Documento del proyecto

Almacenamiento de votos

Evolución y Gestión de la Configuración

Cristian Fernández Rivas — Gestor de la configuración

David Letrán González — Gestor de la configuración

Carlos López García — Gestor de la configuración

Miguel Ángel Núñez-Romero Olmo — Gestor de la configuración

Cristina Padilla Carrero — Líder de grupo

Luis Pintado Lozano — Gestor de la configuración

Rev. 1.8 | 20 de Diciembre de 2014

1. Índice

Contenido

1. Índice …………………………………………………………….………………..……….. 3

2. Control de cambios …………………………………………..……………....………….. 4

3. Resumen del proyecto ..……………………………...………….…………….………….. 5

4. Introducción ……………..……………………………...………….……….…..…………. 6

5. Gestión del código fuente ..………………………….……...…………….…..………… 7

6. Gestión de la construcción e integración continua ..………...………………………….. 10

7. Gestión del cambio, incidencias y depuración ……..……....…………………….…… 20

8. Mapa de herramientas ………...…………………………….…..………...………..… 24

9. Conclusiones …………...…………………………….………………..…..…………….. 26

10. Anexo ……………………………………….....………………..……..……………….. 27

No suele ser un capítulo más.
Mirar otros docs de ejemplo

2. Control de cambios

VERSIÓN	MOTIVO	REALIZADO POR	FECHA
1.0	Creación	Carlos López	19/10/2014
1.1.	Gestión del código fuente	Carlos López	22/11/2014
1.2.	Gestión de incidencias	David Letrán	3/12/2014
1.3.	Gestión de la integración	Miguel Ángel Nuñez-Romero	3/12/2014
1.4	Introducción	Cristian Fernández	17/12/2014
1.5	Conclusiones	David Letrán	18/12/2014
1.6	Mapa de herramientas	Luis Pintado	16/12/2014
1.7	Resumen	Cristina Padilla	18/12/2014
1.8	Corrección de errores	Todos	20/12/2014

Tabla 1: Control de cambios

3. Resumen del proyecto

Nuestro trabajo consiste en el estudio sobre la gestión de la configuración y la realización del proyecto de software Agora@US. En nuestro caso concretamente, la realización consistirá en el subsistema encargado del almacenamiento de votos.

Nuestro subsistema tendrá que cumplir con la tarea de ser capaz de almacenar votos cifrados sobre una determinada votación y devolver dichos votos cuando sean solicitados.

Para la realización de nuestro subsistema, hemos tenido que definir un sistema en el cual se recogen las pautas a seguir con respecto a la gestión del código fuente, la gestión de la construcción e integración continua, la gestión de cambios, incidencias y depuración; y las herramientas utilizadas para lograr nuestro objetivo.

En la gestión del código fuente tratamos temas como la política de checkouts que vamos a utilizar o las gestión de ramas que vamos a usar.

En la gestión de la construcción e integración continua, hablaremos sobre la metodología a seguir a la hora de integrar los distintos componentes y comprobar que dicha integración funciona a lo largo del desarrollos teniendo en cuenta las incidencias que se producen.

Para la gestión de cambios, incidencias y depuración se especificará el método a seguir cuando se detecta una incidencia o se quiere introducir un nuevo cambio para que este se apruebe y se solucione o se lleve a cabo.

La herramienta principal utilizada ha sido GitHub, que nos ofrece apoyo tanto para la gestión del código como para la gestión de cambios, incidencias y depuración.

4. Introducción

No es así.

Se quiere crear un proyecto informático que consiste en un sistema de votación, que pretende sustituir a los métodos actuales de votación para alcaldes y presidentes. También se podría hacer una serie de preguntas a los usuarios y estos votan estilo referéndum.

Para ello, se divide el sistema en 11 subgrupos para facilitar la creación del sistema, y posteriormente, se integran entre sí para que funcione correctamente. Cada subgrupo elegirá el lenguaje de programación que prefiera para su subsistema y el de la base de datos, en caso de necesitar una.

En este documento se expondrá el todo el trabajo hecho sobre el subsistema ‘Almacenamiento de votos’. Este subsistema se encarga de almacenar los votos recibidos del subsistema ‘Cabina de Votación’ y esta programado en MySQL.

Para la integración, se pretende dar dos métodos a los demás subsistemas mediante una API, uno para guardar los votos en la base de datos, y otro para obtener todos los votos de una votación.

Introducción pobre, corta y poco elaborada.

5. Gestión del código fuente

5.1. Gestión del código fuente de nuestro subsistema

Para gestionar el código de nuestro módulo de almacenamiento de votos para la aplicación Agora@US hemos utilizado un repositorio de código sobre el cual, durante el desarrollo, hemos volcado los cambios realizados durante el avance de nuestro trabajo de forma colaborativa.

En primera instancia, al comienzo de este trabajo, creamos un repositorio **Subversion** que alojaba el código en la plataforma de gestión de proyectos **Projetsii** utilizando el programa **TortoiseSVN**, eventualmente, tras conocer algo más en profundidad los valores y características de **Git**, decidimos migrar nuestro desarrollo al servicio **GitHub**.

Sobre la política de los **checkouts**, decidimos hacerlos **no reservados** tras observar que más de un miembro del equipo tendría la necesidad de trabajar sobre un mismo archivo al mismo tiempo y la opción de hacerlos reservados podría retrasar el trabajo. Además, forzamos un conflicto editando un mismo archivo en paralelo y tras intentar confirmar el cambio se ha producido el conflicto. La herramienta que usamos para gestionar el repositorio nos lo había notificado pudiendo así localizar el archivo en concreto. Para solventarlo utilizamos la herramienta de **diff** integrada en TortoiseSVN para abrir ambas versiones en paralelo y poder ver las diferencias. Por último mezclamos ambas versiones, utilizando lo válido de ambas y hemos cambiado el estado del archivo para confirmar que ya no existía conflicto.

Habéis hecho esto mismo en GIT?

En cuanto a la gestión de las ramas de nuestro proyecto, para el proyecto entero, pensamos que sería una buena opción hacer una rama por motivos físicos, es decir, una rama por cada subgrupo perteneciente al proyecto, pudiendo haber dos o más subgrupos que compartan una misma rama en caso de que la funcionalidad sea compartida. Dentro de nuestro subgrupo, pensamos que deberían existir ramas por motivos de entorno, en el caso de que decidiéramos cambiar, por ejemplo, el tipo de la base de datos. En cuanto a la política sobre los **merge**, hemos decidido que sólo se hará merge de una tarea con la **baseline** cuando dicha rama se considere completa y sin errores. En caso de ser una rama generada debido a algún motivo de entorno, se realizará merge cuando se consiga hacer funcionar la baseline en dicho entorno. Una rama se eliminará cuando se realice un merge de esa rama. Por último, con respecto a los permisos de usuarios, dentro de la rama perteneciente a nuestro subgrupo , todos tendremos el máximo nivel de permiso sobre todas las ramas y subramas ya que todos tenemos la misma autoridad sobre el código que desarrollamos.

Apartado extramadamente pobre en su descripción. Se podría comentar mucho más
Denota o bien poco tiempo en la elaboración o bien que se ha hecho realmente poco
en durante el curso en cuánto a gestión del código fuente.

5.2. Ejercicio práctico sobre gestión del código fuente

Enunciado:

Crear un repositorio en GitHub y guardar en el mismo el código de un proyecto llamado HolaMundo. Crear una nueva rama dado que se ha producido un cambio en el lenguaje a utilizar en este proyecto. Realizar el cambio sobre la nueva rama. Hacer un commit. Abrir un Pull Request y unirlo a la rama principal del proyecto.

Solución:

1.) Una vez iniciada la sesión en GitHub con tu usuario y contraseña, hacer clic en el icono **+** de la parte superior a la derecha y rellenar el formulario para la creación de un repositorio. Hacer clic en **Create repository.**

2.) Al crear el repositorio, por defecto, solo hay una rama llamada **master**. Para resolver una Issue, lo más recomendable es hacerlo en una nueva rama. Para crear esta rama, dentro de la página de nuestro repositorio, haz clic en el desplegable llamado **branch: master**, escribe un nombre para la rama y pulsa **Enter**. Ahora tienes dos ramas que, por ahora, contienen exactamente lo mismo.

3.) Para disponer de este repositorio en local, puedes hacerlo de dos formas:

- Mediante la aplicación oficial de escritorio de GitHub, una vez iniciada sesión, haz clic en el icono **+** y posteriormente en **clone**. Aparecerán todos tus repositorios y debes elegir el repositorio sobre el que estamos trabajando.

- La otra forma es mediante la consola, ejecutando el siguiente comando:

```
git clone git@github.com:USUARIO/REPOSITORIO.git
```

4.) Una vez realizado el cambio utilizando tu IDE o editor de texto favorito, puedes hacer commit de dos formas análogamente al paso anterior:

- Mediante la aplicación oficial de escritorio de GitHub, tras haber realizado la modificación, aparecerá una ventana llamada **Uncommited changes**, dentro de la misma rellena el nombre y la descripción del commit y haz clic en **Commit to [rama]**

y a continuación en la opción **Sync**, ubicada en la esquina superior izquierda para enviar los cambios a la copia almacenada en GitHub.

- La otra forma es mediante la consola, ejecutando los siguientes comandos:

```
git add [nombre del archivo]
git commit -a
git push
```

5.) Abrir un **Pull Request** consiste en proponer unos cambios para que sean discutidos y, cuando sean finalmente aceptados, se unan a la rama principal del proyecto. Para hacer esto, haz clic en la opción **Pull Requests**, ubicada en la barra lateral y haz clic en el botón **New pull request**, a continuación, selecciona la rama sobre la que has hecho los cambios que deseas unir a la rama principal, observa la diferencia de los cambios que existen comparando la rama master con la nueva, y para finalizar, pulsa en el botón **Create pull request**, escribe un nombre para el Pull Request y pulsa el botón **Create pull request**.

6.) Para unir el Pull Request a la rama principal del proyecto, simplemente haz click en **Confirm merge** y borra la rama, ya que la información que contiene ya se encuentra en la rama principal del proyecto.

Un ejercicio muy simple y pobremente descrito.

6. Gestión de la construcción e integración continua

Empezaremos hablando de la construcción e integración de nuestro subsistema (Almacenamiento de votos) de forma interna, donde hablaremos de cómo se ha realizado y daremos algunas propuestas de cómo podríamos mejorar la integración. Del mismo modo continuaremos hablando de la gestión e integración de nuestro subsistema con el resto de subsistemas que depende, así como la construcción e integración de todo el proyecto.

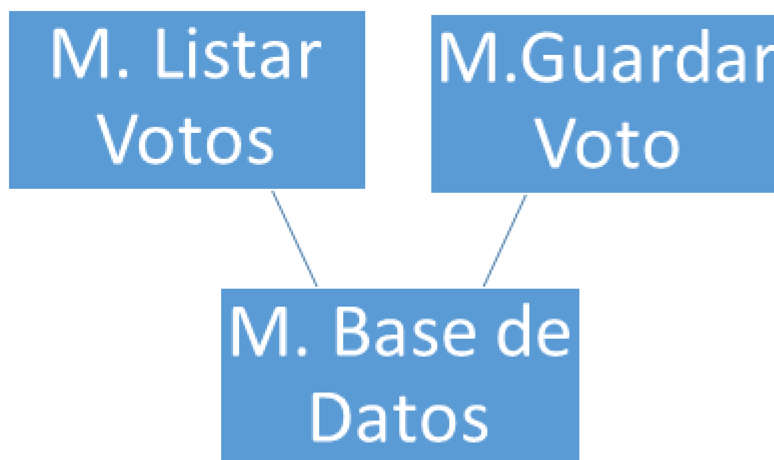
6.1. Integración interna del subsistema

Nuestro subsistema se compone básicamente de tres módulos:

- **Módulo base de datos**: Es la base de datos de nuestro sistema, en él se almacenarán los votos cifrados.

- **Módulo API para guardar voto**: Este módulo escrito en PHP nos permite guardar un voto en nuestra base de datos a través de un método POST.

- **Módulo API listar votos de una votación**: Este módulo escrito en PHP nos permite listar todos los votos de una determinada votación a través de un método GET.



Como se observa en el diagrama, el módulo para listar votos y guardar votos necesitan acceder a la base de datos para cumplir con su función.

Ya que nuestro subsistema no dispone de una capa de presentación, vimos oportuno utilizar una integración bottom-up, es decir empezamos desarrollando la capa de datos hasta la capa de “lógica”. Por lo tanto, empezamos desarrollando el módulo base de datos, luego desarrollamos el módulo que se encargaba de guardar un voto. Una vez que éste último módulo estaba preparado, lo integramos con él módulo de base de datos. Por último desarrollamos el módulo para listar votos, y una vez que estaba listo, lo integramos al igual que el anterior con el módulo base de datos.

La forma en la que comprobamos si un módulo estaba listo era a través de pruebas unitarias:

- **M. Base de datos:** Para comprobar si este módulo funcionaba correctamente insertamos ejemplos de prueba en la base de datos, comprobamos que se habían guardado correctamente y luego los listamos.

  Forma pobre de hacer pruebas. En IISSI se ven formas más avanzadas de hacerlo. Al menos con CRUD

  Ejemplo de inserción de un voto:

  ```
  INSERT INTO Votes(vote, votation_id) VALUES ('$vote', '$votation_id');
  ```

  Ejemplo de listado de votos:

  ```
  SELECT vote FROM Votes WHERE votation_id = '$votation';
  ```

- **M. API para guardar votos y M. API para listar votos:** La explicación se hará de forma conjunta ya que se han realizado de forma similar. Como la integración con la base de datos todavía no se había realizado se simulaban esos datos a través de variables. Se comprobó que al hacer uso de la API para guardar votos, el número de votos almacenados en una variable en forma de lista aumentaba y el voto tenía el formato correcto (*votación – voto cifrado*). Se comprobó que al hacer uso de la API para listar votos ésta se realizara de acuerdo al formato especificado.

¿por qué?

No creímos necesario el uso de ninguna herramienta para realizar estos test unitarios, nos basamos en que los resultados tras una operación fuesen correctos. Una buena práctica sería utilizar test unitarios para cada una de las funcionalidades internas de nuestra aplicación. En nuestro caso nos encontramos con que tenemos código escrito en PHP y código en SQL. Para las pruebas SQL tenemos la opción de realizar paquetes de prueba donde para cada entrada esperamos recibir una salida determinada. En el caso de PHP, con una simple búsqueda en internet tenemos herramientas que nos permiten realizar las pruebas que deseamos, una de éstas es PHPUnit.

Sería recomendarble hacer uso de ellas.

Una vez que los distintos módulos funcionan de forma correcta individualmente, integraremos unos con otros. Recordemos que las integraciones son M. base de datos con M. API para guardar votos y con M. API para listar votos. Para resolver estos casos conectamos los distintos módulos con la base de datos a través de una conexión y sustituimos las variables, que habíamos creado antes para simular el comportamiento, por inserciones y recogida de información real sobre la base de datos. Al igual que antes no utilizamos ninguna herramienta que nos permitiera comprobar de forma continua que estas integraciones se realizaban correctamente, simplemente comprobamos de forma manual que todo el subsistema estaba integrado y funcionaba correctamente. Sin embargo, sería una buena práctica utilizar los métodos mencionados anteriormente (paquetes de pruebas, PHPUnit, etc.).

6.2. Integración del subsistema con los subsistemas relacionados

En este apartado explicaremos la forma de gestionar la comunicación con los subsistemas que estamos directamente relacionados, siendo estos Cabina de Votación y Recuento.

- **Cabina de Votación:** Entre otras, la función de cabina de votación consiste en cifrar los votos y enviarnos esos votos cifrados para que nuestro subsistema lo almacene.

- **Recuento:** Necesita hacer un recuento de una determinada votación. Para ello necesita comunicarse con nuestro subsistema ya que es el que tiene almacenado los votos.

Recordamos que la forma en la que nos con la que nos comunicaremos con los demás subsistemas será mediante los dos módulos para la API que hemos creado. Decidimos realizar esta implementación para la comunicación ya que los subsistemas con los que nos comunicamos están desarrollados con distintos lenguajes de programación (Java y Python). Estos módulos los subimos a un servidor externo para que ningún subsistema tuviera la necesidad de instalar nada para usar nuestra funcionalidad.

Un diagrama que describiría adecuadamente esta situación sería el siguiente:

¿No tenéis interacción con modificación de resultados?

Y los métodos se utilizan de la siguiente forma:

Método (URL)	Tipo	Descripción	Parámetros	Respuesta	Ejemplo
vote(http://php-egc.rhcloud.com/vote.php)	POST	Permite almacenar un voto para una determinada votación.	**vote:** voto cifrado. **votation_id:** id de la votación	Json con un mensaje de respuesta que indica si la operación se ha resuelto correctamente. (el mensaje será 1 si todo salió bien y 0 en caso contrario)	{“msg”:1}
get_votes(http://php-egc.rhcloud.com/get_votes.php)	GET	Devuelve la lista de votos de una determinada votación.	**votation_id:** id de la votación.	Json con la lista de votos y un campo “msg” que indica si la operación se realizó correctamente.	{“votes”:[“voto1”,” voto2”],” msg”:1}

No se deja claro si se devuelven también los votos cifrados. Se entiende que sí.

Además de la creación de la API, para facilitar la integración con los subsistemas, les proporcionamos a cada uno unos métodos implementados en sus respectivos lenguajes de programación para que pudieran usar nuestra API. Estos métodos fueron probados por nuestro subsistema. Sin embargo, no realizamos los tan importantes test de integración continua que nos permiten comprobar que la integración se está realizando de forma correcta con el paso del tiempo, donde seguro que se producirán cambios. Hemos estudiado y propuesto como lección aprendida la utilización de la herramienta Jenkins para este fin.

Debería haberse al menos intentado usar

Como ejemplo de uso de Jenkins en la integración de almacenamiento con Recuento y Cabina de Votación proponemos la creación y posterior ejecución de pruebas diarias (usando si se desea los fragmentos de código proporcionados) que nos permitan saber en todo momento si nuestro sistema se integra de forma correcta con el resto los subsistemas ante un posible cambio tanto interno (nuestro subsistema) como externo.

Como propuesta teórica, la integración se realizaría primero con cabina de votación ya que necesitamos que haya votos guardados en nuestra base de datos para que posteriormente se pueda realizar el recuento de una votación. En la práctica la integración se ha realizado de ese modo, sin embargo, los votos que se encuentran actualmente en nuestra base de datos o son de prueba o no están cifrados, ya que Cabina de Votación aún no ha podido cifrarlos.

6.3. Propuesta de comunicación de todos los subsistemas

Recordemos que los subsistemas del proyecto son:

- Autentificación.
- 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.

En la práctica los subsistemas nos hemos integrados por fases (“Big Bang”), cada uno de los subsistemas se ha desarrollado de manera independiente y luego se ha combinado todo en las sesiones de integración. Los resultados de esta práctica han sido conflictos que nos está costando mucho solventar al integrar la mayoría de los subsistemas. Como lección aprendida aportamos que la integración incremental hubiera sido una mejor opción. Para ello crearíamos un primer esqueleto de todo el sistema comunicándose y luego añadiríamos poco a poco las “piezas” que le faltan comprobando que la integración siga funcionando de forma correcta.

Proponemos que para el orden de una integración de todo el sistema esta se haga de acuerdo a la funcionalidad del sistema. Siendo este un sistema de votación, primero necesitaríamos que un usuario se identifique, que cree una votación, que cada uno de los usuarios vote, se haga un recuento, etc. Por eso proponemos que el orden de la integración sea el siguiente:

1. Autentificación.
2. Creación/administración de censos.
3. Creación/administración de votaciones.
4. Verificación.
5. Cabina de votación.
6. Almacenamiento de votos.
7. Recuento.
8. Modificación de resultados.
9. Frontend de resultados.
10. Visualización de resultados.
11. Deliberaciones.

Sin embargo, para paralelizar la integración y ante la posibilidad de que haya subsistemas que no estén listos para la integración, damos la posibilidad de que determinados subsistemas se puedan integrar de forma independiente al orden. Un ejemplo de esto sería la forma en la que nosotros nos hemos comunicado con los subsistemas relacionados, ya que nos hemos integrado cabina-almacenamiento-recuento, sin embargo cabina todavía no se había podido integrar correctamente con verificación. Somos conscientes de que esta forma de integrarse supone un riesgo, ya que no sabemos si la integración de por ejemplo verificación con cabina podría afectar a la integración que hemos realizado previamente, por lo que habría que controlarlo.

6.4. Propuesta para la integración continua y acciones realizadas al respecto

Falta mucho para que nuestro sistema lleve a cabo una buena práctica de integración continua, pero se han llevado algunas acciones y propuestas que nos acerquen a este objetivo:

No es un requisito imprescindible.

- **Mantener un único repositorio:** Al inicio cada uno de los subgrupos tenían un repositorio de código diferente, incluso de diferentes plataformas. Al cabo del tiempo todos los grupos hemos convergido a un único repositorio de código, al cual podemos acceder desde aquí: https://github.com/EGC-1415-Repositorio-compartido/repvoting .

- **Test de integración:** para saber que la integración se realiza de manera eficiente y no "subamos" cambios que afecten a la integridad del sistema, se ha propuesto que los grupos utilicen un servidor Jenkins para que los elementos nuevos pasen una batería de pruebas de forma que estemos algo más seguros de que el trabajo se ha realizado de forma correcta.

- **Inspección de calidad del código:** para asegurarnos de que nuestro código dispone de un mínimo de calidad proponemos la utilización de herramientas como por ejemplo SonarQube para la programación en Java.

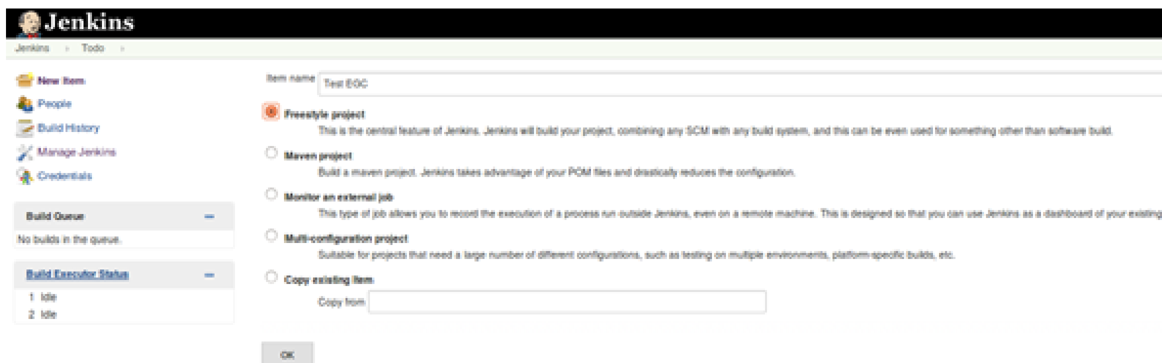
6.5. Ejercicio práctico sobre integración continua

Enunciado:

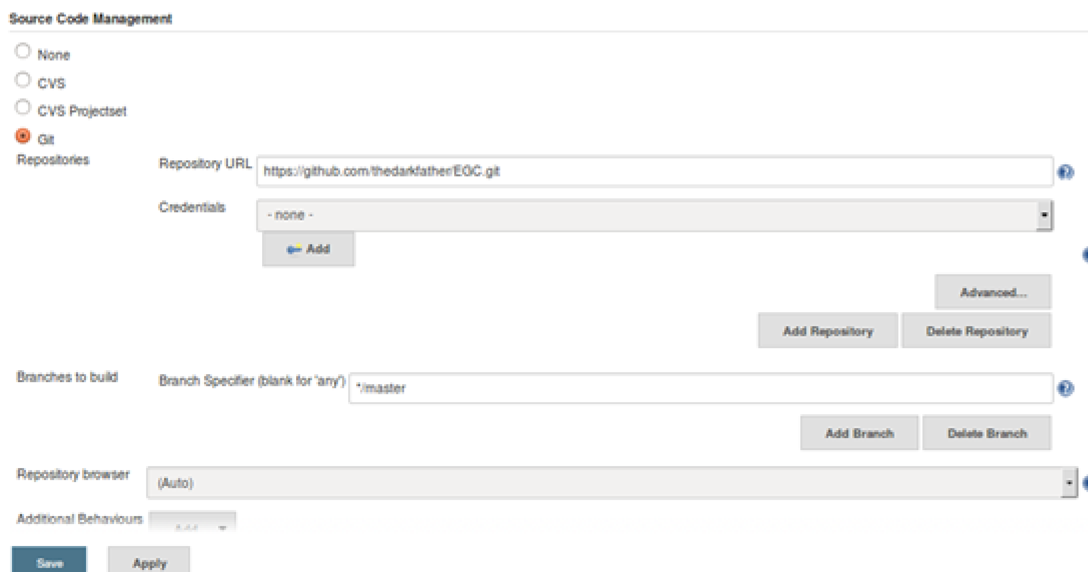
Asegurar la integración continua durante el desarrollo de la aplicación. Crear un test de prueba donde se compruebe que la integración de los distintos elementos de la aplicación es correcta.

Solución:

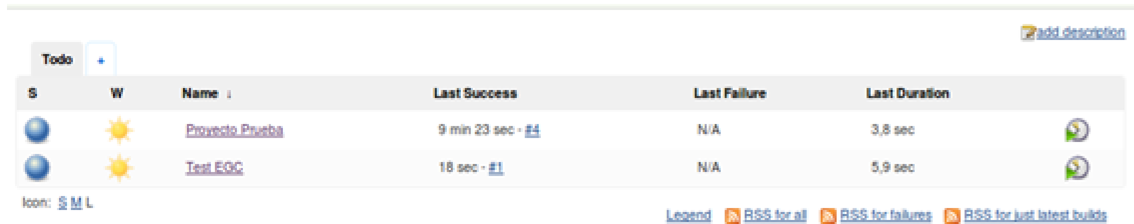
Crear un nuevo proyecto Jenkins. Para ello pulsamos en "New Item". Ponemos un nombre al proyecto, indicamos que es un proyecto libre y pulsamos en "ok".



1) Nos aparecerá una ventana de configuración. Añadimos si queremos una descripción al ítem anterior y le indicamos que utilizaremos un repositorio git y la dirección de éste.

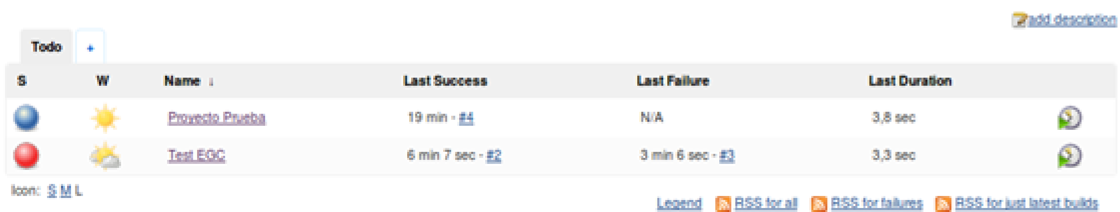


2) Vemos que el nuevo item se ha creado en la pantalla principal de Jenkins. Pulsamos en “Schedule a built”. Si todo ha ido correctamente nos aparecerá el símbolo de un sol brillando a la izquierda de nuestro proyecto.

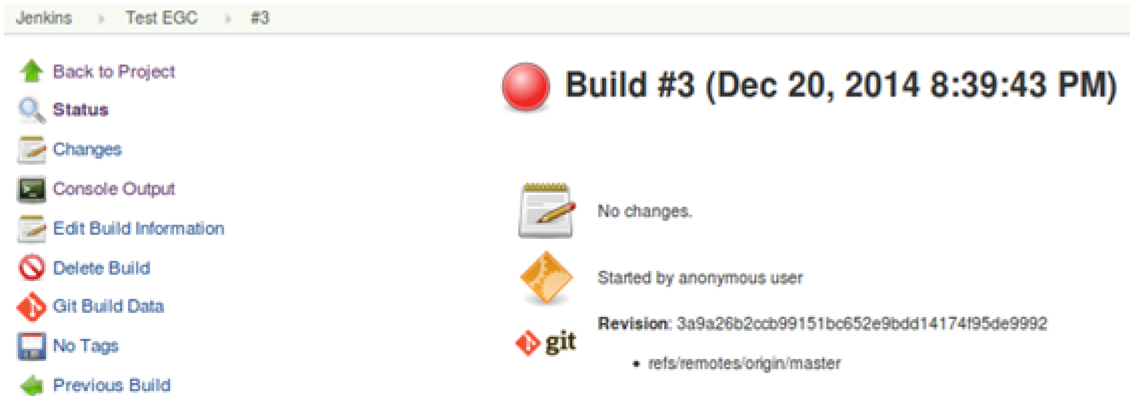


3) Accediendo a la configuración de nuestro proyecto añadimos a la construcción un paso nuevo en el que vamos a ejecutar el archivo test por consola y guardamos.

4) Construimos de nuevo y vemos que la ejecución ha fallado.



Observamos que el sol ya no está tan brillante como antes, ya que una de las ejecuciones ha fallado. Si accedemos a ese proyecto podemos ver a qué se debe ese fallo.



También podremos acceder a la salida de la consola para observar el error con más detalle.

5) Corregimos el error y volvemos a lanzar la ejecución. Y Comprobamos que la nueva ejecución funciona correctamente.

Console Output

```
Lanzada por el usuario anonymous
Ejecutando.en el espacio de trabajo /var/lib/jenkins/jobs/Test EGC/workspace
 > git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url https://github.com/thedarkfather/EGC.git # timeout=10
Fetching upstream changes from https://github.com/thedarkfather/EGC.git
 > git --version # timeout=10
 > git fetch --tags --progress https://github.com/thedarkfather/EGC.git +refs/heads/*:refs/remotes/origin/*
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
 > git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision c7a9c878c1eb81e8ddec9c3d7dc379dc52d418ea (refs/remotes/origin/master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f c7a9c878c1eb81e8ddec9c3d7dc379dc52d418ea
 > git rev-list 3a9a26b2ccb99151bc652e9bdd14174f95de9992 # timeout=10
[workspace] $ /bin/sh -xe /tmp/hudson2260284960749249476.sh
+ python test1.py
todo ha ido correctamente
Finished: SUCCESS
```

Nota: Todos los errores e incidencias deberían ser reportadas como se describe en el apartado gestión de cambios, incidencias y depuración.

No queda claro si el ejercicio se ha hecho con vuestro propio proyecto. En general parece como si vuestro proyecto estuviera poco sincronizado con las cosa que proponéis en el documento.

7. Gestión del cambio, incidencias y depuración

7.1. Gestión de incidencias

En este apartado se describe el sistema que seguimos para reportar incidencias que requieran cambios así como el proceso que se seguirá para su resolución.

La herramienta utilizada para dicho propósito será Github, una vez detectada una incidencia se deberá acudir al apartado de incidencias de la aplicación y crear una nueva. La incidencia deberá seguir el siguiente formato:

Título:
Descripción:
Versión:
Entorno:
Pasos para reproducirla:
Comportamiento esperado:
Comportamiento observado:

En el campo título se debe poner una frase que resuma el problema y que permita diferenciar una incidencia de otra claramente. En el campo descripción se incluirá una pequeña reseña que enuncie el problema detectado. En el campo versión se pondrá la versión de la aplicación en la que ha ocurrido, en nuestro caso esto será el ID del commit correspondiente en Github. En la parte de entorno se debe especificar el entorno en el cual se ha desarrollado la incidencia así como la versión del mismo y cualquier detalle disponible sobre su configuración que pueda aportar información para la resolución de la incidencia. Lo siguiente será especificar los diferentes pasos a seguir para reproducir la excepción y por último especificar el comportamiento que se esperaba de la aplicación y el comportamiento que se ha observado en su lugar.

Dado que Github nos ofrece por defecto un campo de título y uno de descripción se usará el de título para el título y el de descripción para el resto de campos.

Una vez tenemos la incidencia en este formato se le asignará una etiqueta. Las etiquetas se ajustarán a la siguiente tabla:

Abierta	Incidencia nueva a la espera de ser confirmada por un revisor*
Aceptada	Incidencia confirmada a la espera de ser asignada a un miembro del equipo
Asignada	Incidencia asignada a un miembro del equipo y en proceso de ser resuelta
Arreglada	Incidencia ya solventada

**Un revisor será cualquier miembro del equipo siempre que no sea el mismo que añadió la incidencia*

¿por qué estas etiquetas y no otras? ¿quién os ha inspirado?

Las etiquetas se corresponden con el estado de la incidencia, una incidencia recién abierta deberá adoptar el estado de Abierta. Los estados tendrán colores relacionados con lo cerca que está una incidencia de resolverse, así cuanto más cerca de resolverse esté más 'frío' será el color de la etiqueta. Además se deberá añadir la etiqueta 'Incidencia' y la etiqueta correspondiente a la prioridad asignada a la incidencia que podrá ser Alta, Media o Baja.

Alta	Debe hacerse/arreglarse a toda costa para cumplir los objetivos actuales
Media	Se debería hacer/arreglar para cumplir los objetivos actuales
Baja	Se podría hacer/arreglar pero no es necesario para cumplir los objetivos actuales

Por tanto cada incidencia deberá tener 3 etiquetas en total, de esta manera será fácil ver de un simple vistazo las más urgentes.

Por último, gracias a la interfaz de Github los demás miembros del equipo podrán añadir comentarios en caso de que sea necesario.

7.2. Gestión de cambios

Para crear una solicitud de cambio el proceso será el mismo, excepto que en el formato se deben obviar los campos no necesarios, esto es todos los campos excepto Título, Descripción y Entorno, y se añadirá la etiqueta 'Solicitud' en lugar de la etiqueta 'Incidencia'

7.3. Depuración

Para la depuración no se definirá una herramienta que todo el equipo deba usar sino que esta se dejará a la elección personal. La depuración se realizará añadiendo puntos de ruptura en el código tras realizar hipótesis sobre dónde puede estar el error y comprobando que los valores de las variables sean los esperados. Una vez localizada la zona de conflicto se procederá a solventarlo de la mejor manera posible.

¿Habéis hecho algo en este sentido?

7.4. Ejercicio práctico sobre gestión de cambios, incidencias y depuración:

Enunciado:

No se entiende bien el enunciado.

Realizando un snippet de código en Python para qué Cabina de votación pudiera integrarse fácilmente con nosotros se detecta una incidencia y por lo tanto hay que reportarla.

Solución:

Primero se crea la incidencia en Github con el formato correspondiente:

Título: Introducción fallida de un voto en la base de datos usando la API

Descripción: Al intentar introducir un voto en la base de datos usando la API, el método responde positivamente pero al listar los votos aparece como un voto vacío.

Versión.

Entorno:
- LiClipse v1.2.1
- Advanced REST Client v3.1.9

Pasos para reproducirla:

1. Consultar en la base de datos los votos para la votación con id=1
2. Usar como URL = http://php-egc.rhcloud.com/vote.php con método POST, introducir la cabecera Content-Type = application/json y en el cuerpo vote=test&votation_id=1
3. Realizar la petición
4. Consultar en la base de datos los votos para la votación con id=1

Comportamiento esperado:

Recibir una lista de votos que comparada con la recibida en el paso 1 aparece un voto nuevo con valor 'test'.

Comportamiento observado:

Recibir una lista de votos que comparada con la recibida en el paso 1 aparece un voto nuevo con valor ''. Se le asigna la etiqueta Incidencia con prioridad Alta y estado Abierta.

Una vez revisada por otro miembro y aprobada se le cambia la etiqueta de estado a Aprobada, luego es asignada a algún miembro del equipo ya sea porque esté involucrado en la parte en la que se produce el fallo o por estar más libre que el resto del equipo y le cambia el estado a Asignada.

Se localiza el fallo depurando y se comprueba que la cabecera Content-Type no era necesaria por lo que se elimina, se vuelven a reproducir los pasos y se comprueba que el comportamiento observado es el esperado. Se cambia el estado de la Incidencia a Arreglada y se termina.

Ejercicio mal explicado y difícil de entender.

8. Mapa de herramientas

Se ha utilizado el sistema operativo Windows, por lo que el conjunto de herramientas a utilizar viene dado por las restricciones de compatibilidad que este aplica.

Se proporciona un diagrama claro y conciso en el que se muestran las herramientas utilizadas para cada actividad, además de las relaciones entre los conjuntos de herramientas utilizadas.

Herramientas de desarrollo

Para las actividades de codificación, hemos utilizado dos IDEs: Liclipse y Netbeans. El primero de ellos lo utilizamos para desarrollar en Java y Python los fragmentos de código necesarios para otros subsistemas para integrarse con nosotros. Netbeans lo utilizamos para desarrollar en PHP toda la funcionalidad de nuestro subsistema. Además, para modificaciones rápidas hemos optado por usar la aplicación SublimeText.

Para desplegar el sistema en local y poder probarlo adecuadamente, utilizamos XAMPP para configurar un servidor compatible con PHP 5.3 junto con una base de datos MySQL. Para gestionar la base de datos de una manera más clara y sencilla utilizamos MySQL Workbench.

Herramientas de gestión del código fuente

Al principio de la asignatura se utilizó un repositorio de código SVN(ofrecido por Projetsii, aunque debido a su inestabilidad, también tuvimos que recurrir a Assembla), pero al comprobar que Git nos ofrecía una funcionalidad más avanzada, decidimos migrar nuestro subsistema a un repositorio de Git de GitHub. Para interactuar con el repositorio SVN, utilizamos TortoiseSVN, mientras que para hacerlo con el de Git, utilizamos la aplicación de escritorio de GitHub junto con órdenes de Git de la consola de comandos.

Herramientas de gestión del proyecto, incidencias e integración

Nuestro primer acercamiento para gestionar nuestro proyecto fue el uso de la plataforma Projetsii, pero debido a su inestabilidad, decidimos usar Assembla. En cualquier caso, esto se vio sustituido de manera temprana por GitHub, ya que nos ofrecía una manera más clara, ágil y sencilla de trabajar, además de que nos permitía centralizar casi todas las tareas que necesitábamos llevar a cabo gracias a sus repositorios Git y su gestión de incidencias. También se creó el repositorio compartido del proyecto en GitHub.

Herramientas de despliegue

Para desplegar en la nube nuestro subsistema optamos por las soluciones gratuitas ofrecidas por Openshift. Pudimos desplegar una base de datos MySQL y un servidor de aplicaciones compatible con PHP. Para hacer peticiones de prueba a nuestra aplicación, usamos la extensión de Chrome AdvancedRestClient, que nos permitió insertar votos de prueba y recuperarlos.

Mapa de herramientas

9. Conclusiones

Tras la experiencia obtenida durante el desarrollo del proyecto y la gestión de la configuración del mismo podemos sacar algunas buenas prácticas que realizar en próximos proyectos.

La gestión de la configuración debe quedar clara desde el principio ya que de esta manera el esfuerzo a lo largo del proyecto se ve concentrado en la propia ejecución del proyecto y no en gestionarlo. Puede requerir un gran esfuerzo el aprendizaje de las herramientas usadas para la gestión pero una vez aprendidas se ve recompensado gracias a que permite una comunicación más clara entre los miembros del equipo tanto en gestión de código como de incidencias que se producen o cambios que se pretenden introducir.

Me parece una conclusión muy interesante.

Gracias a este proyecto, por destacar un aspecto sobre los demás, hemos entrado en contacto con herramientas de integración continua como Jenkins, que si se usa de un modo conveniente, se convierte en una potente parte del sistema que debería ser cada vez más imprescindible en proyectos venideros por la gran seguridad que otorga.

No sé si seguridad es la palabra

Por último, decir que una integración continua durante el proyecto permite detectar errores de manera temprana haciendo que estos no se acumulen en una integración final que podría alargarse más de lo deseado.

10. Anexo

Junto al documento del proyecto se adjuntan:

- El diario de grupo.
- Las actas de reunión (incluídas en el diario de grupo).