

# Documento del proyecto

---

Grupo 10: Verificación

EVOLUCIÓN Y GESTIÓN DE LA  
CONFIGURACIÓN

Integrantes:

GARCÍA NIETO, DIEGO  
LEÓN RIEGO, JOSÉ MIGUEL  
MARTÍN MAROTO, SERGIO  
PACHÓN JIMÉNEZ, ANDRÉS  
SIERRA SILVA, SAMUEL  
UTRERA JAÉN, DANIEL

## Contenido

1.	Gestión de Código Fuente.....	3
1.1	Caso práctico .....	3
2.	Gestión de Construcción e Integración Continua. ....	6
2.1	Integración Continua entre subsistemas .....	6
2.2	Integración continua en nuestro subsistema.....	8
2.3	Caso práctico .....	8
2.4	Construcción.....	11
2.5	Caso práctico .....	11
3.	Gestión del Cambio, Incidencias y Depuración .....	15
3.1	Caso práctico .....	15
4	Mapa de herramientas.....	17

## 1. Gestión de Código Fuente.

- Nuestro proyecto es pequeño en cuanto a diversidad de la funcionalidad, por lo tanto tampoco será necesario crear numerosas ramas durante la realización del proyecto.
- Se generarán ramas para cualquier actividad relacionada con usar librerías desconocidas, intentar mejorar la eficiencia de un método, en definitiva todo lo relacionado con la investigación y mejora de los métodos.
- 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.
- En cuanto a los derechos, todos tendremos derecho de lectura y escritura sobre todos los archivos, por lo que no habrá roles como tal, se supone que en un proyecto tan pequeño estos roles provocarían demora en el trabajo no teniendo sentido la creación de roles.

## ES MUY CORTO Y POCO CONVINCENTE que se atribuya a lo pequeño del proyecto

### 1.1 Caso práctico

Tenemos un archivo python para un proyecto, y hemos detectado que no se contempla correctamente la división por 0. Por ello, vamos a clonar el repositorio, arreglar el método y subirlo de nuevo al repositorio.

### ¿qué tiene que ver con vuestro sistema?

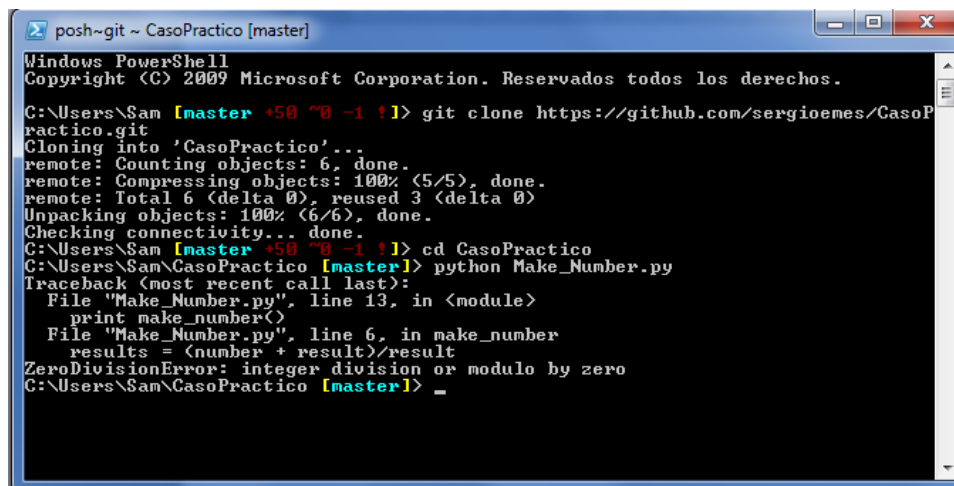
El contenido del fichero Make\_Number.py inicialmente es el siguiente:

```
def make_number():
    result = 0
    numbers_list = [1,2,3]
    for number in numbers_list:
        results = (number + result)/result

    return result

if __name__ == "__main__":
    print make_number()
```

Comenzamos clonando el repositorio (creado previamente) cuya dirección es la siguiente: <https://github.com/sergioemes/CasoPractico.git>



```
posh~git ~ CasoPractico [master]
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\Sam [master +50 ^0 -1 !] > git clone https://github.com/sergioemes/CasoPractico.git
Cloning into 'CasoPractico'...
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 6 (delta 0), reused 3 (delta 0)
Unpacking objects: 100% (6/6), done.
Checking connectivity... done.
C:\Users\Sam [master +50 ^0 -1 !] > cd CasoPractico
C:\Users\Sam\CasoPractico [master] > python Make_Number.py
Traceback (most recent call last):
  File "Make_Number.py", line 13, in <module>
    print make_number()
  File "Make_Number.py", line 6, in make_number
    results = (number + result)/result
ZeroDivisionError: integer division or modulo by zero
C:\Users\Sam\CasoPractico [master] > _
```

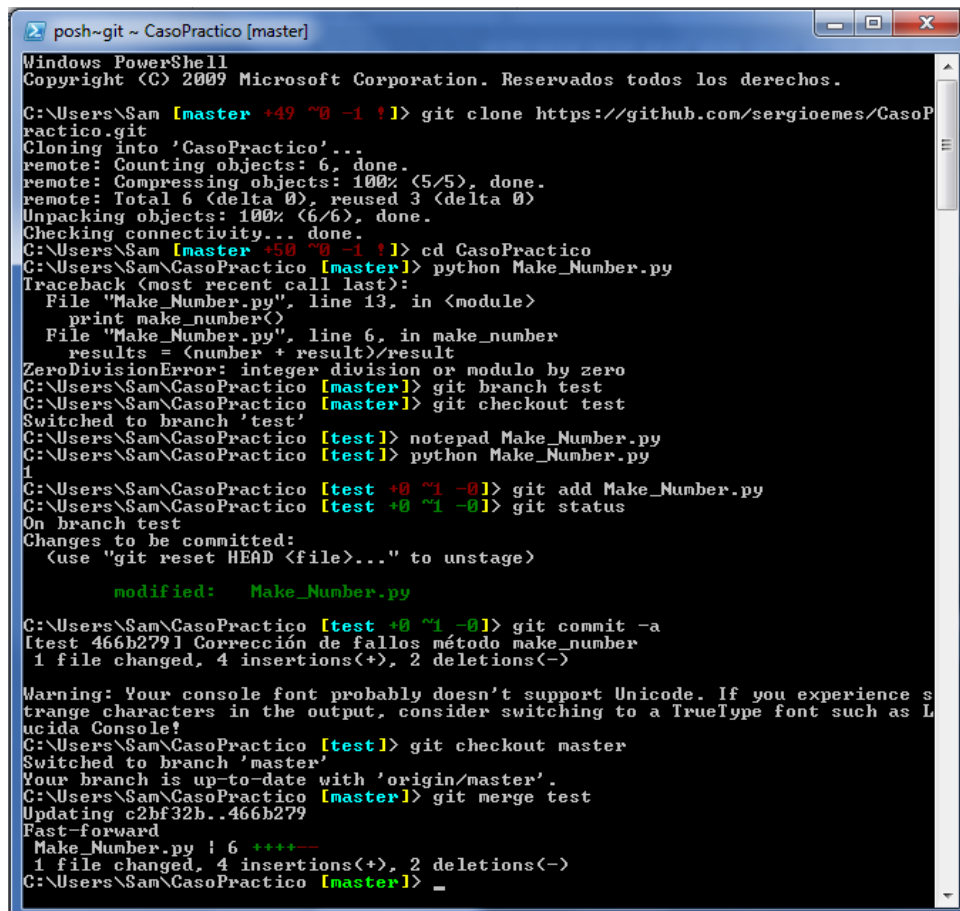
Ilustración 1. Probando el archivo

Ejecutamos el archivo y comprobamos que lanza un error. Pasamos a visualizar el archivo .py y detectamos tres errores:

- El resultado de los cálculos se guarda en una variable llamada “results” que luego no se devuelve por error.
- En lugar de sumar esta variable “results” al nuevo número de la lista en cada iteración, se suma la variable “result”.
- Observamos un nuevo error a través de la información mostrada por la consola que debe ser solventado. Como ya se comentó se trata de una división por cero. Revisando el código nos percatamos que no se hace incremento del divisor ANTES de dividir, ya que este divisor se guarda en una variable “result” inicializado a 0.

Para corregir estos errores hacemos lo siguiente:

- Creamos una rama llamada test y nos pasamos a dicha rama. Editamos el archivo.
- Ejecutamos el archivo para comprobar si los cambios realizados han corregido los errores.
- Añadimos el archivo y hacemos commit.
- Unimos la rama test a la rama master.



```
posh-git ~ CasoPractico [master]
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\Sam [master +49 ^0 -1 ^1]> git clone https://github.com/sergioemes/CasoPractico.git
Cloning into 'CasoPractico'...
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 6 (delta 0), reused 3 (delta 0)
Unpacking objects: 100% (6/6), done.
Checking connectivity... done.
C:\Users\Sam [master +50 ^0 -1 ^1]> cd CasoPractico
C:\Users\Sam\CasoPractico [master]> python Make_Number.py
Traceback (most recent call last):
  File "Make_Number.py", line 13, in <module>
    print make_number()
  File "Make_Number.py", line 6, in make_number
    results = (number + result)/result
ZeroDivisionError: integer division or modulo by zero
C:\Users\Sam\CasoPractico [master]> git branch test
C:\Users\Sam\CasoPractico [master]> git checkout test
Switched to branch 'test'
C:\Users\Sam\CasoPractico [test]> notepad Make_Number.py
C:\Users\Sam\CasoPractico [test]> python Make_Number.py
1
C:\Users\Sam\CasoPractico [test +0 ^1 -0]> git add Make_Number.py
C:\Users\Sam\CasoPractico [test +0 ^1 -0]> git status
On branch test
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   Make_Number.py

C:\Users\Sam\CasoPractico [test +0 ^1 -0]> git commit -a
[test 466b279] Corrección de fallos método make_number
 1 file changed, 4 insertions(+), 2 deletions(-)

Warning: Your console font probably doesn't support Unicode. If you experience strange characters in the output, consider switching to a TrueType font such as Lucida Console!
C:\Users\Sam\CasoPractico [test]> git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
C:\Users\Sam\CasoPractico [master]> git merge test
Updating c2bf32b..466b279
Fast-forward
 Make_Number.py | 6 +++--
 1 file changed, 4 insertions(+), 2 deletions(-)
C:\Users\Sam\CasoPractico [master]> _
```

Ilustración 2. Comandos realizados

## 2. Gestión de Construcción e Integración Continua.

### 2.1 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. También se les proporcionará un .jar para el conector MySQL, para el caso en el que dicho subsistema no lo tuviera importado ya.

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: cuando necesiten cifrar un voto, llamarán a nuestro método "getPublicKey" con la "id" de la votación en la que se encuentre (un String) y cifrarán usando dicha clave que les proporcionamos.
- 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”

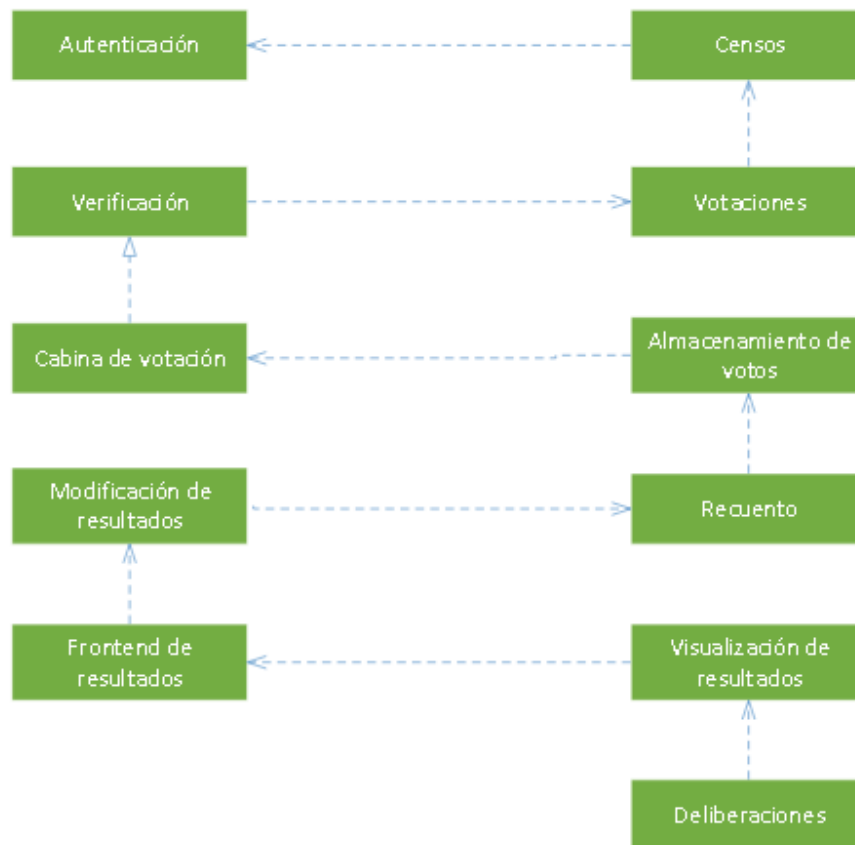



Ilustración 3. Integración modelo sandwich

## 2.2 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: una vez al día, 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:



The screenshot shows the Jenkins 'Test Result' page. At the top, it displays '0 failures (±0)' with a blue progress bar. To the right, it indicates '5 tests (±0)', 'Took 6.1 sec.', and a link to 'add description'. Below this is the 'All Tests' section, which contains a table with the following data:

Package	Duration	Fail	(diff) Skip	(diff) Pass	(diff) Total	(diff)
<a href="#">test.java</a>	6.1 sec	0	0	5	5	

Ilustración 4. Resultados de la ejecución de tests

De este modo, cualquier cambio que impida que los test sean ejecutados correctamente, podrá descubrirse en menos de 24 horas de subir el código al repositorio, ya que Jenkins nos informaría de la situación.

## 2.3 Caso práctico

Vamos a realizar un caso práctico que servirá de ejemplo del uso de Jenkins en nuestro subsistema. 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:



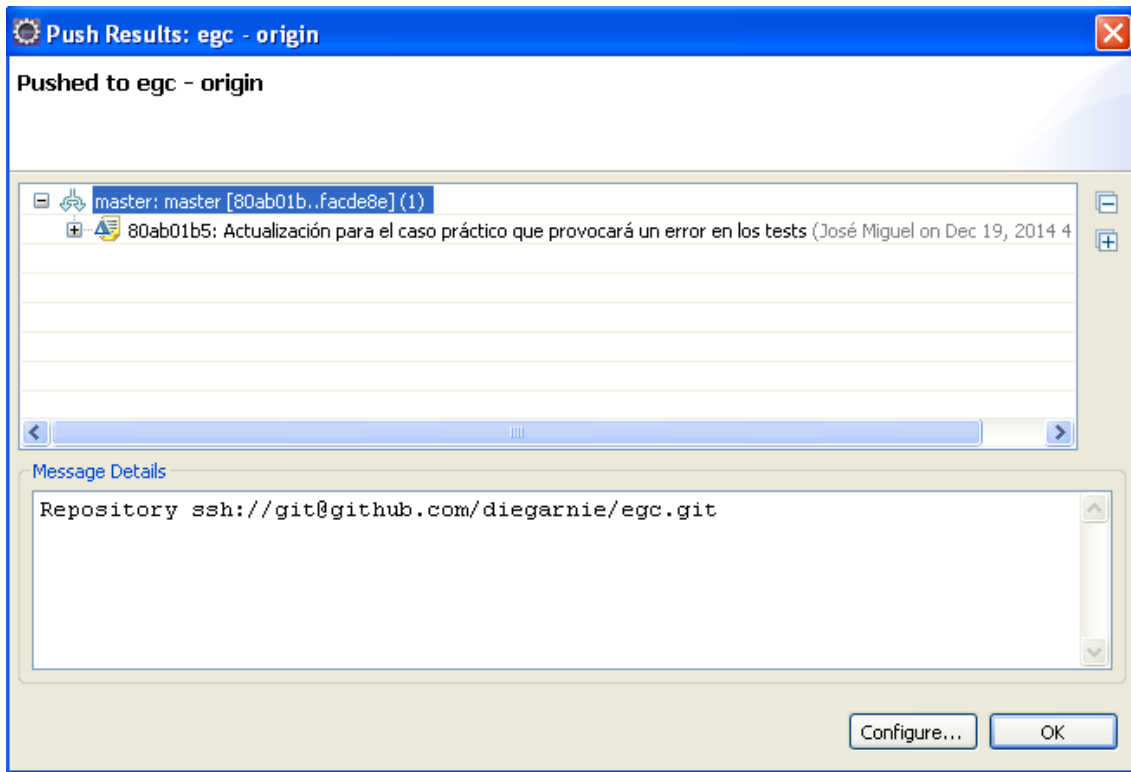


Ilustración 5. Fallo subido para provocar un error

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:

**Test Result**

1 failures (+1)

5 tests (#0)  
Took 5.3 sec.  
[add description](#)

**All Failed Tests**

Test Name	Duration	Age
<a href="#">test.java.VerificacionTest.test1</a>	0.75 sec	1

**All Tests**

Package	Duration	Fail	(diff)	Skip	(diff)	Pass	(diff)	Total	(diff)
<a href="#">test.java</a>	5.3 sec	1	+1	0	4	-1	5		

Ilustración 6. Error en la ejecución de los tests

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

### Regression

test.java.VerificacionTest.test1

Failing for the past 1 build (Since #12)  
[Took 0.75 sec.](#)  
[add description](#)

**Error Message**

```
expected:<[]MIIBIjANBgkqhkiG9wOB...> but was:<[ ]MIIBIjANBgkqhkiG9wOB...>
```

**Stacktrace**

```
junit.framework.ComparisonFailure: expected:<[]MIIBIjANBgkqhkiG9wOB...> but was:<[ ]MIIBIjANBgkqhkiG9wOB...>
    at junit.framework.Assert.assertEquals(Assert.java:100)
    at junit.framework.Assert.assertEquals(Assert.java:107)
    at test.java.VerificacionTest.test1(VerificacionTest.java:31)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:601)
    at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:47)
    at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
    at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:44)
    at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
    at org.junit.internal.runners.statements.RunBefores.evaluate(RunBefores.java:26)
    at org.junit.internal.runners.statements.RunAfters.evaluate(RunAfters.java:27)
    at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:271)
    at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:70)
    at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:50)
    at org.junit.runners.ParentRunner$3.run(ParentRunner.java:238)
    at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:63)
    at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:236)
```

Ilustración 7. 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 (40)  
[Took 7 sec.](#)  
[add description](#)

**All Tests**

Package	Duration	Fail	(diff)	Skip	(diff)	Pass	(diff)	Total	(diff)
<a href="#">test.java</a>	7 sec	0	-1	0		5	+1	5	

Ilustración 8. Error solventado

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

los ejercicios propuestos no tienen relación con vuestro proyecto

## 2.4 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.

## 2.5 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.

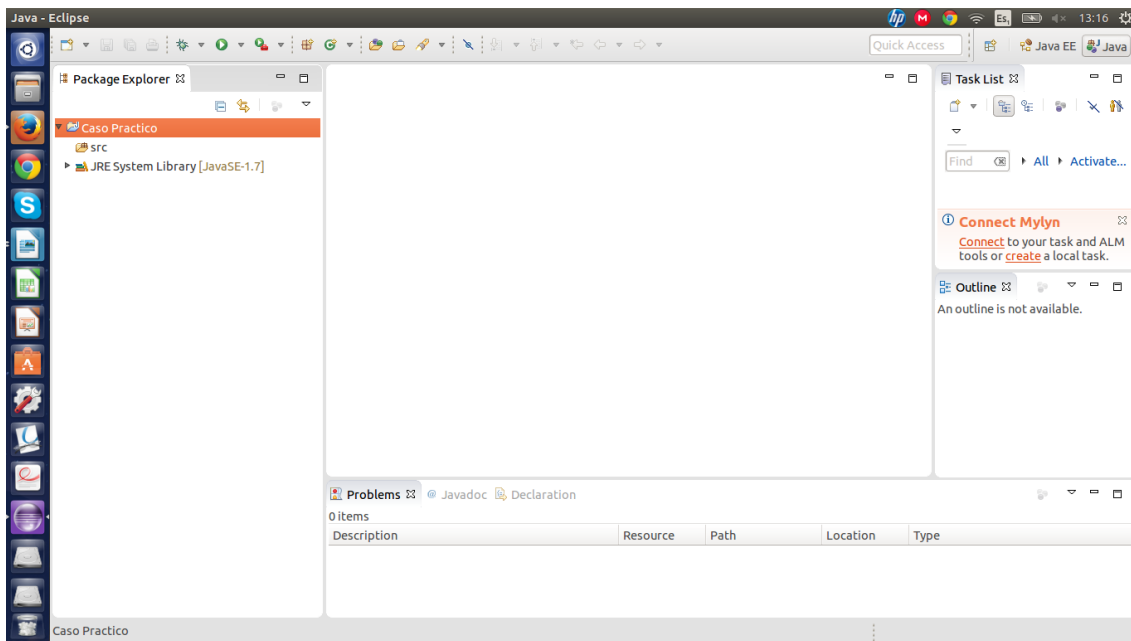


Ilustración 9. Vista general proyecto de prueba

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”.

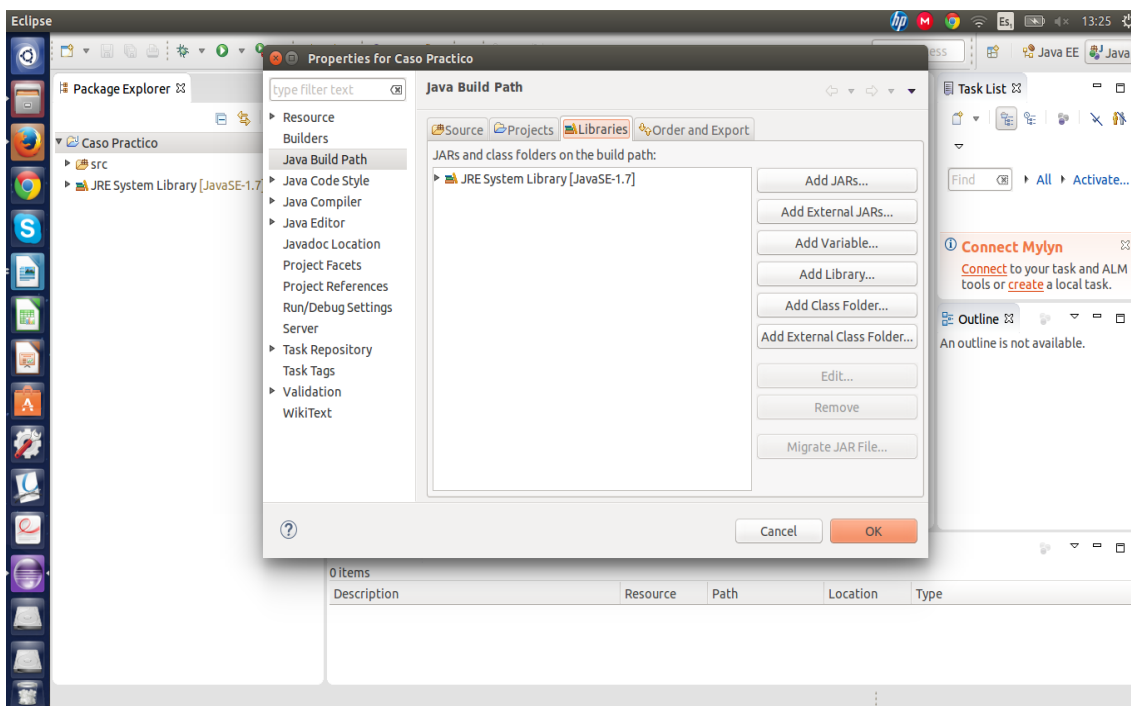


Ilustración 10. Propiedades del proyecto

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.

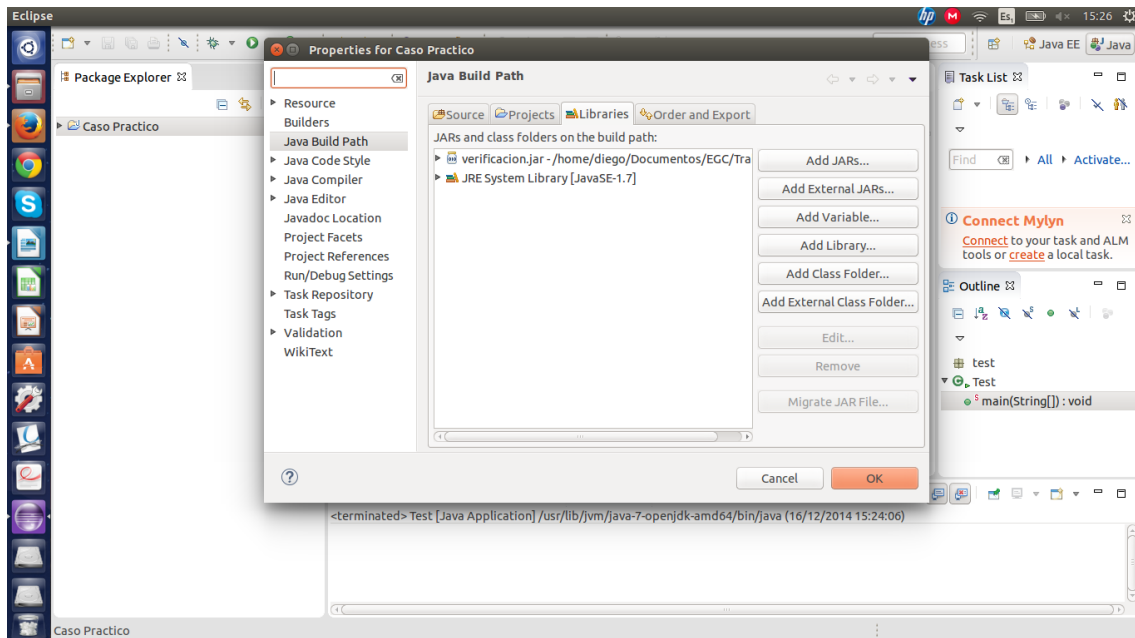


Ilustración 11. Añadir .Jar

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:

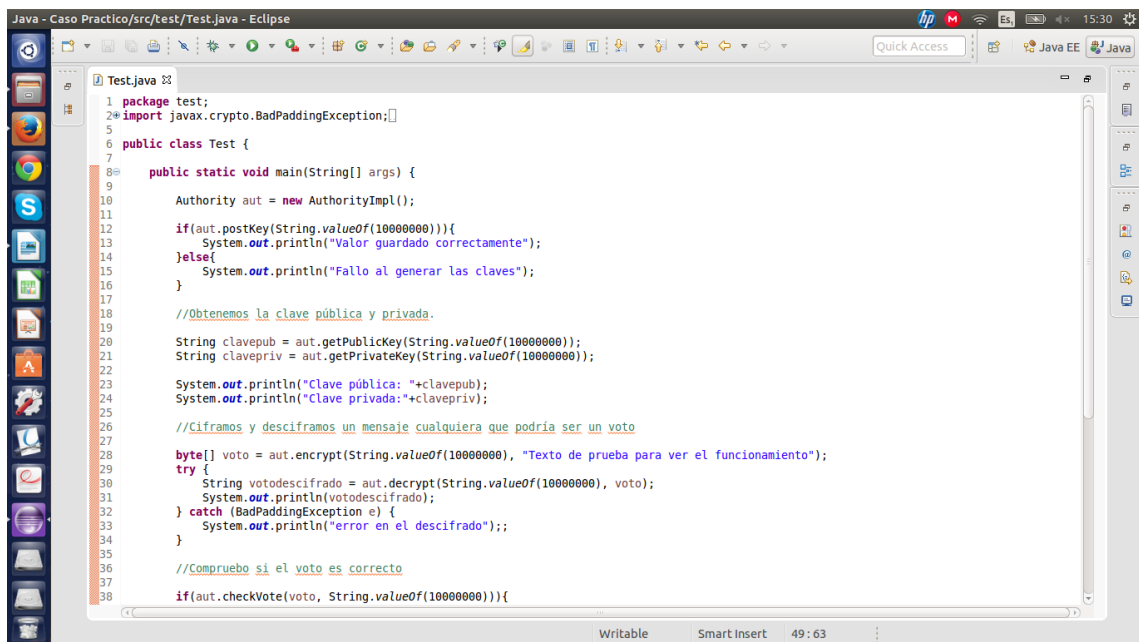


Ilustración 12. Clase Test I

```
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 voto
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 13. 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: MIGFMA0GCSqGSIb3DQEBQQUAA4GNADCBiQKgQCao47vBNvzerWsbhUF
Clave privada: MIICDQIBADANBgkqhkiG9w0BAQEFASCA18wggJbAgEAAoGBAICjju8E:
Texto de prueba para ver el funcionamiento
El voto no ha sido alterado
El voto ha sido alterado
```

Ilustración 14. Salida del Test

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

este ejercicio sí tiene que ver con el proyecto pero es tal vez el más sencillo

### 3. Gestión del Cambio, Incidencias y Depuración

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:

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

#### 3.1 Caso práctico

Para el caso práctico, nos basaremos en un caso real ocurrido durante el desarrollo de nuestro subsistema.

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:

<b>AUTOR</b>	Andrés Pachón
<b>FECHA</b>	17/11/2014 (Primera sesión de integración)
<b>DESCRIPCIÓN</b>	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.
<b>PASOS PARA REPRODUCIR EL PROBLEMA ENCONTRADO:</b>	
1.	Crear un proyecto Django con Eclipse.
2.	Tratar de importar nuestro archivo .jar.
<b>ESPERADO</b>	Importación correcta para poder utilizar nuestros métodos.
<b>RECIBIDO</b>	No es posible importar el archivo.
<b>VERSIÓN UTILIZADA</b>	Las versiones de Django, Python o Eclipse son irrelevantes en este caso.
<b>PRIORIDAD</b>	Alta
<b>INFORMACIÓN ADICIONAL</b>	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 Owner

**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 15. Ejemplo plantilla en GitHub

se podría mejorar también mucho



## 4 Mapa de herramientas

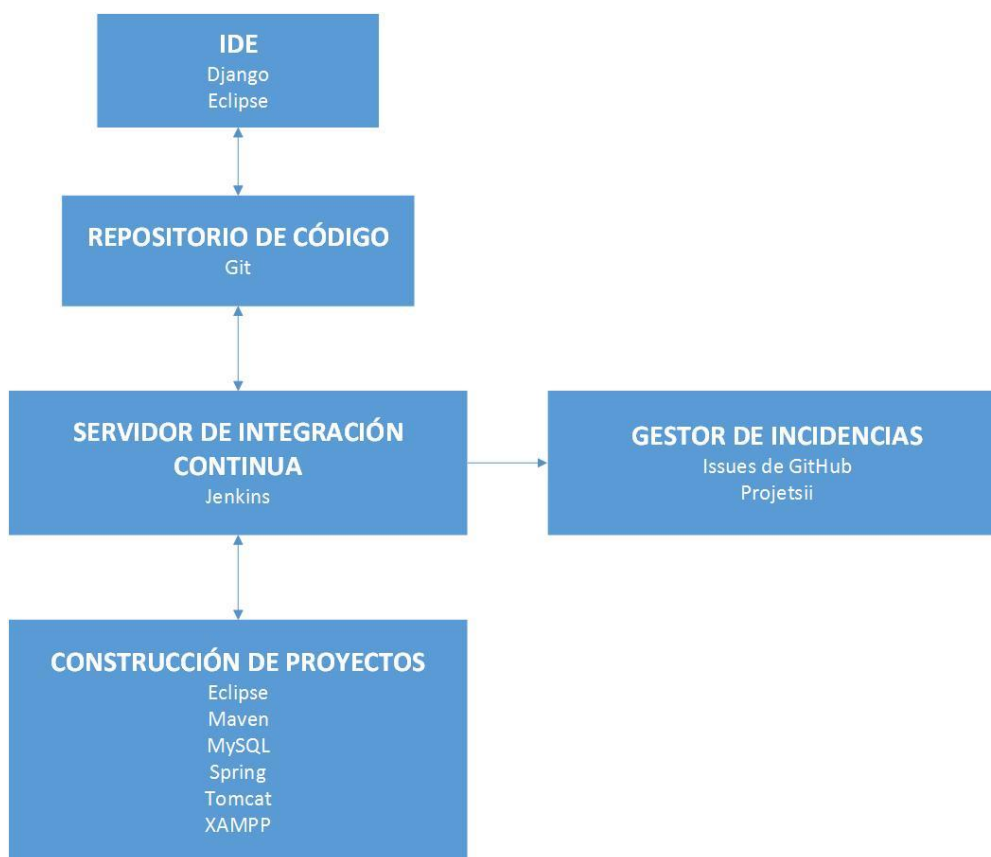


Ilustración 16. Mapa de herramientas del proyecto

**Django:** Framework de desarrollo de aplicaciones para PHP.

**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:** Por recomendación del profesorado usaremos Git como herramienta de control de versiones. En nuestro subsistema utilizamos un repositorio propio además del repositorio común.

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

**Maven:** Framework encargado de gestionar las dependencias de los proyectos.

**Spring:** Framework de desarrollo de aplicaciones para Java.

**Tomcat:** Servidor web destinado a alojar proyectos web java. Lo usarán subsistemas como “Creación/Administración de censos”. Aún faltaría por definir si cada subsistema ejecutará su funcionalidad en aplicaciones distintas o se integrará todo en una sola aplicación web.

**XAMPP:** Distribución de Apache, MySQL, PHP y Perl. Usada por el subsistema de “Autenticación”, el cual ha enfocado su funcionalidad diseñando un frontend en PHP.