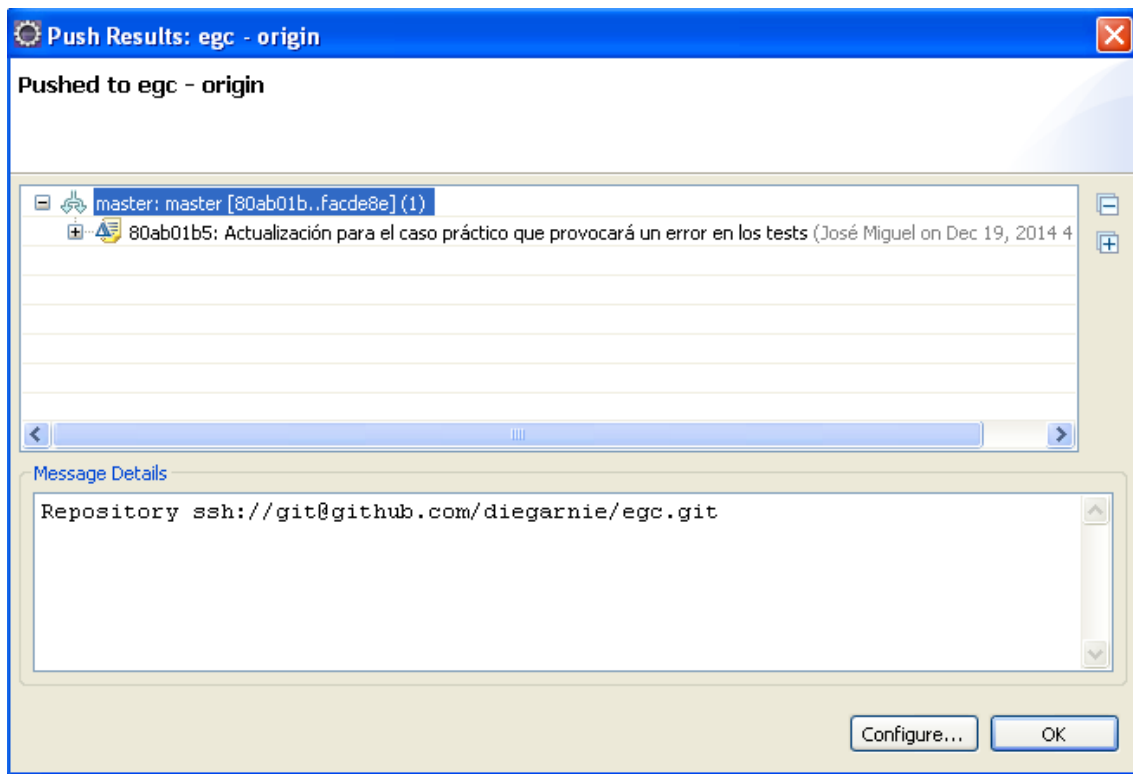


Ilustración 10. Fallo subido para provocar un error

Esperamos a que se ejecute la tarea programada e ingresamos en Jenkins para ver el resultado. Como vemos en el informe, se ha producido un fallo en uno de los tests:

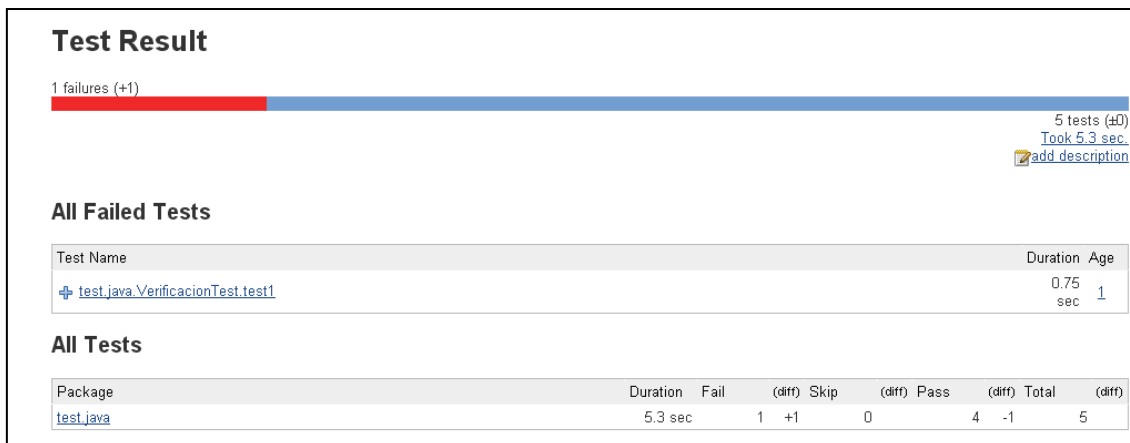


Ilustración 11. Error en la ejecución de los tests

Vemos los detalles de la traza y comprobamos la causa del error:



Ilustración 12. Traza de error

Ahora es cuando nos damos cuenta de que nuestro último aporte al repositorio produjo este error, con lo que pasamos a realizar la modificación para solucionarlo y enviar los cambios al repositorio.

Una vez enviados estos cambios, nos vamos a Jenkins y entramos en nuestro proyecto. Pulsamos sobre construir ahora para que se ejecuten los test y vemos que el error ha sido resuelto:

Test Result

0 failures (-1)

5 tests (#0)

[Took 7 sec.](#)
[add description](#)


All Tests


Package	Duration	Fail	(diff)	Skip	(diff)	Pass	(diff)	Total	(diff)
test.java	7 sec	0	-1	0		5	+1	5	

Ilustración 13. Error solventado

Así, gracias a Jenkins se ha detectado el error de forma rápida, lo que facilita su corrección.

Seguimos con el segundo ejemplo práctico en el que obtendremos el archivo Jar actualizado justo después de la corrección de un bug de nuestro subsistema. Se trata del Issue#6, en el que se indica que al obtener una clave de cifrado AES cualquiera, la primera posición de dicha clave siempre contiene un espacio en blanco y esto no debería ser así. Una vez que el desarrollador procede a la corrección del bug (el procedimiento será descrito en el apartado de depuración), y tras enviar los cambios al repositorio remoto, Jenkins creará el Jar actualizado y lo subirá al repositorio. En la imagen se observa que tras el envío del cambio al repositorio, se genera un nuevo envío automático mediante Jenkins con el Jar actualizado:


JAR generado y subido mediante Jenkins
 Jenkins authored 2 hours ago
 8e378cc


Solucionado el bug descrito en el ...
 jmleon authored 2 hours ago
 4806370

Issue#6(#6).
 También se ha completado la batería de tests ya que no existían tests relacionados con el cifrado AES.

Ilustración 14. Commit realizado por Jenkins.

A continuación nos dirigimos a la carpeta Jar del repositorio para obtener el archivo recién generado. Pulsamos sobre el archivo binario y seguidamente sobre la opción “View Raw” que nos permitirá descargar el archivo. Tras descargar el archivo vamos a proceder a importarlo en eclipse y a llamar a la función `getSecretKey` de la clase `AuthorityImplAES` para verificar que no se incluye un espacio en blanco al obtener una clave de cifrado AES (este espacio se incluía antes de reparar el bug como veremos en el apartado de depuración), con lo que comprobaríamos que el funcionamiento del Jar está actualizado.

```

Problems  @ Javadoc  Declaration  Console  [X]
<terminated> test [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (02/02/2015 21:31:16)
Clave Obtenida:bmV2q1MarBMfNNvVq+AQZ1MhjRwaBHpVaMzAwBwM1B0=
  
```

Ilustración 15. Ejecución del método `getSecretKey` tras importar el Jar.

Efectivamente, en la imagen se aprecia que tras la cadena “Clave Obtenida:” se concatena la clave sin un espacio intermedio, con lo que hemos comprobado que el funcionamiento del Jar que acabamos de descargar ha sido actualizado varios minutos después de la reparación del error.

5.4. Construcción

En cuanto a la construcción, tal y como se ha explicado anteriormente, se proporcionará un .jar el cual contendrá toda la funcionalidad conveniente de nuestro subsistema. Así mismo se aportará lo necesario para instalar y configurar el conector de MySQL para Java.

Las herramientas que se han usado para la construcción son las siguientes:

- **IDE de Eclipse V4.4 (Luna):** utilizado para la implementación, construcción y depuración del código, así como para la generación del .jar.
- **Base de datos MySQL en Nube:** para ello se ha usado un servicio gratuito en Hostinger.es. Lo deseable sería haber conectado directamente nuestro programa java con una base de datos remota, pero este servicio es de pago, por lo que se ha usado un script php intermedio actuando como api, que también podrá ser usado por otros grupos que no usen java para obtener las claves de cifrado.

5.5. Caso práctico

El caso práctico siguiente trata de importar el .jar que ofrecemos a los demás subsistemas para conseguir la integración, descargándolo de GIT de nuestro repositorio e importándolo a Eclipse. Una vez descargado el .jar de nuestro repositorio procedemos con la correcta instalación del mismo. En primer lugar abrimos nuestro eclipse, en el cual debemos tener el proyecto al que queremos añadir nuestro subsistema. Para este ejemplo se ha creado un proyecto ficticio en el cual se va a explicar paso por paso como importar nuestro .jar y hacer uso de la funcionalidad que ofrece.

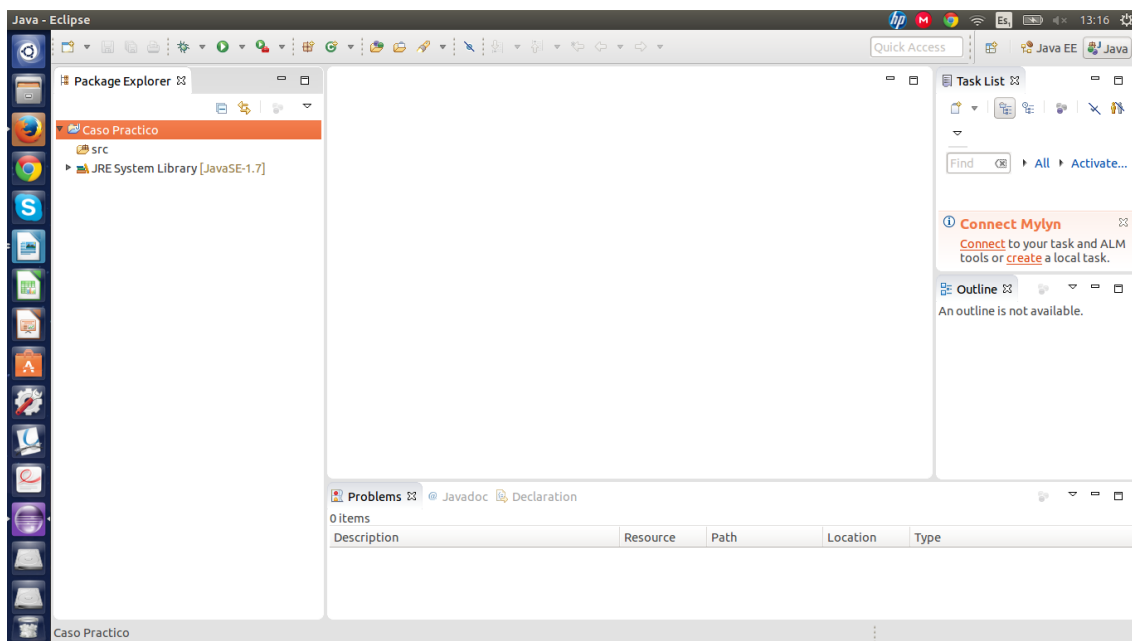


Ilustración 16. Vista general proyecto de prueba

Ahora pinchamos sobre nuestro proyecto con el botón derecho y seleccionamos “Propiedades” en el menú desplegable que se mostrará. Al hacer esto se nos mostrará una nueva ventana en la cual debemos buscar la pestaña “Java Build Path” en la parte izquierda de la misma (Si no se encuentra, se puede escribir en el buscador que aparece en la parte superior izquierda). Después seleccionamos la pestaña “Librerías”. Y pulsamos sobre “Add External Jars”.

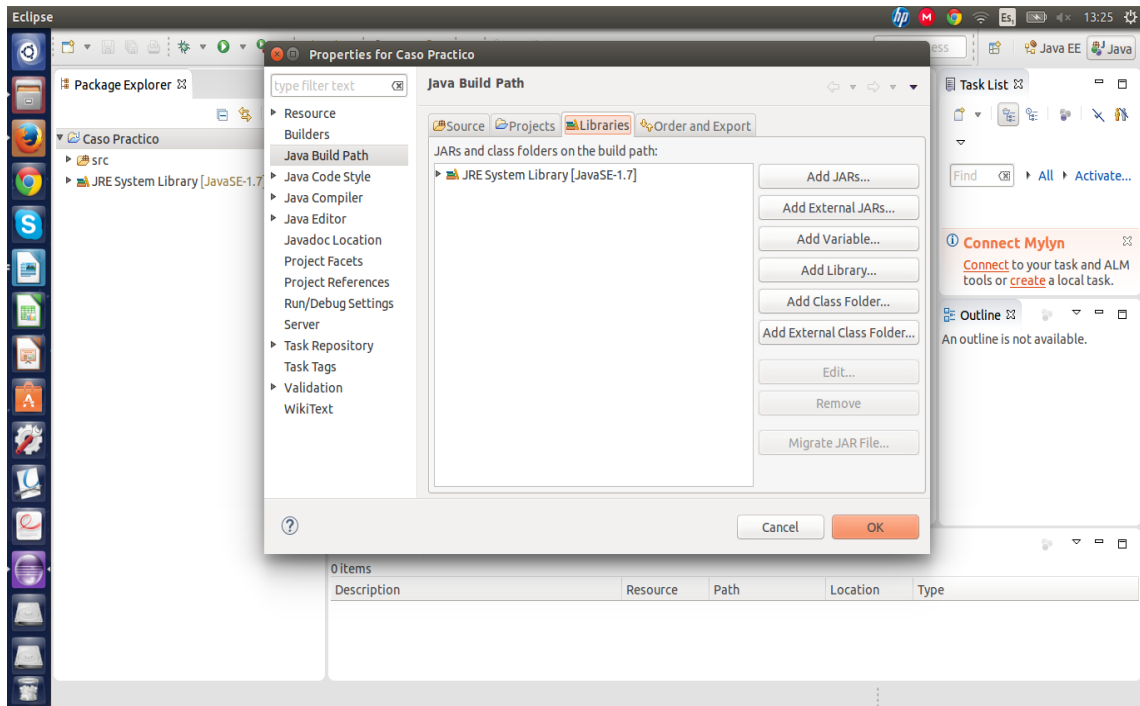


Ilustración 17. Propiedades del proyecto

En la nueva ventana debemos buscar donde se encuentra descargado nuestro .jar y seleccionarlo, en la ventana anterior de librerías debería mostrarse el nuevo .jar que hemos importado.

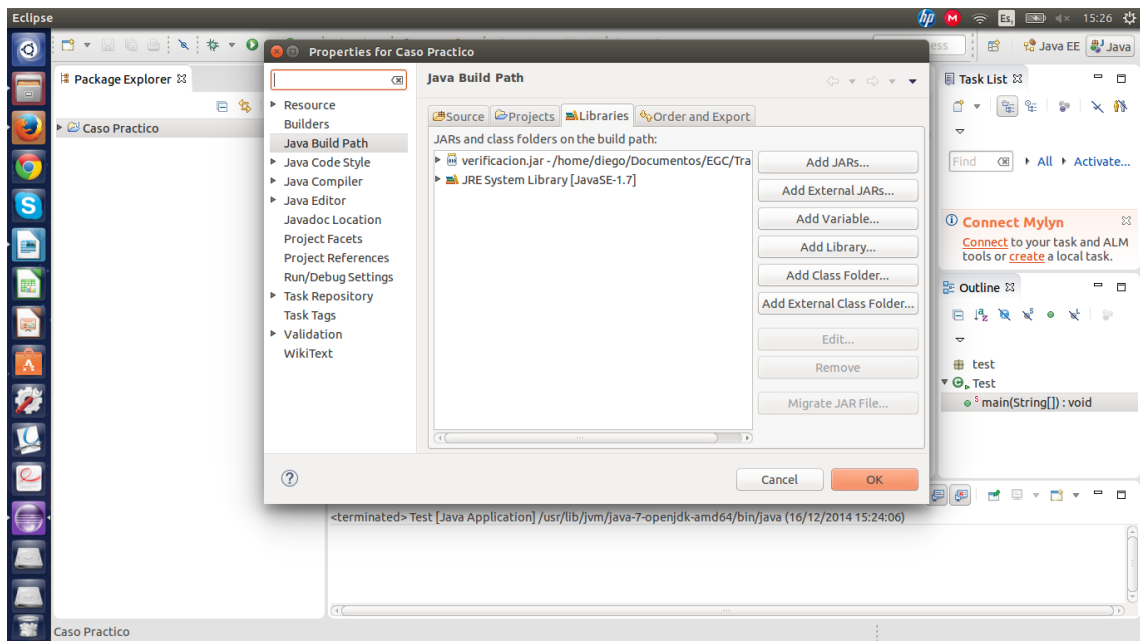


Ilustración 18. Añadir .Jar

Bien una vez hecho esto nuestro subsistema debería estar ya correctamente integrado. A continuación se expondrán capturas del código de un pequeño test que demuestra que todo está correcto y no ha habido ningún tipo de problema:

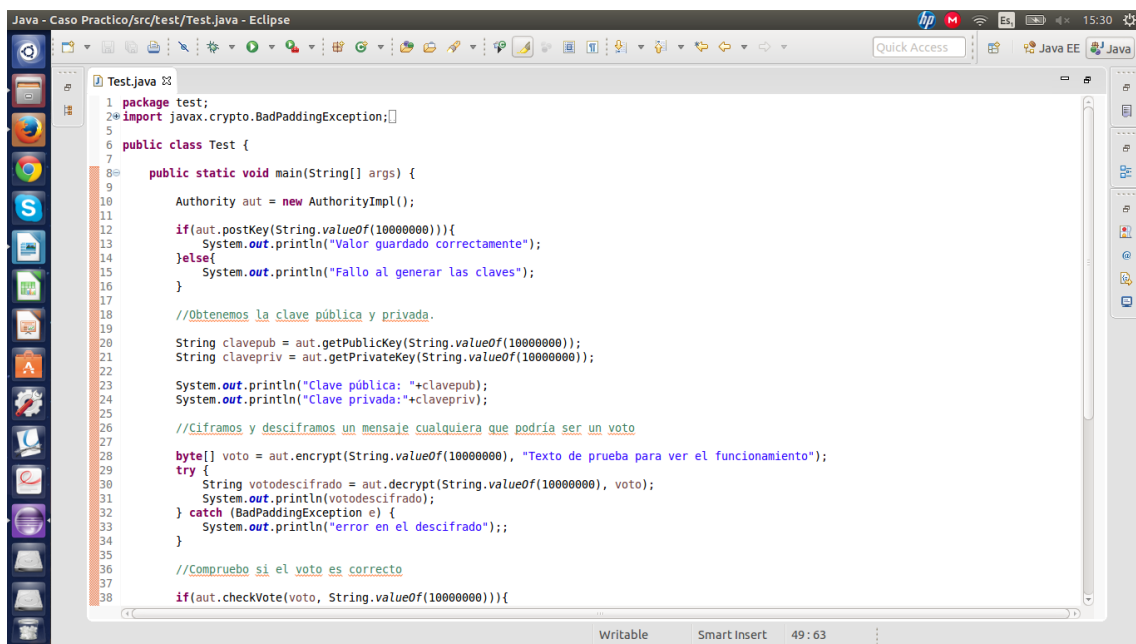
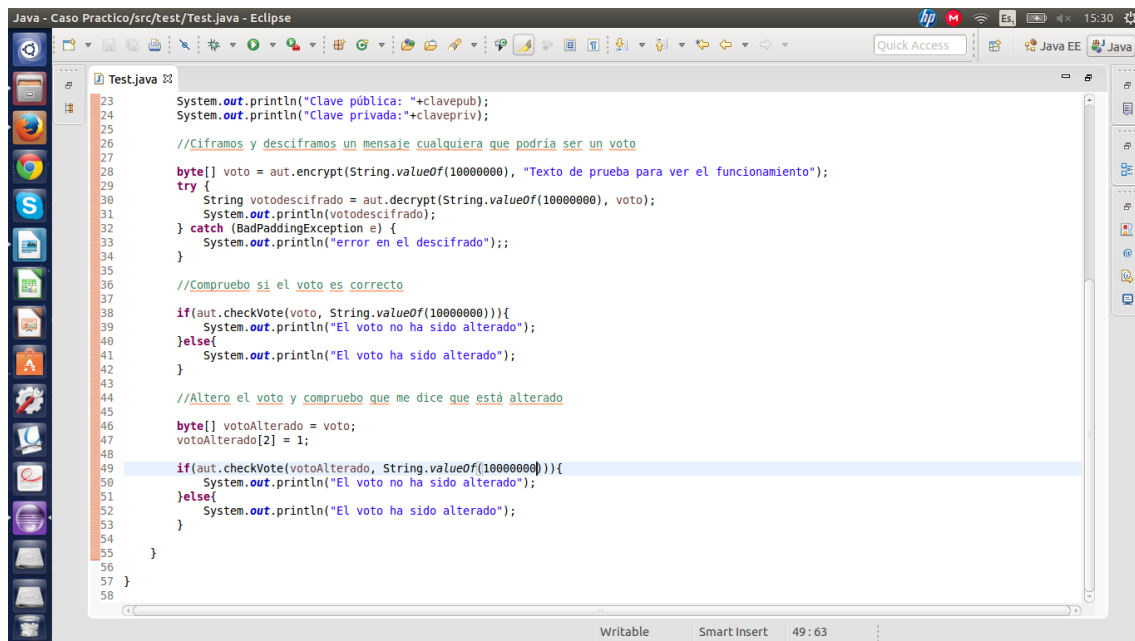


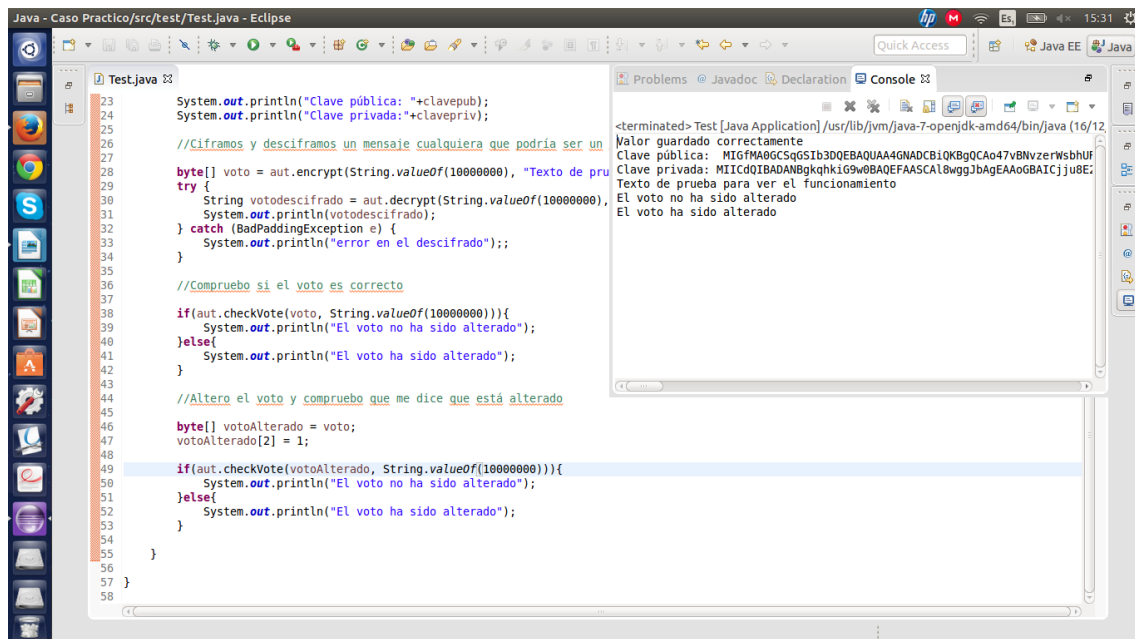
Ilustración 19. Clase Test I



```
23 System.out.println("Clave pública: "+clavepub);
24 System.out.println("Clave privada: "+clavepriv);
25
26 //Ciframos y desciframos un mensaje cualquiera que podría ser un
27
28 byte[] voto = aut.encrypt(String.valueOf(10000000), "Texto de prueba para ver el funcionamiento");
29 try {
30     String votodescifrado = aut.decrypt(String.valueOf(10000000), voto);
31     System.out.println(votodescifrado);
32 } catch (BadPaddingException e) {
33     System.out.println("error en el descifrado");
34 }
35
36 //Compruebo si el voto es correcto
37
38 if (aut.checkVote(voto, String.valueOf(10000000))) {
39     System.out.println("El voto no ha sido alterado");
40 } else {
41     System.out.println("El voto ha sido alterado");
42 }
43
44 //Altero el voto y compruebo que me dice que está alterado
45
46 byte[] votoAlterado = voto;
47 votoAlterado[2] = 1;
48
49 if (aut.checkVote(votoAlterado, String.valueOf(10000000))) {
50     System.out.println("El voto no ha sido alterado");
51 } else {
52     System.out.println("El voto ha sido alterado");
53 }
54
55 }
56
57 }
58
```

Ilustración 20. Clase Test II

Y la salida que muestra al ejecutar el test es la siguiente:



```
<terminated> Test [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (16/12)
Valor guardado correctamente
Clave pública: MIIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCAo47vBNvzerWsbhUF
Clave privada: MIICdQIBADANBgkqhkiG9w0BAQEFAASCAL8wggJbAgEAAoGBAICjju8E
Texto de prueba para ver el funcionamiento
El voto no ha sido alterado
El voto ha sido alterado
```

Ilustración 21. Salida del Test

Como podemos ver todos los métodos han sido correctos y no se ha producido ningún error.

También vamos a mostrar cómo se realizaría la implementación de nuestro .jar en aquellos subsistemas que utilizan Maven, pues el procedimiento es distinto. A diferencia del anterior ejemplo este se ha realizado en Windows 7, y hemos creado un proyecto en Maven sencillo para probar esto.

Lo primero que tenemos que hacer es instalar Maven en eclipse (el que viene instalado por defecto no vale). Esto se puede hacer desde el eclipse Marketplace.

Después instalamos dependencias al repositorio, para ello usaremos un comando de Maven que es el siguiente:

```
mvn install:install-file -Dfile=<path-to-file> -DgroupId=<group-id> -DartifactId=<artifact-id> -Dversion=<version> -Dpackaging=<packaging>
```

<path-to-file>: ubicación de nuestro jar, en nuestro caso es C:\Users\Superwaki\Documents\Universidad\EGC\Trabajo\verificacion.jar

<group-id>: groupId del jar, en este ejemplo hemos puesto verificación.

<artifact-id>: artifactId, hemos puesto main.

<version>: versión del jar, se puede poner una cualquiera, aunque se recomienda poner la versión del jar.

<packaging>: es el packaging del archivo que en nuestro caso es jar

Para ejecutarlo podemos seguir los siguientes pasos. Primero pinchamos en el proyecto con el botón derecho del ratón, seleccionamos Run As y Run Configurations.

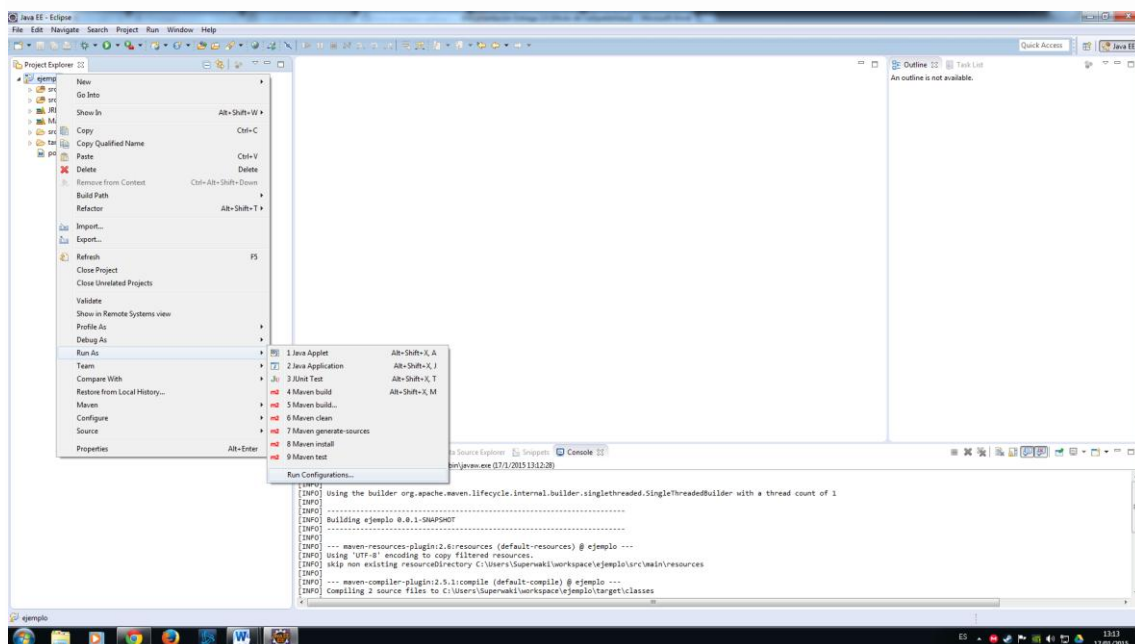


Ilustración 22.Run As/Run Configurations.

Nos saldrá una ventana, si no debemos crear un nuevo Maven Build en el cual pondríamos el proyecto y en el campo goal el comando anterior pero sin mvn al principio.

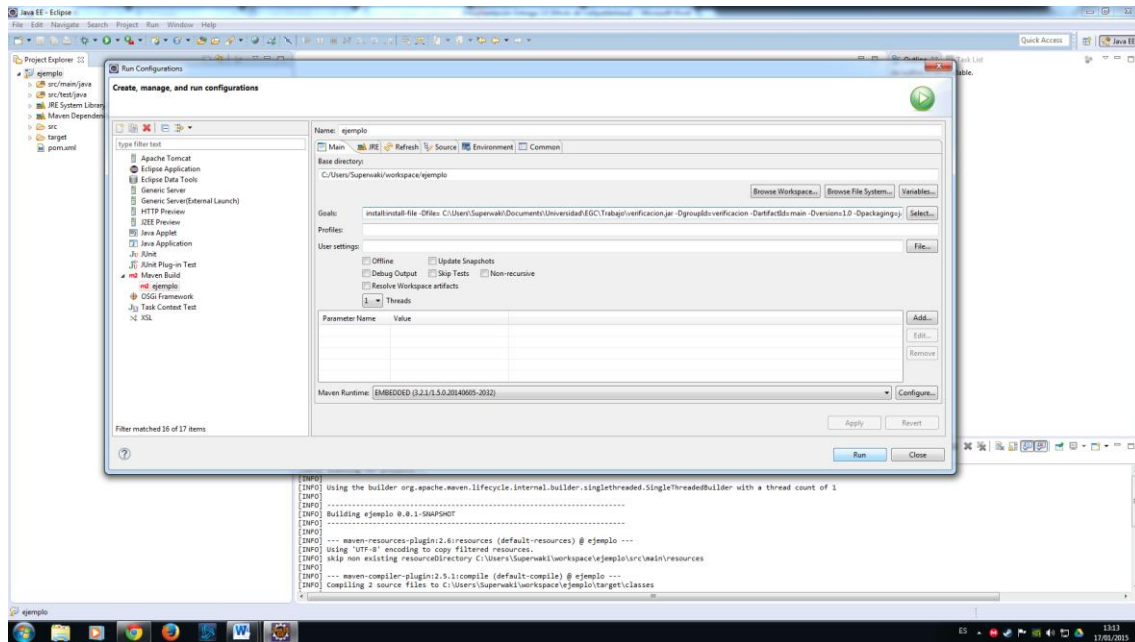


Ilustración 23. New/Maven Build.

Le damos a run y si todo va bien, debería haber una salida parecida a esta:

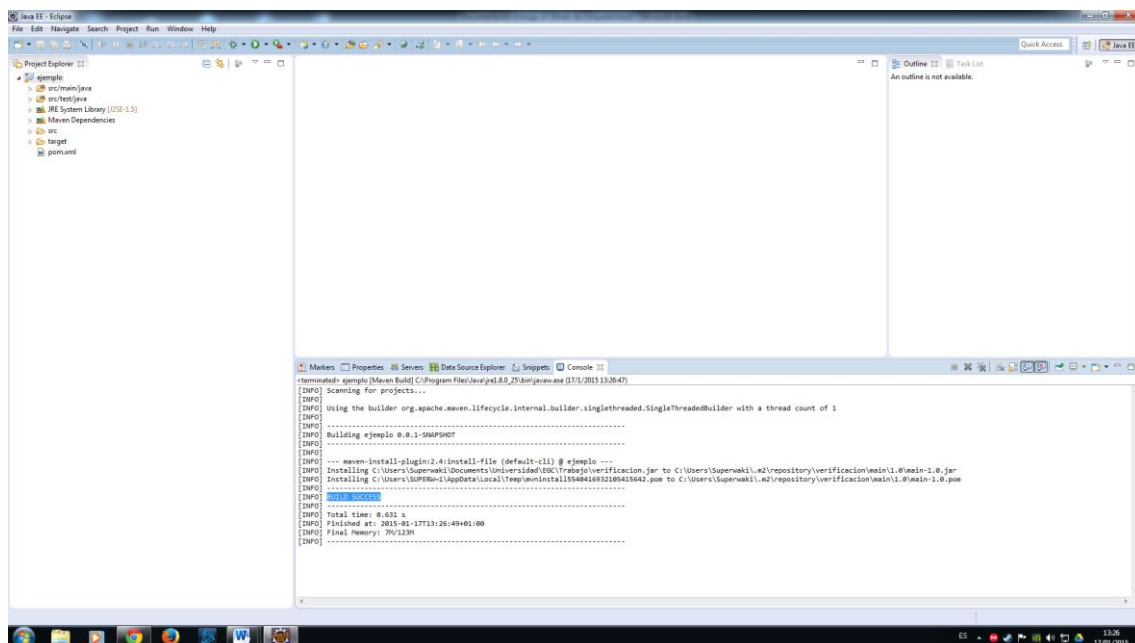


Ilustración 24. Salida en Consola.

Después de hacer esto, debemos añadir unas líneas en nuestro fichero pom.xml:

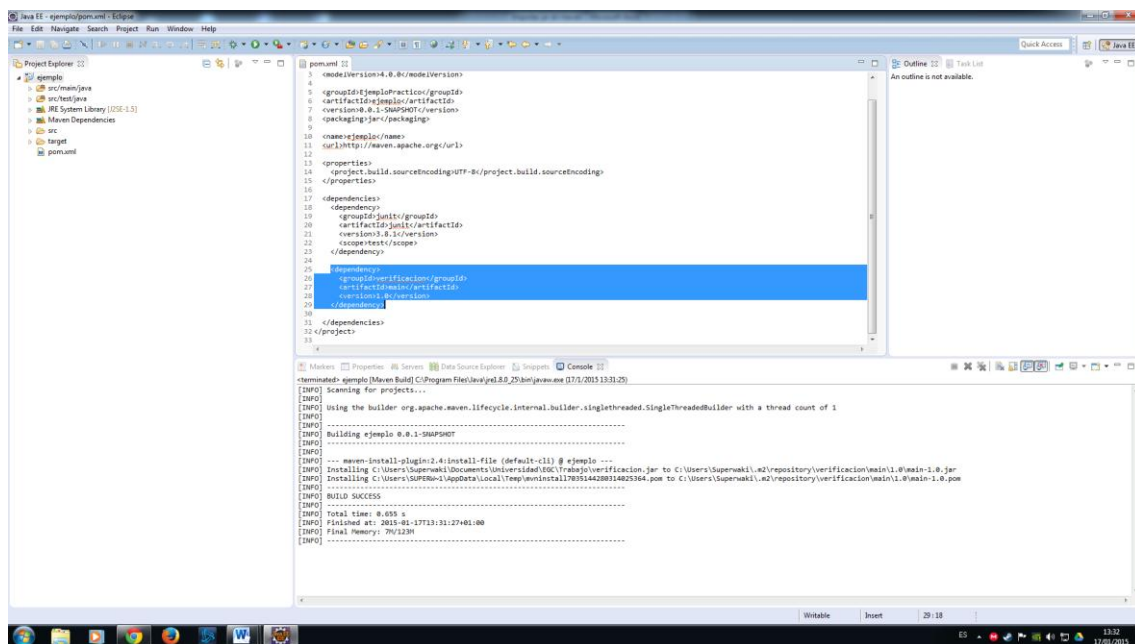


Ilustración 25. Fichero pom.xml.

Tras hacer todos estos pasos, el jar estará correctamente añadido al proyecto y se podrá hacer uso de él, como se muestra en la siguiente captura:

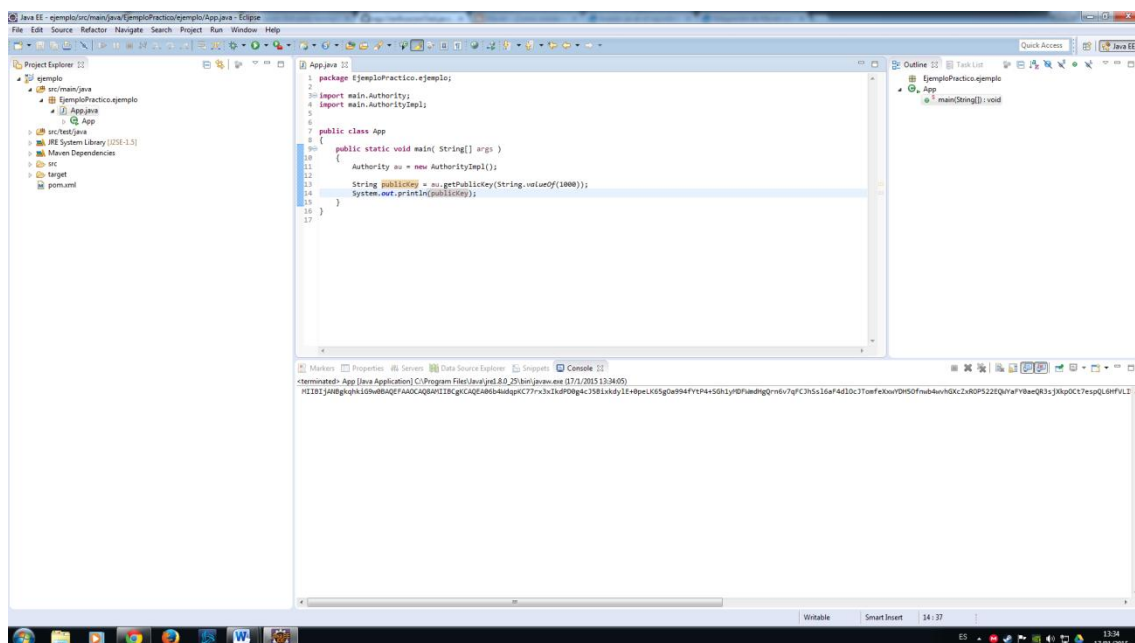


Ilustración 26. JAR correctamente añadido.

Se puede ver como usamos `Authority` y `AuthorityImpl`, además usamos `getPublicKey` para obtener una clave ya existente y la mostramos por consola.

5.6. Integración de todos los subsistemas

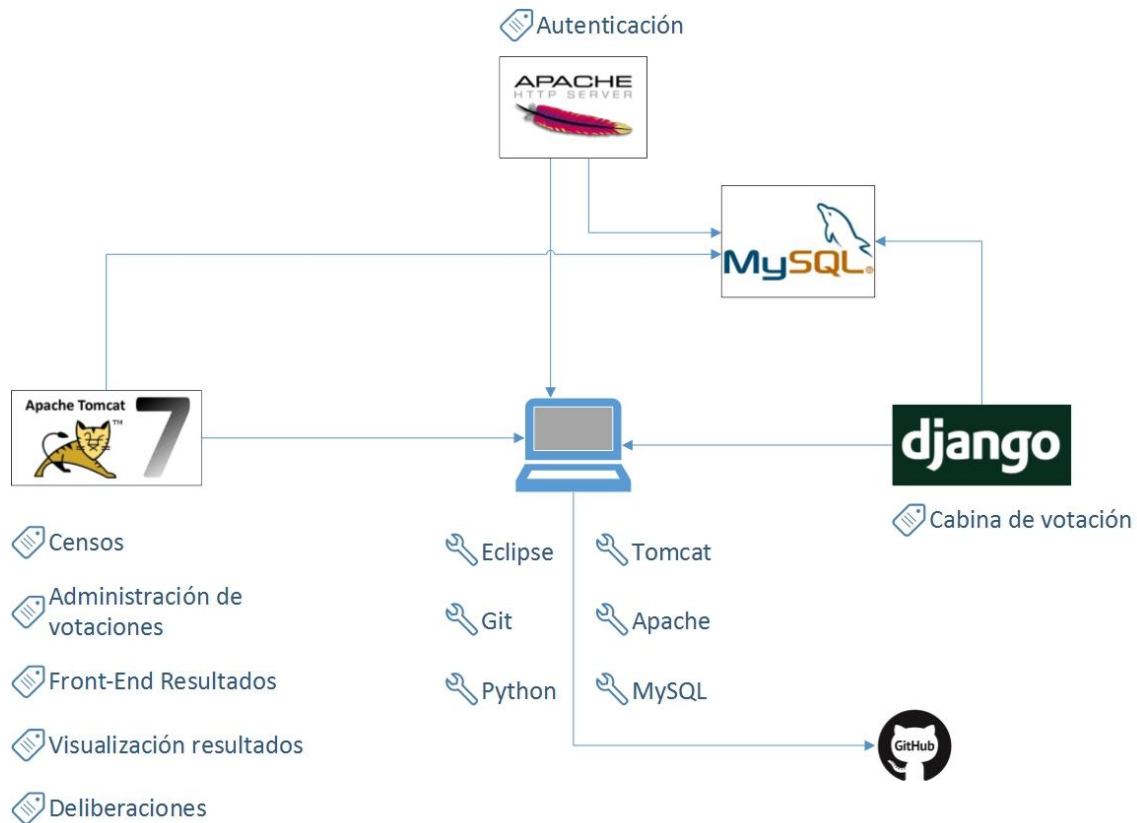


Ilustración 27. Esquema despliegue subsistemas

La integración final de todos los subsistemas se ha llevado a cabo en una sola máquina.

Para hacerlo posible ha sido necesario instalar las diferentes herramientas que enumeramos a continuación:

- Python 2.7 y diversas dependencias: Esta herramienta es usada por Cabinas de votación y el proceso de despliegue se encuentra desarrollado en la wiki ([http://1984.lsi.us.es/wiki-egc/index.php/Grupo_de_Cabina_de_votaci%C3%B3n_\(2014-15\)](http://1984.lsi.us.es/wiki-egc/index.php/Grupo_de_Cabina_de_votaci%C3%B3n_(2014-15)))
- Tomcat 7: Como podemos observar en la figura anterior la mayoría de los sistemas se han desplegado en un servidor de Tomcat que fue configurado para usar el puerto 8081 para las peticiones HTTP.
- Apache: En el caso de Autenticación se usó el servidor de Apache que viene en el paquete de herramientas XAMPP configurado para usar el puerto 8080.
- MySQL: Contiene diversas bases de datos, prácticamente una por cada subsistema a integrar sin contar aquellos que usen algún tipo de hosting como nuestro subsistema.

En el caso de autenticación bastó con copiar sus archivos a la carpeta “htdocs” en el directorio de XAMPP. Los subsistemas que han usado Spring, excepto Modificación, se han desplegado a través de la interfaz de gestión de Tomcat (<http://localhost:8080/manager>); en el caso del subsistema de modificación el despliegue se lleva a cabo tal y como se explica en el archivo “README.md” que aparece en su carpeta del repositorio común.

6. Gestión de la calidad

6.1. Instalación y Configuración.

En cuanto a la gestión de la calidad, hemos decidido utilizar la herramienta SonarQube. Esta herramienta nos permite la elaboración automática de informes acerca de la calidad del código que estamos desarrollando. La utilizamos para saber si el código que implementamos tiene una calidad suficiente o no, teniendo en cuenta unos criterios y unas métricas que explicaremos un poco más adelante.

Antes de comenzar, explicaremos brevemente cómo hemos configurado esta herramienta para utilizarla en nuestro proyecto.

En primer lugar, hemos creado una base de datos MySQL necesaria para el funcionamiento de Sonar. El script para crear la base de datos es el siguiente:

```
CREATE DATABASE sonarqube CHARACTER SET utf8 COLLATE utf8_general_ci;
```

```
CREATE USER 'sonarqube' IDENTIFIED BY 'sonarqube';
```

```
GRANT ALL ON sonarqube.* TO 'sonarqube'@'%' IDENTIFIED BY 'sonarqube';
```

```
GRANT ALL ON sonarqube.* TO 'sonarqube'@'localhost' IDENTIFIED BY 'sonarqube';
```

```
FLUSH PRIVILEGES;
```

A continuación, descargamos “sonarqube-4.0” y configuramos el archivo “sonar.properties” localizado en la ruta “My Documents\Downloads\sonarqube-4.0\conf”. Cambiamos e indicamos valores del servidor de Sonar y de la base de datos MySQL creada previamente.

Instalamos el servidor en local y lo ejecutamos mediante consola a través del archivo situado en la ruta “My Documents\Downloads\sonarqube-4.0\bin\windows-x86-64\StartSonar.bat”. Accederíamos al servidor de Sonar mediante la URL siguiente: <http://localhost:9000/>. La vista que debería presentar es la siguiente:

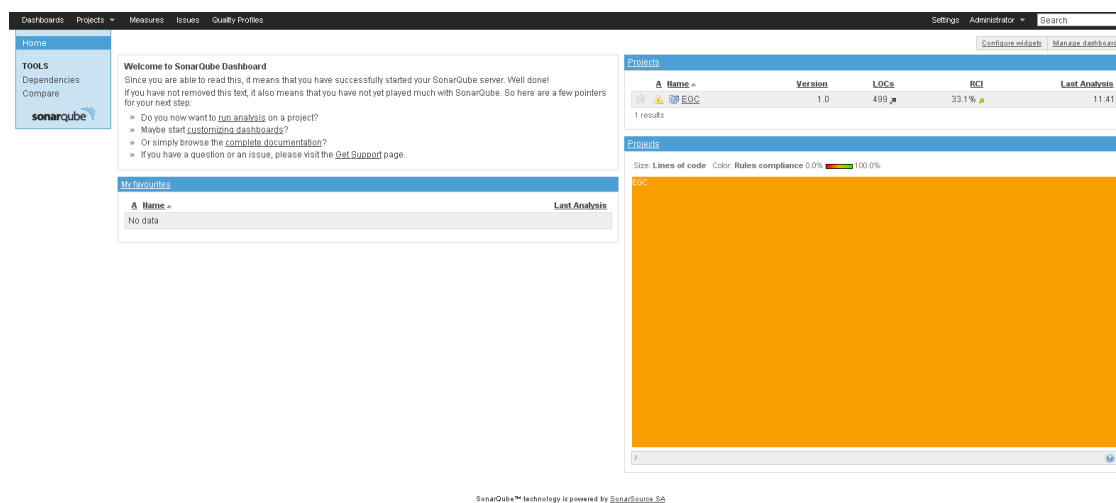


Ilustración 28. Vista general de SonarQube

Ahora procedemos a integrar Sonar con Jenkins, para ello instalamos el plugin de Sonar para Jenkins. Es el siguiente:

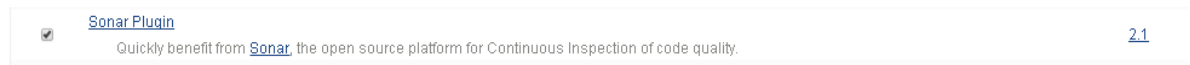


Ilustración 29. Plugin de Sonar para Jenkins

Tenemos que instalar Sonar Runner herramienta adicional necesaria para analizar código, lo hacemos desde Jenkins y le decimos que lo instale automáticamente:

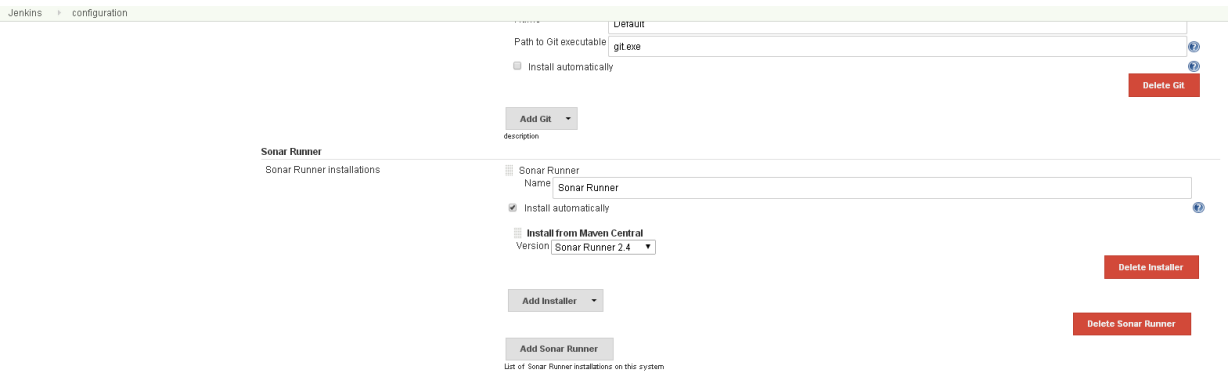


Ilustración 30. Configuración de Sonar Runner en Jenkins

Aparte debemos proporcionar información sobre la base de datos y servidor de Sonar, para ello rellenamos los campos del apartado Sonar dentro de la configuración de Jenkins:

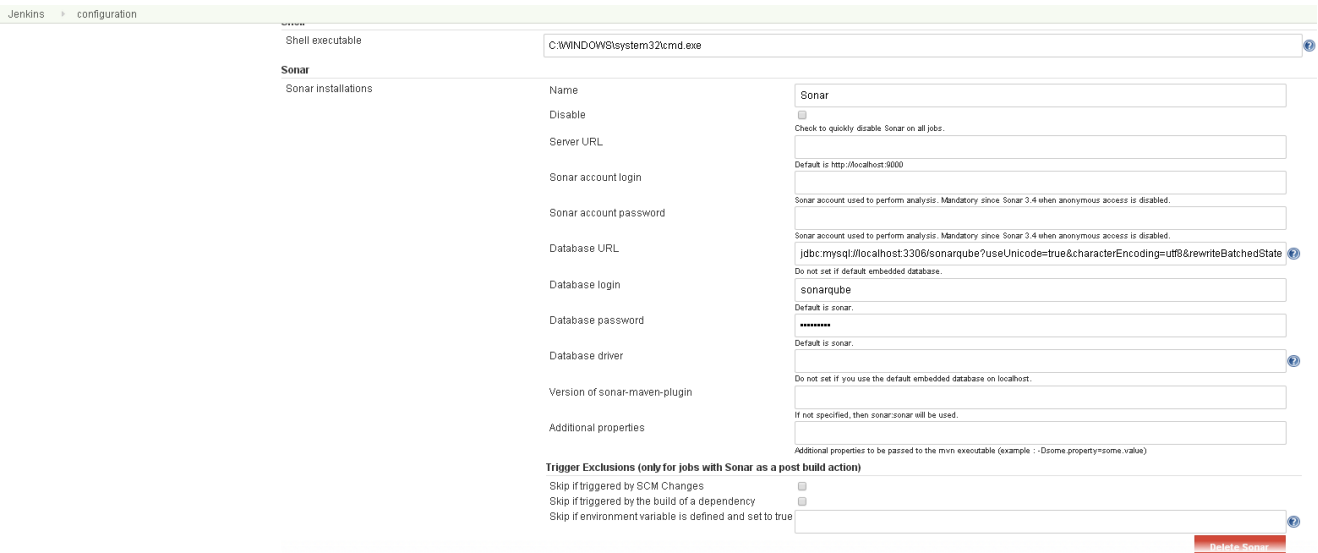


Ilustración 31. Configuración de Sonar en Jenkins

Habríamos resuelto la conexión entre Sonar y Jenkins, sólo faltaría añadir un paso en nuestra tarea de Jenkins que analice el código. Lo hemos hecho de la siguiente forma:

- En primer lugar, vamos a añadir en la ventana de configuración de nuestra tarea de Jenkins un nuevo paso del tipo “Invoke Standalone Sonar Analysis”.

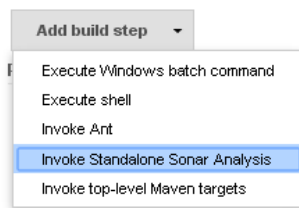


Ilustración 32. Paso para añadir Sonar

- Procedemos a rellenar la información del paso que acabamos de crear. Solo habría que indicar las propiedades del proyecto. Le decimos un nombre y una clave de proyecto que elegimos nosotros, la versión (por defecto 1.0), lenguaje del código (en nuestro caso java) y la ruta donde se encuentran los archivos del proyecto. En concreto, esta ruta es aquella que utiliza Jenkins para guardar los archivos del proyecto.



Ilustración 33. Datos del paso de Sonar Análisis

- Por último, ejecutamos la tarea de Jenkins y vemos los resultados que nos proporciona sobre la calidad de nuestro código.

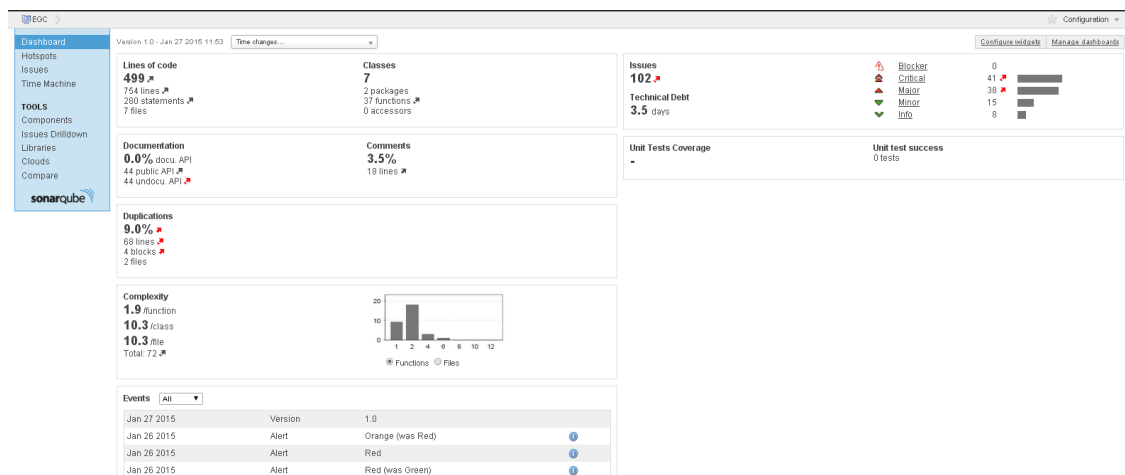


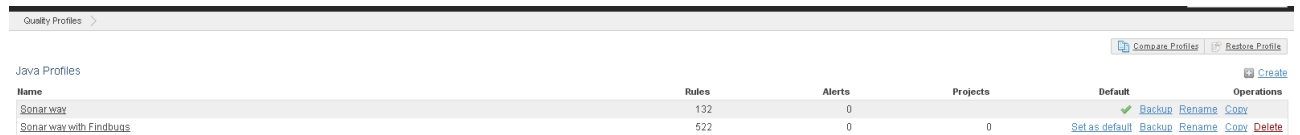
Ilustración 34. Mediciones previas al estudio

Nota: para la configuración de Sonar (servidor y base de datos) nos hemos basado en las páginas web indicadas en la bibliografía, aunque para configurar el plugin para Jenkins no tuvimos que hacer uso de ningún tutorial porque se averiguó probando e investigando.

6.2. Configuración de perfiles de calidad e interpretación de métricas.

Sonar ofrece infinidad de funcionalidades, pero nosotros nos vamos a centrar en modificar perfiles de calidad e interpretar las métricas.

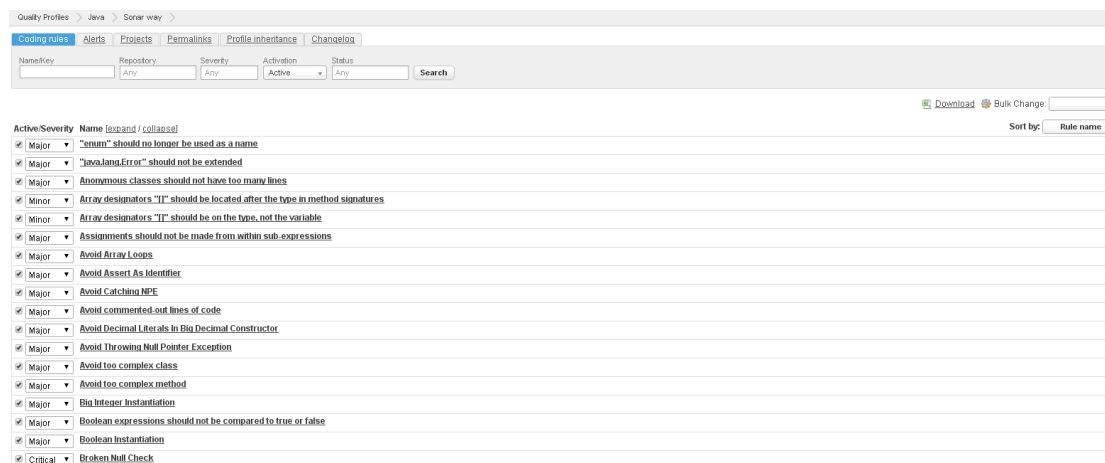
En primer lugar, vamos a modificar el perfil de calidad que viene por defecto para java y vamos a establecer diferentes umbrales para diversos aspectos del código seleccionables. Entramos como usuario administrador (en nuestro caso, username: admin password: admin) y nos vamos al apartado “perfiles de calidad”:



Quality Profiles >					
Java Profiles					
				Compare Profile	Restore Profile
					Create
Name	Rules	Alerts	Projects	Default	Operations
Sonar way	132	0		✓	Backup Rename Copy
Sonar way with Findbugs	522	0	0		Set as default Backup Rename Copy Delete

Ilustración 35. Perfil de calidad para Java

Seleccionamos entonces el perfil por defecto “Sonar way”, obtendremos la siguiente vista:



Quality Profiles > Java > Sonar way >					
Coding rules Alerts Projects Permalinks Profile inheritance Changelog					
Name/Key	Repository	Severity	Activation	Status	Search
	Any	Any	Active	Any	
Download Bulk Change					
Sort by: Rule name					
Active/Severity	Name (expand / collapse)				
✓ Major	"enum" should no longer be used as a name				
✓ Major	"java.lang.Error" should not be extended				
✓ Major	Anonymous classes should not have too many lines				
✓ Minor	Array designators "[]" should be located after the type in method signatures				
✓ Minor	Array designators "[]" should be on the type, not the variable				
✓ Major	Assignments should not be made from within sub-expressions				
✓ Major	Avoid Array Loops				
✓ Major	Avoid Assert As Identifier				
✓ Major	Avoid Catching NPE				
✓ Major	Avoid commented-out lines of code				
✓ Major	Avoid Decimal Literals In Big Decimal Constructor				
✓ Major	Avoid Throwing Null Pointer Exception				
✓ Major	Avoid too complex class				
✓ Major	Avoid too complex method				
✓ Major	Big Integer Instantiation				
✓ Major	Boolean expressions should not be compared to true or false				
✓ Major	Boolean Instantiation				
✓ Critical	Broken Null Check				

Ilustración 36. Vista de modificación de un perfil de calidad

Lo primero de todo será seleccionar la pestaña “Projects” y elegir nuestro proyecto para que sean aplicados los umbrales que vamos a establecer. Podemos añadir todos los proyectos que queramos.



Quality Profiles > Java > Sonar way >					
Coding rules Alerts Projects Permalinks Profile inheritance Changelog					
Add project: <input type="text"/>					
Remove EGC EOC verification					
Remove All					

Ilustración 37. Pestaña Projects

Ahora vamos a crear una alerta para ello pulsaremos en la pestaña “Alerts” y elegiremos el aspecto del código del cual queremos establecer los umbrales.



Ilustración 38. Vista de gestión de alertas

Por ejemplo, uno de los que hemos elegido se expone a continuación: la complejidad media de función. Decidimos que cuando sobrepase 1.5 sea un aviso y que cuando sobrepase 3 se considere un error.



Ilustración 39. Creación de la alerta

Al volver a analizar la calidad nos mostraría un aviso diciéndonos que sobrepasamos dicho umbral tal y como se muestra a continuación:

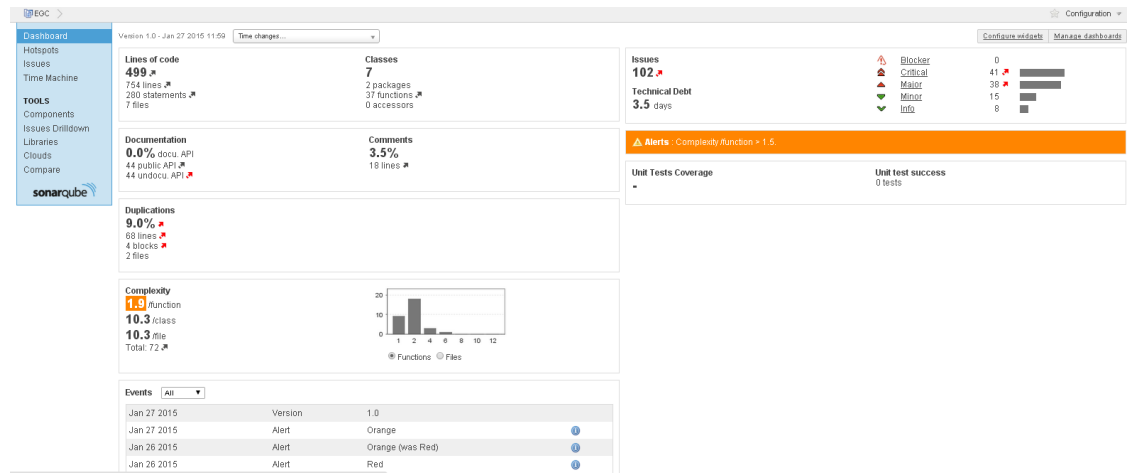


Ilustración 40. Mediciones mostrando la alerta creada

Procederíamos así para el resto de aspectos del código elegidos. Mostramos a continuación dichos aspectos y los umbrales establecidos para cada uno:

Aspectos	Umbral de aviso	Umbral de error
Complejidad total (COMPLEJIDAD)	Es mayor que 63 (*)	Es mayor que 105 (*)
Complejidad/función (COMPLEJIDAD)	Es mayor que 1.5	Es mayor que 3
Complejidad/clase (COMPLEJIDAD)	Es mayor que 9	Es mayor que 15
% de Comentarios (DOCUMENTACIÓN)	Es menor que 20%	Es menor que 15%
% de líneas duplicadas (DUPLICACIÓN)	Es mayor de 5 %	Es mayor de 20%
Incidencias bloqueantes (INCIDENCIAS)	Es mayor que 0	Es mayor que 1
Incidencias críticas (INCIDENCIAS)	Es mayor de 20	Es mayor que 30
Incidencias mayores (INCIDENCIAS)	Es mayor de 30	Es mayor de 80
Incidencias menores (INCIDENCIAS)	Es mayor de 20	Es mayor de 40

(*) Queremos 9 para el aviso y 15 para el error por cada clase, pero Sonar calcula esta métrica como la suma de todas las complejidades de cada clase, por ello se calculará como 7 por el límite. Debido a que 7 es el número de clases que tenemos.

Nota: Con respecto a la elección de dichos umbrales usamos nuestra propia experiencia y diversas fuentes de información citadas en el apartado bibliografía. Sin embargo, estos umbrales son específicos para nuestro proyecto, ya que los umbrales de la mayoría de los aspectos del código se deciden según el proyecto y no hay un estándar general. El único aspecto del código que tiene umbrales generalizados es la complejidad. Según la referencia citada en la bibliografía (Wikipedia) obtendremos menos riesgo cuando su valor se encuentre en torno a 10, por ello siempre elegimos siempre valores cercanos a 10 porque buscamos el menor riesgo posible. Los demás aspectos se rigen por opiniones generales y experiencia.

Definimos todos estos umbrales en el perfil de calidad mencionado anteriormente tal y como se muestra a continuación.

Alert is created. [\[hide\]](#)

[Coding rules](#) **[Alerts](#)** [Projects](#) [Permalinks](#) [Profile inheritance](#) [Changelog](#)

Create alert

Select a metric

Blocker issues	Value	is greater than	0	1	Update	Delete
Comments (%)	Value	is less than	20 %	15 %	Update	Delete
Complexity	Value	is greater than	9	15	Update	Delete
Complexity /class	Value	is greater than	9	15	Update	Delete
Complexity /function	Value	is greater than	1.5	3	Update	Delete
Critical issues	Value	is greater than	20	30	Update	Delete
Duplicated lines (%)	Value	is greater than	5 %	20 %	Update	Delete
Major issues	Value	is greater than	30	60	Update	Delete
Minor issues	Value	is greater than	20	40	Update	Delete

Ilustración 41. Umbrales definidos creados

Procedemos a ejecutar de nuevo la tarea de Jenkins para visualizar las alertas que recibimos según los umbrales establecidos. Como podemos observar Sonar nos muestra todas las alertas en un cuadro rojo, algo que dificulta distinguir con claridad si se trata de un error o un aviso. Sin embargo, nos marca el aspecto del código con su respectivo color. En nuestro caso tenemos en total 7 alertas. Hay 2 aspectos del código que superan el umbral de error: *Comments (%) <15* y *Critical issues >30*. Y también hay 5 aspectos del código que superan el umbral de aviso: *Complexity > 63*, *Complexity/function > 1.5*, *Complexity/class > 9*, *Duplicated lines(%) > 5* y *Major issues > 30*. A continuación en el caso práctico 1 y 2 tomaremos decisiones e intentaremos solucionar algunas de estas alertas.

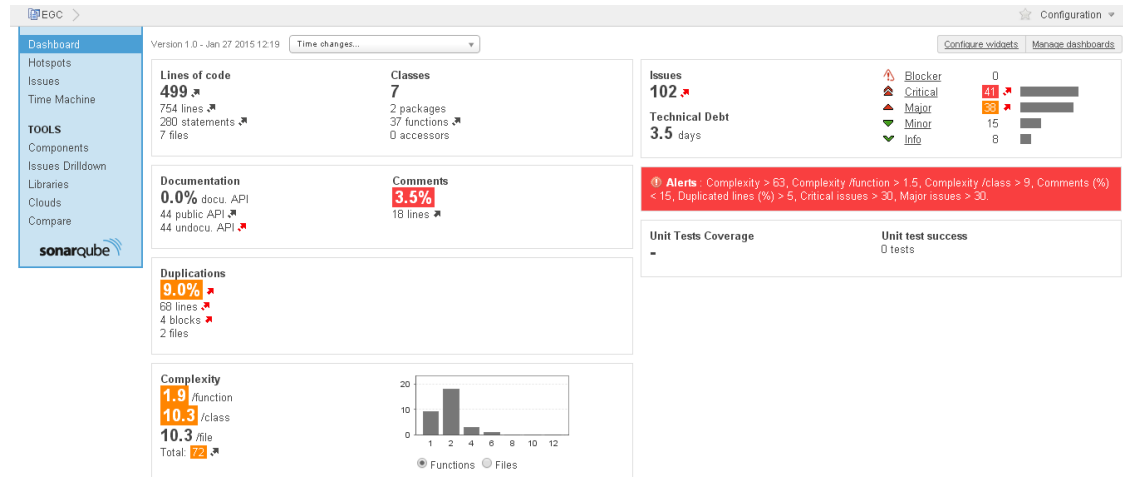


Ilustración 42. Alertas para todos los aspectos del código

6.3. Caso práctico 1.

Gracias al estudio realizado en el apartado anterior hemos detectado ciertas carencias en diferentes aspectos del código. Hemos decidido mejorar el siguiente aspecto: % de comentarios. Hemos actuado de la siguiente manera.

En primer lugar, el gestor de calidad va a realizar un “issue” en GitHub notificando al desarrollador y pidiendo que añada más comentarios. Tal y como explicamos en el apartado gestión del código el desarrollador estudiará la incidencia y aceptará o no el cambio. Mostramos a continuación la incidencia y la respuesta del desarrollador.

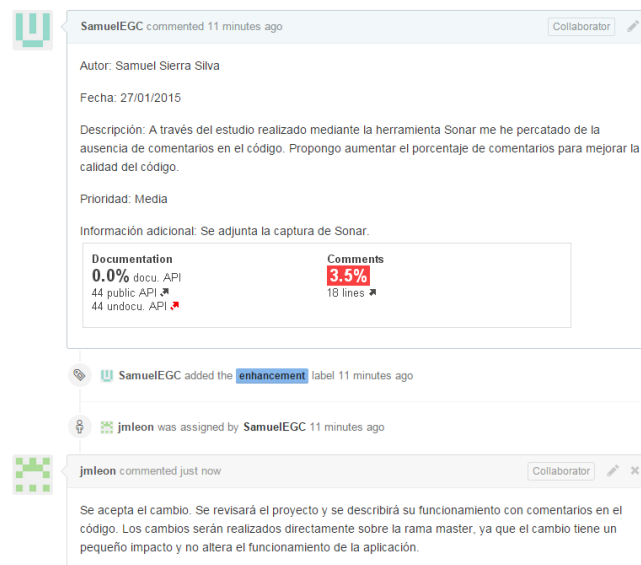


Ilustración 43. Incidencia creada para el caso práctico 1

Este sería el estado de la calidad antes de la modificación realizada por el desarrollador. Como podemos observar el umbral de error establecido para el % de comentarios es incumplido y esto debe de ser solventado.

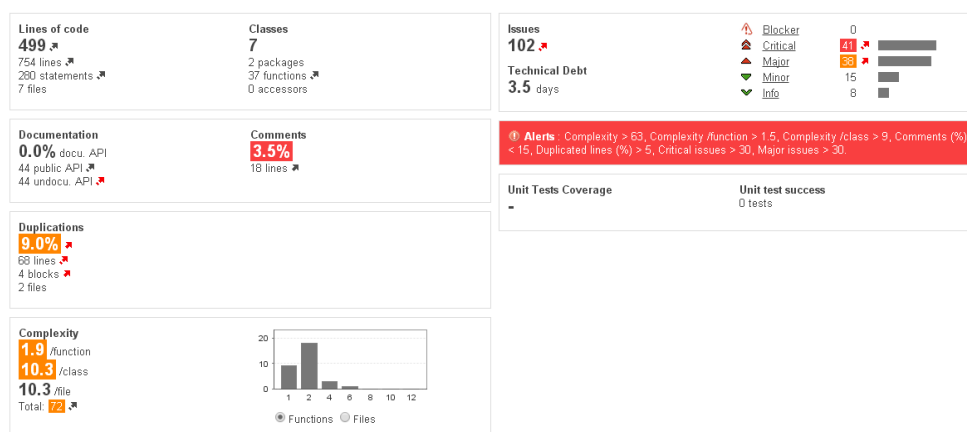


Ilustración 44. Umbral de porcentaje de comentarios incumplido

El desarrollador ha realizado los cambios y ha hecho push al repositorio. Por lo tanto, el gestor de calidad procede a ejecutar de nuevo el análisis de calidad del código con resultados satisfactorios, ya que se ha completado el objetivo propuesto y se cumple el umbral de error establecido para el aspecto propuesto. Curiosamente las incidencias informativas aumentaron y solo se añadieron comentarios. Parece ser por lo que hemos observado que la palabra “método” en los comentarios se interpreta como un comentario TODO porque termina en “todo”. En el siguiente caso práctico el desarrollador eliminará estas incidencias de menor importancia.

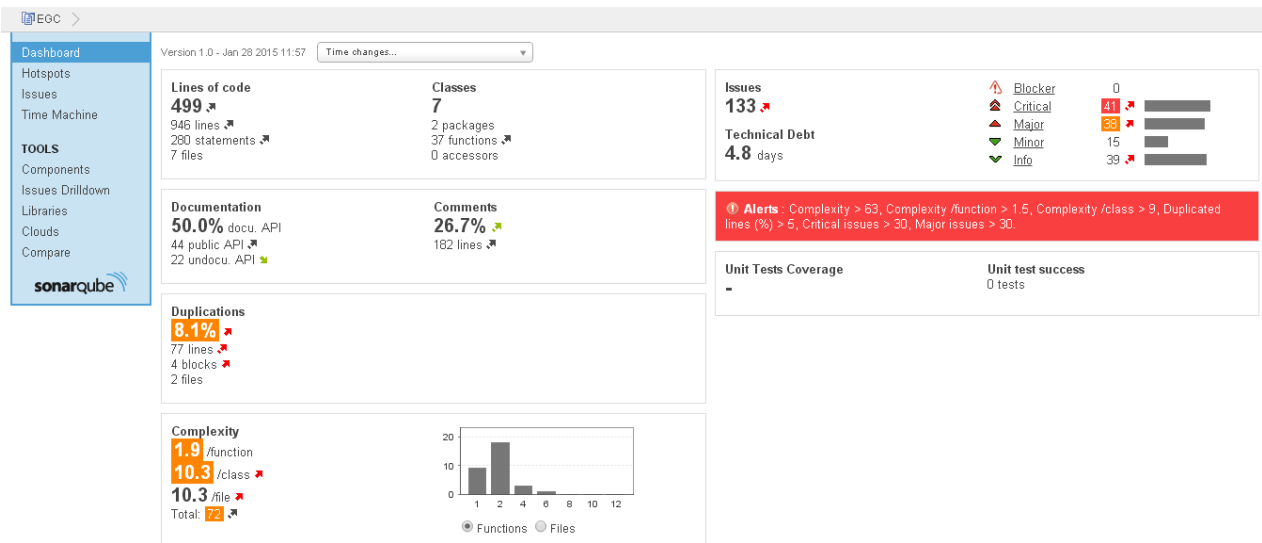


Ilustración 45. Resultado final, mejoras realizadas y aumento de la calidad del código

6.4. Caso práctico 2.

En este ejemplo, el gestor de calidad, revisando las incidencias del proyecto, identifica tres que podrían ser solucionadas con facilidad. Serían las siguientes:

- Dos incidencias mayores que nos dicen que hay 2 métodos que no se utilizan y están vacíos y que hay unos comentarios que deberían de borrarse.

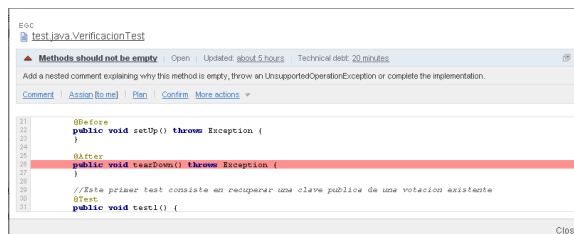


Ilustración 46. Incidencia mayor 1

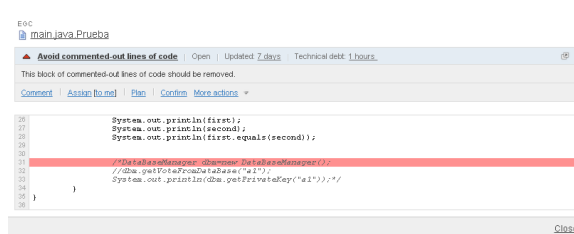


Ilustración 47. Incidencia mayor 2

- Una incidencia informativa que nos muestra un comentario TODO ya finalizado y que no se borró.

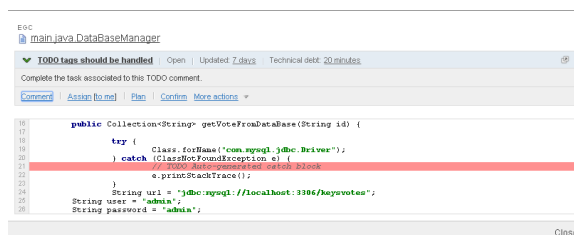



Ilustración 48. Incidencia informativa

Se procede igual que en el caso práctico anterior, el gestor de calidad informa con una “issue” al desarrollador y este valora el impacto del cambio. Volvemos a mostrar la incidencia y la correspondiente respuesta.



SamuelEGC commented a day ago

Collaborator

Autor: Samuel Sierra Silva


Fecha: 27/01/2015

Descripción: Mediante el análisis realizado con Sonar, se han detectado 2 incidencias mayores y 1 incidencia informativa. Propongo que se solucionen ya que considero que no presentan dificultad.

Prioridad: Media

Información adicional: Se adjunta las 2 incidencias mayores y la incidencia informativa en dicho orden.

EGC

 test.java.VerificacionTest

▲ **Methods should not be empty**

Open | Updated: about 5 hours | Technical debt: 20 minutes

Ⓟ

Add a nested comment explaining why this method is empty, throw an UnsupportedOperationException or complete the implementation.

Comment

Assign (to me)

Plan


Confirm

More actions

```
21 @Before
22 public void setUp() throws Exception {
23
24
25
26 @After
27 public void tearDown() throws Exception {
28
29
30 //Este primer test consiste en recuperar una clave publica de una votacion existente
31 @Test
32 public void test1() {
```

Close

EGC

 main.java.Prueba

▲ **Avoid commented-out lines of code**

Open | Updated: 7 days | Technical debt: 1 hour

Ⓟ

This block of commented-out lines of code should be removed.

Comment

Assign (to me)

Plan


Confirm

More actions

```
26
27 System.out.println(first);
28 System.out.println(second);
29 System.out.println(first.equals(second));
30
31 //DatabaseManager dba=new DataBaseManager();
32 //dba.getVoteFromDataBase("a1");
33 System.out.println(dba.getPrivateKey("a1"));*/
34
35
36 }
```

Close

EGC

 main.java.DataBaseManager

▼ **TODO tags should be handled**

Open | Updated: 7 days | Technical debt: 20 minutes

Ⓟ

Complete the task associated to this TODO comment.

Comment

Assign (to me)


Plan

Confirm


More actions

```
16 public Collection<String> getVoteFromDataBase(String id) {
17
18
19     try {
20         Class.forName("com.mysql.jdbc.Driver");
21     } catch (ClassNotFoundException e) {
22         // TODO Auto-generated catch block
23         e.printStackTrace();
24     }
25     String url = "jdbc:mysql://localhost:3306/keysvotes";
26     String user = "admin";
27     String password = "admin";
```


Close

 jmleon

was assigned by SamuelEGC a day ago

 SamuelEGC

added the **enhancement** label a day ago



jmleon commented a day ago

Collaborator

Se acepta el cambio. Se realizará la corrección sobre la rama master ya que se trata de cambios menores.

Ilustración 49. Incidencia creada para el caso práctico 2

El desarrollador ha seguido estudiando las incidencias aportadas por Sonar y decide continuar resolviéndolas porque no suponen ninguna dificultad. Exponemos solo estas 3 en este ejercicio práctico, pero dejamos constancia de que se han resuelto muchas más. Aquí está el comentario del desarrollador en la misma “issue” corroborando lo mencionado anteriormente.



Ilustración 50. Comentario en incidencia notificando de mejoras

Mostramos el estado de la calidad del código después de las modificaciones, como se comentó se han eliminado las incidencias informativas añadidas en el apartado anterior aparte de algunas más y las mencionadas en este caso práctico.

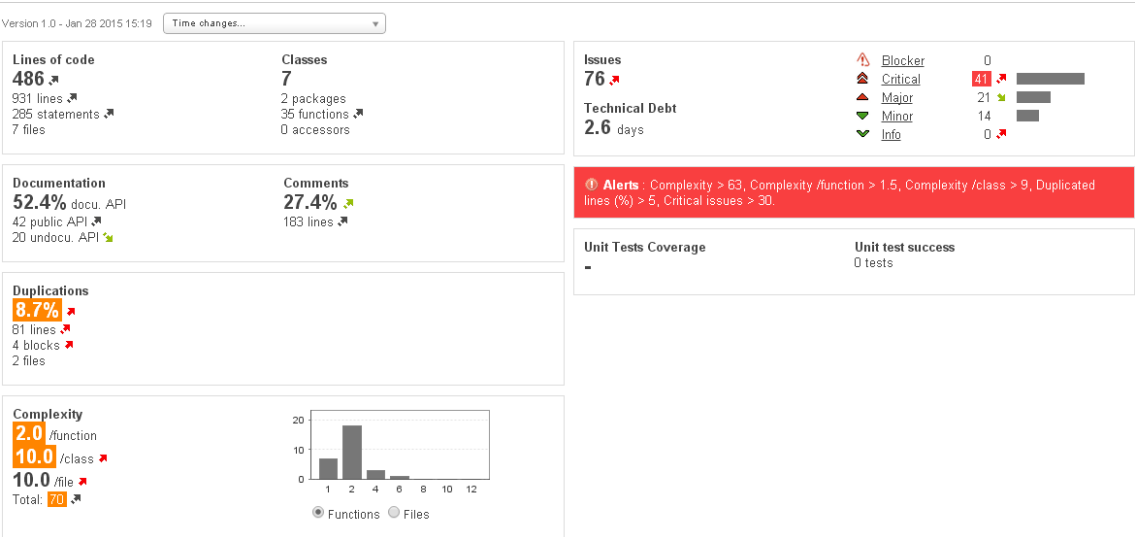


Ilustración 51. Resultado final reduciendo las incidencias

7. Gestión del Cambio, Incidencias y Depuración

7.1. Descripción

Como en casi todos los apartados, al inicio del proyecto, al ser un grupo tan reducido, la comunicación entre los integrantes del subsistema es bastante clara y frecuente, por tanto, todos estábamos al día de posibles incidencias y/o cambios. La actuación consecuente con esto, se derivaba en una reunión en la que todos los integrantes participaban, daban su valoración y se proponía una resolución al problema presentado.

Sin embargo, con la propia experiencia adquirida durante el desarrollo del proyecto, vimos la necesidad imperativa de utilizar alguna herramienta para ayudarnos a gestionar las incidencias y cambios.

Tras valorar las herramientas vistas en clase, decidimos utilizar GitHub, con su funcionalidad “issues”. Para ello, utilizaremos la siguiente plantilla cada vez que haya que reportar un error o solicitud de cambio:

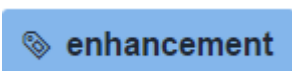
AUTOR	[La persona que encontró el error o que vio la necesidad de cambio]
FECHA	[dd/MM/yyyy]
DESCRIPCIÓN	[Explicación del problema lo más detallado y claro posible]
PASOS PARA REPRODUCIR EL PROBLEMA ENCONTRADO:	
1.	
2.	
3.	
ESPERADO	[Suceso esperado si no hubiera error]
RECIBIDO	[Suceso recibido (valor devuelto o excepción)]
VERSIÓN UTILIZADA	[Cualquier información relevante acerca de alguna versión de alguna herramienta]
PRIORIDAD	[Alta, Media o Baja]
INFORMACIÓN ADICIONAL	[Cualquier información adicional de interés]

Se abrirá una nueva “issue” en GitHub siguiendo dicha plantilla, y se informará a los demás miembros a través de los canales de comunicación establecidos. Esto ayudará a que quede constancia del cambio, y los demás miembros podrán valorar la necesidad de cambio o depuración, comentar posibles soluciones o aportar más información, que también quedará constatada con la ayuda de la herramienta de gestión de incidencias.

GitHub nos proporciona un amplio abanico de etiquetas para ayudar a clasificar las incidencias que reportamos, sin embargo, no las utilizamos todas. Según el objetivo de la “issue”, añadimos las siguientes etiquetas:



Utilizamos la etiqueta de bug cuando se trata de un fallo o error detectado.



Utilizamos la etiqueta enhancement cuando se trata de una mejora que se pretende hacer.

help wanted

Utilizamos la etiqueta help wanted generalmente en la comunicación con otros grupos, cuando se requiere la ayuda en concreto de un subsistema.

question

Utilizamos la etiqueta question para la comunicación con otros grupos, cuando tenemos alguna duda que necesitamos que sea respondida por otro subsistema.

conflict

Utilizábamos la etiqueta conflicto cuando había algún conflicto de código a la hora de hacer los commits. Actualmente no la utilizamos, ya que sólo un desarrollador se encarga del código.

En cuanto a la gestión de incidencias de manera global, se llegó a un consenso entre todos los grupos y se decidió utilizar la gestión de “issues” que proporciona GitHub, utilizando para ello el espacio común creado en la herramienta para la gestión del código fuente común.

Además, cabe destacar una buena práctica utilizada para la gestión de incidencias de manera global, en el caso de que fuera necesario informar a otro subsistema de algún error o necesidad de cambio.

Para ello, utilizaremos también GitHub aprovechando el repositorio común que estamos utilizando todos los subsistemas. Se hará de la siguiente forma: al crear la “issue”, en su nombre, pondremos una etiqueta en mayúsculas y entre corchetes referenciando el nombre del grupo con el que nos queremos comunicar.

Por ejemplo, si necesito informar de un error al grupo cabina de votación, generaría una “issue” con un nombre de la forma “[CABINA] Error al importar el archivo...”. De esta forma, cuando algún miembro de dicho grupo acceda a la herramienta, puede ver de un simple vistazo a qué “issues” debe prestarles mayor atención.

Sim embargo, esto sigue requiriendo que los miembros de otros grupos estén continuamente revisando si hay “issues” nuevas que deban revisar. Por ello, pensamos que se debería haber seguido una política de gestión de incidencias global en la que un representante de cada grupo publica su nombre de usuario de GitHub, de forma que se pueda utilizar el sistema de menciones de GitHub (@nomreUsuario) para notificar directamente a dicho subsistema. Al mencionar a un usuario, éste recibe un correo electrónico avisándole de dicha mención, luego sería una forma fácil de estar informado en todo momento.

Cuando otro grupo reporte una incidencia que nuestro subsistema debe resolver, se actuará de la misma forma descrita anteriormente: ante dicha incidencia, será el desarrollador de nuestra aplicación el encargado de valorar el impacto de dicho cambio, y tomar las decisiones oportunas para solucionarlo. Una vez tomada la decisión, se contestará a la “issue” recibida, informando de las acciones que se van a llevar a cabo para resolver la incidencia.

Dicho esto, explicaremos ahora qué proceso de depuración seguimos a la hora de arreglar un bug, siguiendo las recomendaciones explicadas en clase de teoría:

1. **Informar de la incidencia:** utilizando GitHub y su herramienta de issues, tal y como acabamos de explicar.

2. **Reproducir:** si se ha seguido correctamente la plantilla que explicamos un poco más arriba, debería ser sencillo reproducir el entorno en el que se observó el error o bug, por lo que deberíamos encontrarnos en un escenario en el que tenemos toda la información (aunque por experiencia, no siempre ha sido así). Como trabajamos dentro de una misma máquina virtual, el entorno suele ser el mismo, salvo para errores muy concretos reportados por otros subsistemas. No utilizamos para ello información de log.
3. **Diagnosticar:** una vez que se ha reproducido el error, o se ve conveniente diagnosticar cuál ha sido la causa del mismo, se comienza el proceso de diagnóstico. No seguimos ningún proceso metódico, sino que nos basamos en nuestra propia experiencia para tratar de determinar la causa. Para ello, generalmente, utilizamos la función “debug” de eclipse.
4. **Arreglar:** para ello, según lo explicado en el apartado de gestión del código fuente, se creará una rama o se trabajará sobre el tronco del proyecto dependiendo del impacto que puede tener dicho error. Una vez que se ha arreglado el problema, se crean una serie de test para comprobar que efectivamente se ha solucionado el error, y para que nuestra batería de test sea más completa. Cuando es estable, si se creó una rama para ello, se hará un merge con la rama principal, y si no, se hará un commit para subir los cambios al repositorio. También se subirán los cambios al repositorio común.
5. **Analizar:** lo más importante es revisar nuestra batería de test, ya que, si pasaba por alto el error, significa que no era tan completa como creíamos.

7.2. Caso práctico

Se aportarán tres casos prácticos: el primero explicando la gestión de incidencias dentro de nuestro grupo, el segundo explicando la gestión de incidencias global y el último mostrando un ejemplo del procedimiento de depuración de un error.

Para el primer caso práctico, nos basaremos en un caso real ocurrido durante el desarrollo de nuestro subsistema para poner un ejemplo de la comunicación interna dentro de nuestro grupo.

Una vez acabada la funcionalidad requerida por los demás subsistemas, nos encontramos con que uno de los grupos, concretamente Cabina de Votación que necesita nuestras claves para poder cifrar los votos, no puede utilizar el archivo .jar que proporcionamos, ya que están desarrollando su aplicación en Django.

Para solventar esta incidencia, se procede en primer lugar a documentar el error utilizando la herramienta “issues” de GitHub con la plantilla explicada anteriormente rellenando los campos con la información que se ha obtenido:

AUTOR	Andrés Pachón
FECHA	17/11/2014 (Primera sesión de integración)
DESCRIPCIÓN	El subsistema Cabina de Votación ha intentado utilizar el archivo .jar que les proporcionamos, pero tras estudiarlo han descubierto que no pueden importarlo debido a que su aplicación está siendo desarrollada en Django con Python, y presenta problemas de compatibilidad.
PASOS PARA REPRODUCIR EL PROBLEMA ENCONTRADO:	
1.	Crear un proyecto Django con Eclipse.
2.	Tratar de importar nuestro archivo .jar.
ESPERADO	Importación correcta para poder utilizar nuestros métodos.
RECIBIDO	No es posible importar el archivo.
VERSIÓN UTILIZADA	Las versiones de Django, Python o Eclipse son irrelevantes en este caso.
PRIORIDAD	Alta
INFORMACIÓN ADICIONAL	Ellos mismos nos proponen una solución, crear una API de nuestra aplicación a la que puedan llamar sin necesidad de importar nuestro código. Se estudiará la viabilidad de dicha propuesta.

Una vez completada la plantilla, se aportó dicha información utilizando la herramienta GitHub:

Error: Imposible importar nuestro .jar en Django. #2

Open diegarnie opened this issue 40 seconds ago · 0 comments

diegarnie commented 40 seconds ago

Autor: Andrés Pachón

Fecha: 17/11/2014 (Primera sesión de integración)

Descripción: El subsistema Cabina de Votación ha intentado utilizar el archivo .jar que les proporcionamos, pero tras estudiarlo han descubierto que no pueden importarlo debido a que su aplicación está siendo desarrollada en Django con Python, y presenta problemas de compatibilidad.

Pasos para reproducir el problema:

1. Crear un proyecto Django con Eclipse.
2. Tratar de importar nuestro archivo .jar.

Esperado: Importación correcta para poder utilizar nuestros métodos.

Recibido: No es posible importar el archivo.

Versión utilizada: Las versiones de Django, Python o Eclipse son irrelevantes en este caso.

Prioridad: Alta.

Información adicional: Ellos mismos nos proponen una solución, crear una API de nuestra aplicación a la que puedan llamar sin necesidad de importar nuestro código. Se estudiará la viabilidad de dicha propuesta.

Labels: None yet

Milestone: No milestone

Assignee: No one—assign yourself

Notifications: Unsubscribe

1 participant

Lock issue

Ilustración 52. Ejemplo plantilla en GitHub

Seguidamente, se avisa al desarrollador de dicha incidencia, y este decide qué hacer basándose en la complejidad del cambio a realizar y el tiempo y recursos de los que disponemos. En este caso concreto, el desarrollador decidió aceptar el cambio, respondiendo a la “issue” con la solución aportada.

Y finalmente, una vez resuelta la incidencia, se procedió a cerrar la “issue”:



Ilustración 53. Respuesta del gestor de código

Y vemos así el proceso que se sigue internamente para gestionar los cambios e incidencias.

De la misma forma, aportamos un ejemplo de qué mecanismo hemos seguido a lo largo del proyecto para resolver las incidencias que se nos plantean por parte de otros grupos.

La incidencia venía de parte del grupo “Cabina de Votación”, y se nos informaba de que la longitud de la clave RSA que utilizábamos era demasiado pequeña para cifrar con éxito los votos para poder guardarlos cifrados en la base de datos.

La incidencia era la siguiente:

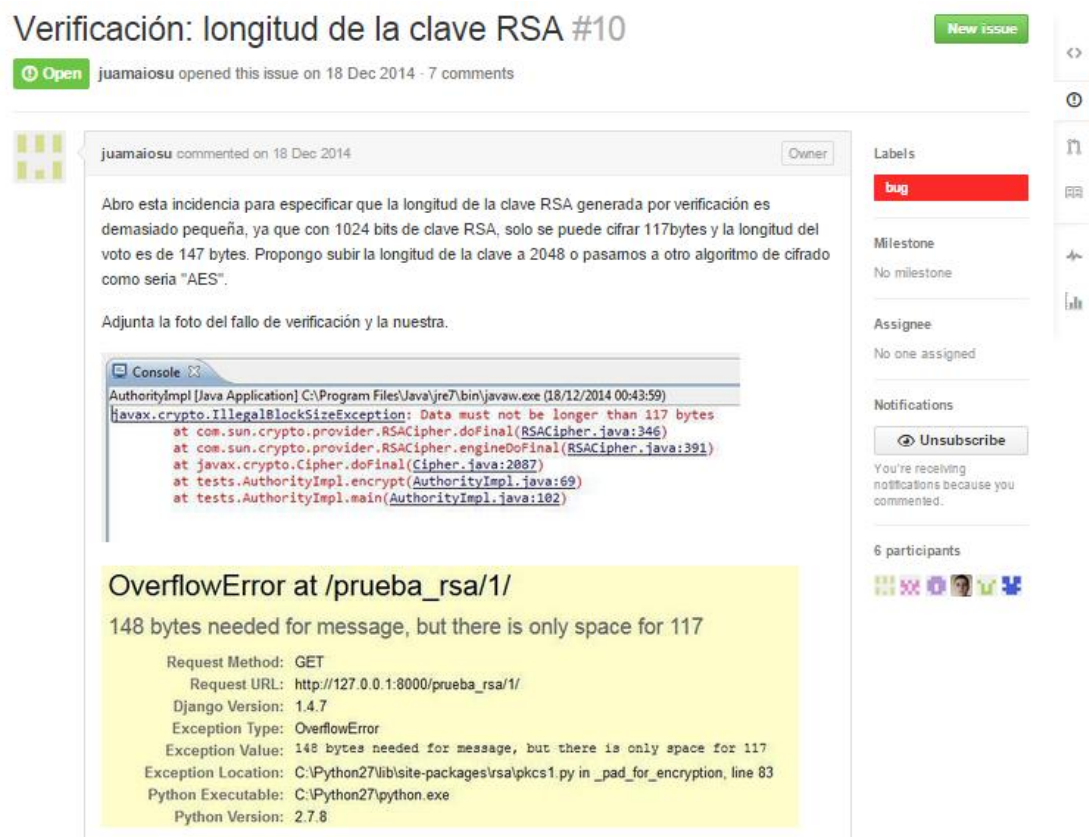


Ilustración 54. Solicitud de cambio de Cabina.

Una vez recibida la incidencia, el desarrollador valoró las dos opciones: aumentar la longitud de la clave a 2048 o cambiar el algoritmo de generación de claves de RSA a AES. Finalmente, tras estudiar el impacto de ambas opciones, y debido en parte a la tardanza con la que se informó de la incidencia, se decidió que se cambiaría la longitud de la clave.

Inmediatamente después de tomar la decisión, se respondió a la “issue”:

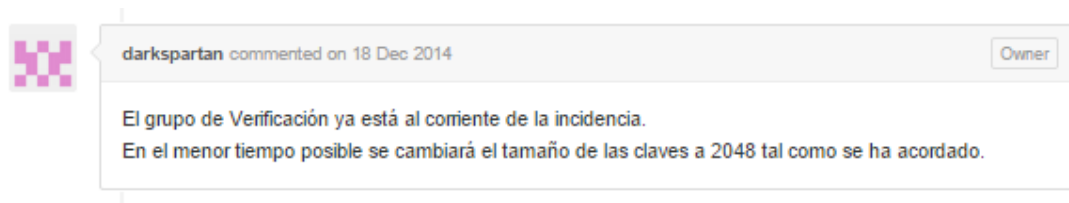


Ilustración 55. Informe de la decisión tomada.

Tras informar de la decisión tomada, el desarrollador comenzó a modificar el código de la aplicación. Cabe destacar que, según lo explicado en la sección de gestión de código fuente, se decidió no crear una nueva rama para resolver esta incidencia, ya que el impacto del cambio no era muy elevado.

Finalmente, una vez que se realizó correctamente el cambio, se subió a nuestro repositorio, y seguidamente al repositorio común y se procedió a informar de ello en la misma “issue” para que los demás subsistemas pudieran descargar de nuevo nuestro código y seguir integrando la aplicación. Como la incidencia ya estaba resuelta, procedimos a cerrarla:



Ilustración 56. Resolución de la incidencia.

Con este ejemplo hemos visto cómo gestiona nuestro subsistema las incidencias con otros grupos.

Por último, vamos a mostrar un ejemplo práctico del procedimiento seguido a la hora de depurar un error mostrando los pasos realizados en la corrección del bug descrito en el Issue#6 de nuestro repositorio local. Tal y como se ha explicado, el primer paso es informar de la incidencia a través de GitHub. De este modo, la persona que detecta el bug creará un Issue describiendo el error, como se muestra en la imagen:

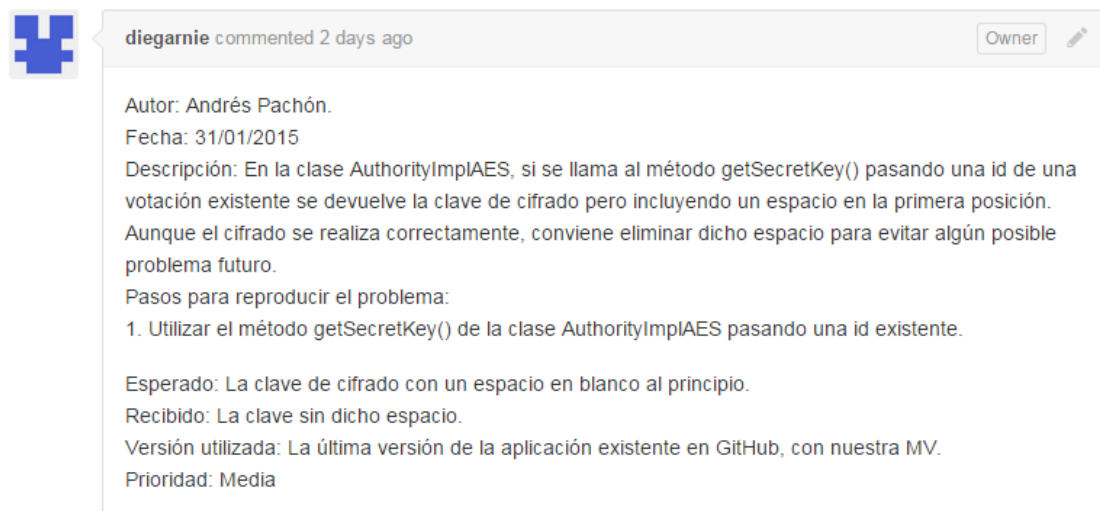


Ilustración 57. Descripción del error.

Una vez que se ha informado sobre el bug, el gestor de código responde indicando que se procede a reparar el bug:

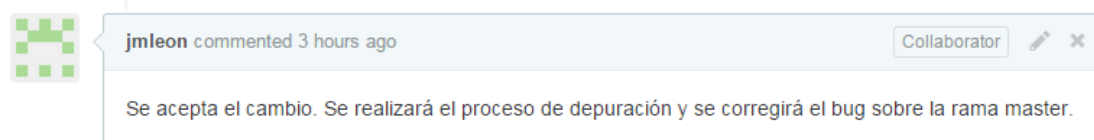


Ilustración 58. Respuesta del gestor de código.

Una vez que el gestor del código está informado, se pasa al segundo paso, el de reproducir el error. Como se aprecia en la descripción del Issue#6, el error consiste en que se incluye un espacio en blanco justo en la primera posición de la clave AES, tras obtenerla de la base de datos remota. Para reproducir el error, llamamos al método getSecretKey de la clase AuthorityImplAES y observamos si realmente existe un espacio en blanco en la primera posición del String que precede a la clave. Usando la herramienta debug de Eclipse obtenemos los siguientes resultados tras invocar al método mencionado:

Name	Value	
"authorityAES.ge...cretKey("1000")"	bmV2q1MarBMfNNvVq+AQZ1MhjRwaBHpV...	
"authorityAES.ge...000").charAt(0)"		
"authorityAES.ge...000").charAt(1)"	b	
Add new expression		

Ilustración 59. Reproduciendo el error.

La primera expresión de la imagen muestra el resultado de llamar al método `getSecretKey` pasando el id 1000. El valor de la variable parece que incluye un espacio en la primera posición, pero para salir de dudas vemos las dos siguientes expresiones que evalúan la primera y la segunda posición de la cadena, con lo que observamos que efectivamente se añade un espacio en blanco en la primera posición.

Una vez reproducido el problema pasamos a diagnosticar el mismo. En primer lugar vamos a comprobar el valor de la clave que está guardado en la base de datos para ver si contiene un espacio, lo que nos puede dar pistas sobre si el error se produce guardando o leyendo la clave.

Entramos en Hostinger y observamos que la tabla contiene el siguiente valor para la votación 1000:

	idvotation	secretKey
<input type="checkbox"/>	<input type="text" value="1000"/>	<input "="" type="text" value="bmV2q1MarBMfNNvVq+AQZlMhjRwaBHpVaMzAwBWMlB0="/>
<input type="checkbox"/>	<input type="text" value="999"/>	<input "="" type="text" value="TckferzJXYebjis25pVeEMIYu+HE6uRONkpfyzTys9Q="/>

Ilustración 60. Valor de la clave en la base de datos.

En la imagen observamos que la entrada que buscamos no contiene un espacio en el primer carácter. (En la imagen parece que la segunda entrada sí que contiene un espacio en la primera posición, aunque en realidad no es así, pues si se hace clic en ella se mostrará seleccionada para editar, y se apreciará que no contiene ningún espacio).

Tras ver que en la base de datos las claves se almacenan correctamente vamos a revisar como se lee la clave, ya que se hace de forma manual concatenando los caracteres a una cadena en blanco mediante un bucle for y es posible que en esta parte encontremos el bug. El código es el siguiente:

```
public String getSecretKey(String id){
    String fullPage = readPage(id,"AES");
    String res = "";

    //En el bucle se extrae el valor de la clave analizando el resultado de llamar a la
    // función readPage.
    for(int j = fullPage.indexOf("SecretKey:") + 10; fullPage.charAt(j)!='<' && j< fullPage.length() ;j++){
        res += fullPage.charAt(j);
    }

    return res;
}
```

Ilustración 61. Código para obtener la clave.

La función concatena a la cadena vacía (aquí no está el error pues no es un espacio) cada carácter que va encontrando tras la palabra "SecretKey:" que genera nuestro script Php intermedio, terminando de añadir caracteres cuando encuentra el símbolo '<'.

Aquí todo parece correcto, con lo que nos dirigimos al script Php y vamos a observar que nos encontramos justo después de la cadena "SecretKey:".

El fragmento del script que nos interesa es el siguiente:

```
mysql_select_db($db, $con);
$result = mysql_query("SELECT * FROM keyvotesAES where idvotation='".$idv.'"", $con);
echo "Secretkey: ".mysql_result($result, 0, "secretKey")."<br>";
```

Ilustración 62. Fragmento del script Php que consulta la base de datos remota.

En la línea marcada observamos como existe un espacio justo detrás de “SecretKey:” que será lo primero que se leerá cuando se llame al método de leer la clave. Por lo tanto debería ser suficiente con eliminar dicho espacio para corregir el error.

Continuamos con el siguiente paso: arreglar.

Para ello basta con eliminar dicho espacio. También se va a crear un nuevo test que detecte el error que había pasado desapercibido. Tras el cambio, ejecutamos los tests, incluyendo el nuevo y vemos que no se produce ningún error. Además, realizando la misma prueba que se realizó en el diagnóstico observamos que esta vez si se obtienen los resultados previstos:








Name	Value	
  "authorityAES.ge...cretKey("1000")"	bmV2q1MarBMfNNvVq+AQZ1MhjRwaBHpV...	
  "authorityAES.ge...000").charAt(0)"	b	
  "authorityAES.ge...000").charAt(1)"	m	
 Add new expression		

Ilustración 63. Resultado de la función getSecretKey tras la reparación del bug.

Damos por concluida la reparación y subimos los cambios.

Por último analizamos nuestra batería de test y estudiamos posibles mejoras. La batería de test del proyecto cuenta con unos test que verifican el funcionamiento básico de los métodos de Authority (crear claves, consultar claves, verificar, cifrar). Sin embargo, tras realizar la última iteración de código en la que se creaban métodos equivalentes a los existentes pero adaptados al algoritmo AES, no se incluyeron dichos test básicos, con lo que, tras realizar este análisis, se ha detectado la necesidad de crearlos y se ha procedido a su creación.

Finalmente cerramos el Issue#6 y damos por concluido el proceso de depuración.

8. Gestión de Liberaciones, Despliegue y Entregas

8.1. Descripción

En cuanto al despliegue, se va a indicar cómo se despliega la parte de nuestro subsistema que nos permite almacenar las claves en la nube y proporcionarlas a los entornos distintos a java que no puedan hacer uso de nuestro archivo jar. Este despliegue se realiza en Hostinger.es, donde se ofrece gratuitamente la posibilidad de crear una página web mediante PHP y MySQL.

Esta parte de nuestro subsistema está formada por un script SQL y varios scripts PHP. El script SQL es el que se usa para la creación de la base de datos, mientras que los scripts PHP son necesarios para proporcionar la API y para la comunicación con nuestros métodos java. En cuanto a la comunicación con nuestro código java, lo ideal sería que nuestros métodos accedieran directamente a la base de datos, pero para disponer de bases de datos remotas era necesario adquirir la versión de pago del hosting, por lo que se optó por usar unos scripts PHP intermedios a modo de API que accedieran a la base de datos.

El despliegue de estos archivos se realiza de forma manual. Para ello, obtenemos los archivos del repositorio de nuestro subsistema, incluidos en la siguiente ubicación: <https://github.com/diegarnie/egc/tree/master/HostingFiles>. Estos archivos deben permanecer siempre actualizados, y en caso de cambio, se deben actualizar en el repositorio antes de pasar a ser desplegados en el hosting.

Una vez obtenidos los archivos se accede a la interfaz web del servicio de hosting y se editan los ficheros PHP o se ejecuta el fichero SQL según corresponda.

8.2. Caso práctico

Como ejemplo práctico se va a realizar el despliegue del código PHP y del script SQL a nuestro servidor en la web. Para ello, obtenemos los archivos que queremos desplegar del repositorio de nuestro subsistema.

Una vez descargados los ficheros, ingresamos en Hostinger y se nos mostrará un panel de control en el que podemos acceder a diferentes características del servicio.

Para desplegar el código PHP, pulsaremos sobre la sección “Administrador de archivos” que aparece señalada en la siguiente imagen:

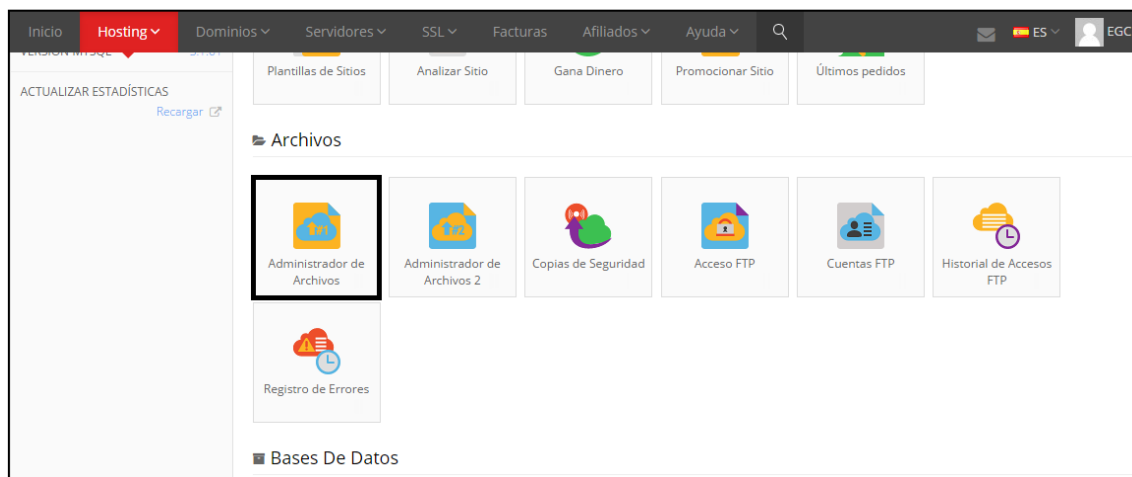


Ilustración 64. Sección “Administrador de Archivos”

Tras abrir el administrador de archivos, nos aparecerá una ventana como la siguiente:

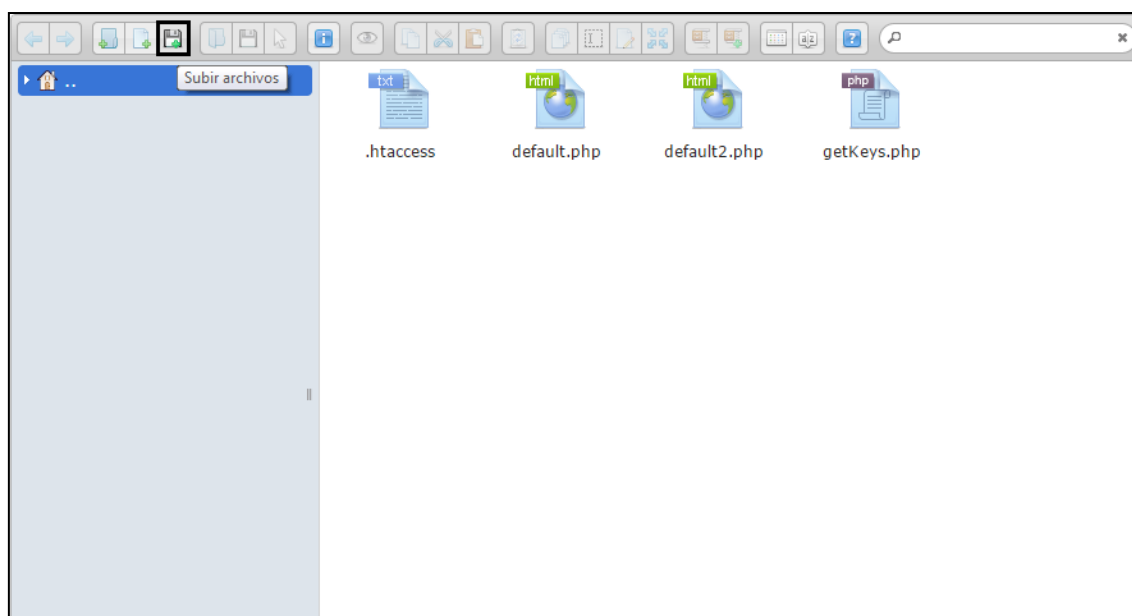


Ilustración 65. Archivos alojados en Hostinger

En la imagen aparecen los archivos PHP del proyecto junto a un archivo llamado .htaccess. Para desplegar los archivos nuevos del proyecto, pulsamos sobre el icono “Subir archivos” ubicado en la barra de herramientas y seleccionamos el archivo o los archivos a actualizar. Una vez seleccionados, pulsamos en “Abrir” y reemplazarán a los archivos antiguos, aunque la herramienta no muestra ningún mensaje de que han sido sustituidos, lo cual puede llevar a a confusión.

Por último vamos a ejecutar nuestro script de creación de la base de datos. Para ello, seleccionamos la sección “phpMyAdmin” como aparece en la imagen:

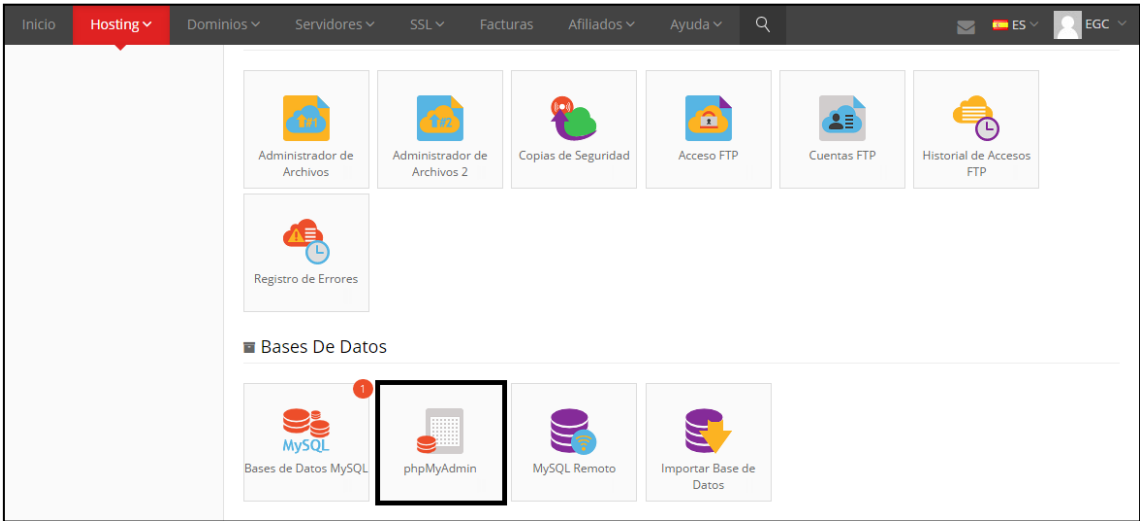


Ilustración 66. Sección “phpMyAdmin”

Seguidamente pulsamos en “Ingresar phpMyAdmin”:

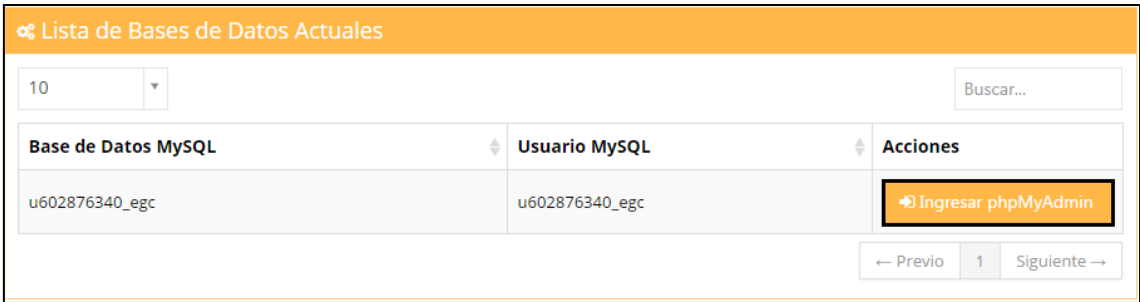


Ilustración 67. Ingresar en “phpMyAdmin”

Una vez hemos ingresado, nos aparecerá un panel desde el que podemos gestionar nuestras bases de datos. Para ejecutar el script, hacemos click en la pestaña SQL y pegamos el código del script necesario para crear la tabla en la base de datos tal como se muestra a continuación:

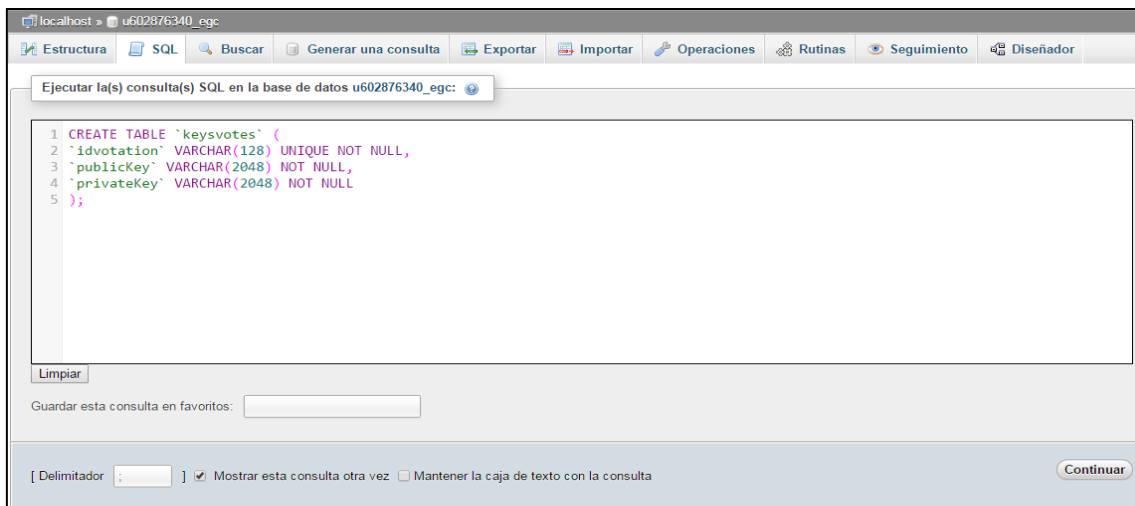


Ilustración 68. Ventana de ejecución de scripts.

Para ejecutar el script pulsamos en el botón “Continuar” y se nos creará la tabla necesaria para almacenar las claves de cifrado, mostrando el siguiente mensaje:

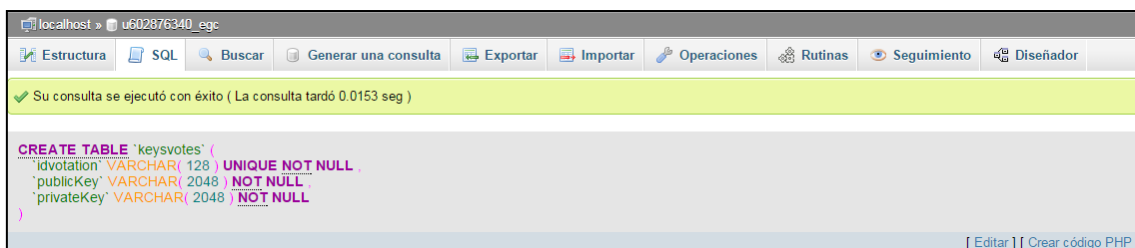


Ilustración 69. Resultado de la ejecución del script.

Con estos pasos, hemos conseguido desplegar nuestra base de datos en un servidor gratuito para que nuestra aplicación pueda acceder a él desde cualquier ordenador con conexión a internet.

9. Mapa de herramientas

9.1. Mapa de nuestro subsistema

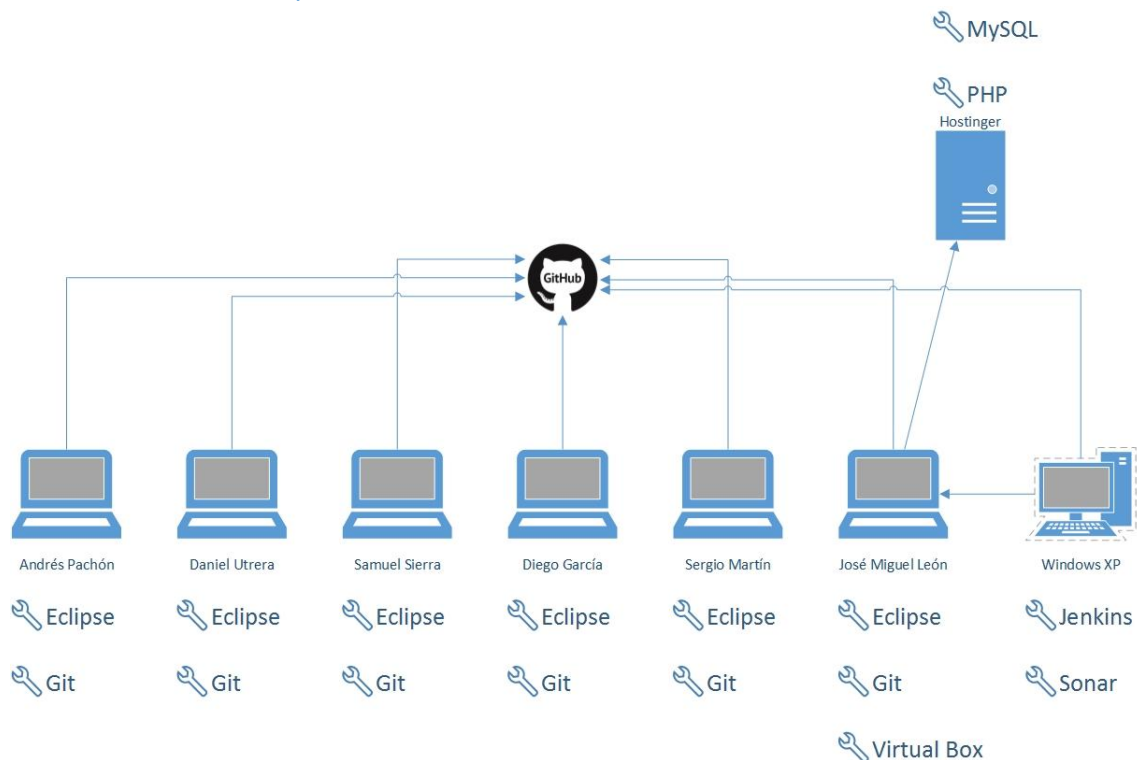


Ilustración 70. Mapa de herramientas Verificación

Eclipse: IDE de Java empleado para el desarrollo y mantenimiento del código correspondiente a nuestro subsistema así como para la generación automática del archivo “.jar”.

Git: Usamos Git como herramienta de control de versiones. En nuestro subsistema utilizamos un repositorio propio además del repositorio común. Además, utilizamos la funcionalidad “Issue” para gestionar las incidencias.

MySQL: Servidor de base de datos utilizado por todos los subsistemas. En nuestro caso disponemos de una tabla en un servidor remoto; en dicha tabla se almacenan los ids de las votaciones, así como las claves públicas y privadas correspondientes a cada votación.

PHP: En el hosting utilizado desplegamos dos ficheros en PHP que se encargan de realizar las llamadas a la base de datos ofrecida por nuestro subsistema.

VirtualBox: Aplicación de gestión de máquinas virtuales. Se monta una máquina virtual con Windows XP para la integración continua con Jenkins.

Jenkins: Aplicación de integración continua. Se ha instalado el plugin de “Sonar” para realizar estudios automáticos de la calidad del código. Además, con Jenkins ejecutamos los tests de funcionalidad y generamos el .jar y lo subimos a nuestro repositorio automáticamente.

Sonar: Herramienta que permite estudiar la calidad del código ofreciendo información de utilidad como la existencia de código duplicado, la complejidad ciclomática, etc.

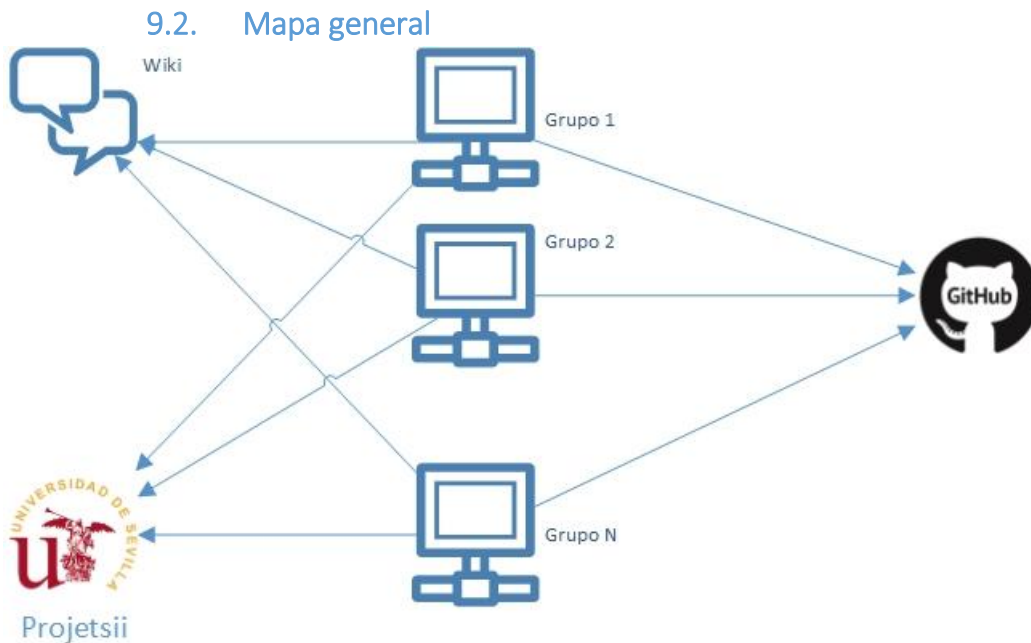


Ilustración 71. Mapa de herramientas general

Git: Entre todos los grupos se ha creado un repositorio común con una rama para que cada subsistema. De esta forma se puede tener acceso constante a las versiones últimas de cada subsistema. Además, utilizamos la funcionalidad “Issue” de Github para gestionar las incidencias.

Projetsii: Ante la necesidad de disponer de un método de comunicación general, se creó un foro en Projetsii. Finalmente la comunicación entre grupos se realizó mayormente a través de los issues de Github.

Wiki: La información de cómo se organizan los subsistemas y la integración entre estos está disponible en el espacio de trabajo de cada subsistema.

10. Conclusiones:

Durante el desarrollo de la práctica, hemos estudiado diferentes aspectos de la gestión de proyectos. En algunos de ellos, ya teníamos ciertos conocimientos, sin embargo, hay otros aspectos que nunca antes habíamos trabajado, al menos tan en profundidad.

Una vez acabado el proyecto, hemos identificado ciertos aspectos en los que se podría haber mejorado, o simplemente algunas conclusiones importantes a tener en cuenta:

- **Herramienta GitHub:** Una vez conocida la herramienta GitHub, hemos visto la imperiosa necesidad de aprender a manejarla y a utilizarla correctamente, ya que aporta mucha más utilidad y funcionalidad que Subversion. Ha ayudado a mantener una buena gestión del código fuente y a gestionar las incidencias. Por lo tanto, ha sido fundamental para el correcto desarrollo de este proyecto.
- **Gestión de Código:** Puede parecer que se pierde mucho tiempo al principio del proyecto definiendo esta política y poniendo esfuerzo en que todo el equipo de trabajo la entienda y la siga, pero este tiempo no se pierde: se invierte.
- **Necesidad de gestionar las comunicaciones:** Es importante que cualquier decisión que se tome o cualquier comunicación acerca de un aspecto del proyecto queden registrados para garantizar el no-repudio.
- **Jenkins:** Empezamos a utilizar esta herramienta en las fases finales del proyecto, y una vez que lo hemos utilizado, creemos que quizás deberíamos haber empezado a utilizarlo para hacer la integración de forma continua en fases más tempranas del proyecto.
- **Lenguaje y tecnologías:** Si bien la libertad a la hora de elegir la tecnología o el lenguaje con el que desarrollar nuestra aplicación ha ayudado en ocasiones a que surjan más conflictos, y por lo tanto, a aprender más de ellos, en otras ocasiones simplemente ha supuesto un impedimento y un retraso a la hora de avanzar en el proyecto. Dicho esto, y valorando ambas cosas, llegamos a la conclusión de que habría sido muy recomendable acordar una tecnología común o lo más compatible posible entre todos al comenzar el proyecto.
- **Avanzar a un ritmo parecido:** El trabajo de los distintos grupos ha avanzado de forma desigual: unos de manera constante y otros esperando al último momento. En nuestro caso concreto, hemos avanzado de forma rápida, teniendo la funcionalidad de nuestra aplicación disponible para los demás grupos en fases tempranas. Sin embargo, a pesar de tener todo prácticamente listo a mediados de cuatrimestre, por la falta de trabajo temprano de otros grupos, los problemas que podríamos haber ido solucionando paulatinamente los hemos tenido que solucionar todos en la última semana antes de la entrega. Entendemos que esto es lo que ocurre en los proyectos reales, pero se podría haber mejorado.
- **Valoración final:** Finalmente, y pensando un poco en todos los aspectos comentados, creemos que en general, el desarrollo de la asignatura, aunque un poco caótico, ha ayudado a enfrentarse a problemas nuevos y a que todos los aspectos de nuestra gestión de proyectos evolucionen hacia una metodología más profesional.

11. Bibliografía

- **Configuración del servidor y base de datos de Sonar.**
<http://enrikusblog.com/sonarqube-instalacion-y-configuracion/>
- **Interpretación de métricas para decidir umbrales.**
<http://www.javiergarzas.com/2013/09/metricas-sonar-1.html>
<http://www.javiergarzas.com/2013/09/metricas-sonar-2.html>
<http://www.javiergarzas.com/2013/09/metricas-sonar-3.html>
http://es.wikipedia.org/wiki/Complejidad_ciclomática

12. Anexos

12.1. Resumen de las Mejoras Realizadas

El contenido de este punto resume las mejoras que se han realizado para la segunda entrega. Según entendemos en la Wiki de la asignatura, este resumen debería estar en un documento propio, pero al no haber ningún tipo de documento en la entrega de Opera en el que incluirlo, optamos por añadirlo como anexo en el presente documento.

A continuación se resumen los aspectos que se han mejorado tras la revisión del profesor de la primera entrega. Estas mejoras aparecerán en el documento en color verde, con lo que serán fácilmente identificables.

Los cambios realizados, ordenados según el documento original, son los siguientes:

- Se ha realizado una descripción de roles y se ha identificado cada miembro del grupo con un rol.
- Se han añadido los apartados de Resumen, Introducción y Subsistema de Verificación y sus funcionalidades en los que se da una breve introducción acerca de la realización del proyecto y del cometido del documento.
- En el apartado “Gestión de código fuente” se ha explicado en más detalle la política que seguimos para gestionar el código fuente, centrándonos sobre todo en los roles seguidos en dicha política. También se ha explicado con más detalle la gestión de código global con otros grupos. Además se ha sustituido el ejemplo práctico de la entrega anterior que no estaba relacionado con nuestro proyecto por una iteración real de nuestro código en la que añadimos una nueva funcionalidad.
- En el apartado “Integración continua entre subsistemas” se ha descrito mejor la integración con el subsistema “Cabina de votación”, describiendo la Api que le proporcionamos.
- En el apartado “Integración continua en nuestro subsistema” se ha descrito la generación automática del archivo Jar y se incluye un ejemplo práctico en el que obtenemos dicho archivo que se ha generado gracias a Jenkins.
- En el apartado de construcción, se ha añadido como importar un jar en aquellos proyectos en los cuáles se hace uso de maven, como por ejemplo en el framework Spring. Este apartado se ha tratado como un ejemplo práctico más y se han añadido capturas de pantalla en las cuales se puede ver el proceso a seguir hasta importar correctamente un jar.
- Apartado “Integración de todos los subsistemas”: Se ha elaborado una ilustración y la correspondiente explicación de las herramientas usadas para el despliegue y los elementos que se integran con cada una de éstas.
- Se ha añadido el apartado gestión de la calidad. En primer lugar, se han explicado los procesos de configuración e instalación de las dos herramientas que se han utilizado para este apartado: Sonar y el “plugin” de Sonar para Jenkins. Se ha realizado un estudio de la calidad de nuestro código. A partir de este estudio, se han determinado

umbrales para diferentes aspectos del código determinando un perfil de calidad. Se ha realizado una interpretación de las métricas resultantes y se ha decidido cuáles son los aspectos del código que se deben mejorar. Por último, se ha iterado solucionando incidencias y umbrales incumplidos obteniendo como resultado una mayor calidad en nuestro código.

- En el apartado “Gestión del cambio, incidencias y depuración” se ha explicado con más detalle la política seguida para gestionar las incidencias, tanto dentro de nuestro grupo como con otros grupos. También se ha añadido una explicación acerca de las etiquetas que usamos para el reporte de incidencias, de igual modo, en nuestro grupo y con otros grupos. Se ha añadido una sección en la que explicamos el proceso de depuración que seguimos inspirado en lo explicado en clases de teoría. Por último, se ha mejorado el ejemplo de gestión de incidencias interno y se ha añadido otro ejemplo en el que explicamos cómo gestionamos las incidencias reportadas por otros grupos. Finalmente realizamos un último caso práctico en el que se depura un error de código siguiendo el procedimiento definió.
- Se ha añadido el apartado “Gestión de liberaciones, despliegue y Entregas”, en el que se describe como se despliega la parte del sistema que está alojada en el hosting gratuito. Además se añade un ejemplo práctico en el que se despliegan varios archivos en el servidor.
- Mapa de herramientas: Se ha cambiado el mapa de herramientas general por uno propio del grupo junto a uno de las herramientas usadas en común. Además, para cada mapa se ha elaborado una ilustración nueva más clara que la anterior.