.

EVOLUCIÓN Y GESTIÓN DE LA CONFIGURACIÓN

GRADO INGENIERÍA DEL SOFTWARE
ETSI INFORMÁTICA

UNIVERSIDAD DE SEVILLA

Grupo Modificación de Resultados(2014/2015)

Control del documento

- Registro de cambios

Versión	Motivo	Autor	Fecha
1.0	Creación del documento	Sinuhé Gutiérrez Gómez	26/11/2014
1.1	Definición de los apartados a realizar	Daniel Jiménez García	5/12/2014
1.2	Realización de los apartados del documento	Todos los integrantes	14/12/2014
2.0	Finalización del documento	Todos los integrantes	21/12/2014

Componentes

NOMBRE	ROL
Gerena Román, Antonio José	Desarrollador
Gonzalez Fuentes, Daniel	Gestor de la configuración
Gutiérrez Gómez, Sinuhé	Gestor de la configuración
Jiménez García, Daniel	Gestor de la configuración
Llamas Lafuente, Ángel Lorenzo	Gestor de la configuración
Moreno Acosta, Tomás	Gestor de la configuración
Rodríguez Tinoco, Jesús	Gestor de la configuración
Salas Fernández, Samuel	Gestor de la configuración
Vázquez Torres, Gabriel	Jefe de Proyecto

Resumen

El proyecto que llevaremos a cabo consiste en un servicio web de software libre destinado a dar soporte para poder realizar votaciones seguras y transparentes. La aplicación en cuestión, llamada Agora Voting[1], recogerá los votos realizados por todos los usuarios haciendo diversas operaciones sobre los mismos: recuento, ponderación,… Una vez realizadas las operaciones el servicio mostrará los resultados de las votaciones así como estadísticas sobres las mismas.

Para su desarrollo, en el que colaboramos alrededor de 80 personas, hemos dividido las funcionalidades del proyecto en varios subgrupos o subsistemas. Un subsistema es una división del sistema principal (Ágora Voting) en pequeñas partes que tienen una funcionalidad específica y que están relacionados directa o indirectamente con las demás partes del sistema. Por ello, debemos realizar una buena gestión para resolver, de la mejor forma posible, los problemas con los que nos encontremos. Vamos a describir los procesos que vamos a llevar a cabo a la hora de desarrollar nuestro subsistema, las herramientas para gestionar su código fuente, las incidencias o bugs que pueden surgir tanto a nivel interno como con cualquier otro subsistema, así como la estructura de nuestro subsistema, la integración con los demás subsistemas que forman parte de Ágora Voting y la funcionalidad de nuestro subsistema.

Nuestro subgrupo, en concreto, se corresponde con el subgrupo de Modificación de resultados. Nuestro objetivo es añadir al sistema la funcionalidad de recuperar los votos realizados en la votación y ponderarlos en función de la ponderación elegida por el usuario que haya sido encargado de crear la votación. Dicha modificación puede ser por distintos factores (localidad, género, situación laboral, etc) que afecta directamente en la ponderación de los votos y en el resultado de la votación.

[1] Ágora Voting: https://agora.agoravoting.com/

Índice

Lo ideal sería que en la intro no aparezca directamente la descripción de vuestro sistema. Esto debería ir en otro apartado concreto

1.Introducción

De entre todos los subsistemas de Agora voting (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 y Cabina de votación) estamos trabajando como sistema de modificación de resultados.

Nuestro sistema se especializa en la modificación de los votos según un criterio específico y hacer un recuento de los mismos para ofrecerlo al sistema "Frontend de resultados" si lo necesitara ya que a Frontend de resultados recibirá un parámetro por el que decidirá si consume la API de nuestro sistema o de la de recuento. El voto lo recogemos del sistema Almacenamiento de votos a partir de un GET a su API. Además, como los votos recogidos están codificados utilizamos la librería del sistema Verificación para decodificar el voto y comprobar que ese voto no se ha modificado después de su creación.

Anteriormente esta funcionalidad era distinta: Sistema para interpretar los resultados de una votación y ofrecer ordenamiento de opciones concreto según diferentes reglas variables. La mayoría de las votaciones no son directas, sino que hay una serie de reglas que ordenan los resultados. Esto puede servir por ejemplo para formar listas con criterios de paridad, criterios de localidad, quitar de los resultados candidatos retirados, etc. Este sistema además de alterar el resultado final debe ofrecer una serie de estadísticas y desviaciones. (Esta funcionalidad se modificó por motivos de académicos y de complejidad.)

A qué os referís con esto?egcegc

La modificación del voto la realizamos a partir del atributo comunidad ya que según la comunidad tendrá una ponderación diferente. Este criterio se ha definido aleatoriamente. Toda esta información se puede encontrar más detalladamente en el apartado sistema de modificación de resultados de la WIKI. Esta es la url: "https://1984.lsi.us.es/wiki-egc/index.php/Grupo_Modificaci%C3%B3n_de_resultados_(2014-15)"

Lo he mirado y no está muy ordenado. Debería estar mejor escrito aquí.

También en este documento se explicarán los procesos, técnicas y herramientas para la gestión del código del proyecto. Se abordarán puntos como la gestión del código fuente en el que se introducirá cómo hemos gestionado las ramas en nuestro proyecto y con qué herramienta, cuáles son los roles capaces o no de manejarlas y cómo tenemos estructurado la forma de realizar un parche.

No es una muy buena práctica...

Respecto a la gestión de la construcción e integración continua definiremos los procesos a que se usan a la hora de construir el proyecto así como las herramientas que utilizamos y cómo las utilizamos. Por ejemplo, para la gestión de la construcción e integración continua hemos establecido un entorno de desarrollo dentro de una máquina virtual para que todos los componentes de nuestro subgrupo trabajen con las mismas versiones y la misma configuración. Para la máquina virtual hemos usado como herramienta VirtualBox. Hemos creado una máquina virtual corriendo con el sistema operativo Ubuntu. Dentro de la misma tenemos montado un entorno de trabajo donde encontramos nuestro proyecto desarrollado bajo la herramienta Eclipse. A su vez en este punto trataremos temas como la periodicidad con la que realizamos la construcción de nuestro proyecto y qué mecanismos de integración continua utilizaremos, que particularmente en nuestro subsistema es diariamente vía Jenkins.

Una vez se haya comentado todo lo necesario sobre los entregables será hora de explicar cómo se deberán utilizar para desplegarlos con éxito. Ya que durante el cuatrimestre se han ido experimentando problemas en los talleres de integración, se decidió junto con el resto de los subsistemas una arquitectura común en la que desplegar el sistema. Introduciéndolo un poco, cada subsistema debe poder desplegarse de manera individual y debe cumplir todos sus los requisitos funcionales además de ofrecer y/o consumir, si fuera necesario, la funcionalidad de los demás subsistemas con los que se relaciona.

Como se ha comentado antes, durante la asignatura han ocurrido varias incidencias, por eso en el punto “Gestión de incidencias y depuración” hablaremos sobre cómo nuestro grupo las ha gestionado como subgrupo independiente y con los demás grupos además de qué métodos de depuración hemos utilizado para solventarlos. En este punto se especifican cómo hemos cubierto las posibles incidencias que han ido ocurriendo a lo largo del desarrollo de las funcionalidades, qué mecanismos hemos llevado a cabo para abordar los problemas y las políticas que hemos estado usando para fomentar o retardar un cambio específico.

Finalizando el documento se encontrará la gestión de la variabilidad en la que se podrá informar de los mecanismos que hemos propuesto para Ágora Voting ya que éste sólo tiene la versión web por lo que la variabilidad del sistema es escasa actualmente. Podremos ver un ejemplo de variabilidad que nuestro grupo añade a Ágora Voting y unos mecanismos a seguir para introducir algún cambio, funcionalidad o versión en el sistema.

Para finalizar podrá ver el mapa de las herramientas relacionadas que usamos en nuestro sistema para que funcione correctamente con los demás subgrupos así como una explicación detallada del mismo.

Por último existe una conclusión sobre este documento y trabajo con el que pretendemos aclarar nuestro punto de vista en este proyecto.

¡¡¡¡citad las fuentes de donde saquéis la info!!!

2. Gestión del código fuente

La finalidad de una buena gestión del código fuente en un proyecto de software en el que trabajan varios desarrolladores son las siguientes:

- Que todo el código del proyecto esté accesible y disponible por todos los desarrolladores implicados.
- Seguir una línea temporal en la que se pueda observar todos los estados por los que ha pasado un fichero a lo largo del desarrollo, pudiendo regresar a una versión anterior facilmente.
- Gestionar distintas variantes del código fácilmente.
- Unir variantes, si las hay, de una forma eficiente y rápida.
- Identificar quién hace cada cambio.
- Mejorar la productividad y la calidad del software final.

Para permitir todo esto la gestión del código fuente de ser ágil y fácil de llevar a cabo.

La primera herramienta que usamos para el desarrollo de nuestro proyecto es una herramienta de control de versiones con bastante popularidad llamada *Subversion* [2]. Decidimos elegir dicha herramienta debido a que ya teníamos bastante manejo en la misma, ya que la habíamos usado en proyectos anteriores, y nos aporta comodidad y seguridad.

Esta solución no cumplia con todos los requisitos que una buena gestión del código necesita.

En esta etapa la gestión era caótica. Todos los componentes del grupo tenían acceso al código, fueran o no desarrolladores, no había una buena política para la creación o unión de ramas ni teníamos conocimientos para realizarlo eficientemente, etc.

Más adelante cambiamos nuestra herramienta para el control de versiones por *Git*[3], mediante un repositorio de código en la nube que proporciona el servicio web *GitHub*[4]. Este cambio fue motivado por dos argumentos esenciales: el contenido de la asignatura nos llevó a estudiar dichas herramientas y la comunicación con otro de los

[2] Subversion (Apache): https://subversion.apache.org/
[3] Git: http://git-scm.com/
[4] Github: https://github.com/

Hubo algún motivo más "técnico" en el cambio? ¿pensáis que se podría haber seguido trabajando con Subversion?

subgrupos, con el que compartimos ciertas partes de código, la cual se llevó mucho mejor de esta manera, con una herramienta conocida y utilizada por todos por todos.

Lo que hicimos es traspasar el código que teníamos en el antiguo repositorio (SVN) al nuevo (GIT) pero siguiendo la misma política del anterior trayéndonos, además del código, todas las malas prácticas. Por lo tanto seguíamos teniendo los mismo problemas.

Por último, en consenso con todos los demás grupos que componen el sistema de votaciones y tomando la iniciativa de uno de estos, se decidió crear un repositorio[5] común en el portal web *GitHub* y adoptar una política similar entre todos. A nuestro proyecto le corresponde la carpeta de *results*.

OJO, en el wiki no pone eso...;-)

Para poder utilizar este repositorio era necesario pedir la admisión en el foro de comunicación común de todos los grupos, donde deberás dejar claro tu usuario en *GitHub*. Específicamente en el foro *Decisiones* en el tema *Repositorio compartidos.*

2.1 Gestión de ramas

Una rama es una línea de código diferente dentro del repositorio común. Según la definición en *http://codehero.co:*

"En git cuando consolidamos cambios, creamos una fotografía del estado actual de la información que se encuentra en escenario, luego se descartan los archivos que no tienen modificaciones y se reverenciaban al estado anterior. Al verificar el estado anterior se comprueba quién es o son padres directos de esta fotografía. Es decir, se busca cuales son los cambios que fueron consolidados antes que este y se les referencia como el padre directo. Siguiendo este modelo, lo primero que consolidamos en el proyecto no tiene padre, los cambios consolidados sobre la rama principal tienen un único padre, y los cambios que provienen de la unión de una o más ramas tienen múltiples padres.

[5] Repositorio común: https://github.com/EGC-1415-Repositorio-compartido/repvoting

Entonces, retomando ¿Qué es una rama? Una rama es una extensión del árbol o tronco principal. Como buena práctica dentro de las ramas del árbol es donde deberíamos introducir los cambios a nuestro proyecto y solo luego de comprobar que dichos cambios funcionan y tienen el comportamiento deseado los unimos con el árbol principal. Esto es porque queremos que el árbol se encuentre lo más limpio posible."

La política de ramas que se usa para nuestro sistema de votación es la siguiente: tenemos una rama *master* donde solo pueden estar los proyectos sin errores, son versiones "entregables" de cada subsistema.

Esta es la propuesta que hizo el grupo de autentificación y la nosotros estamos siguiendo:

"Se crearán ramas según las funcionalidades que haya que implementar, creando una rama por cada nueva funcionalidad. En la rama *master* se realizarán los cambios comunes a todos los grupos, de ella se podrá tomar lo que se necesite de cada grupo. Además cada grupo tiene su propia carpeta creada en la rama *master* y de la que deberán crear una rama para trabajar. Las carpetas de cada grupo son las siguientes:

- auth: Autenticación
- votes: Creación/administración de votaciones
- results: Sistema de modificación de resultados
- storage: Almacenamiento de votos
- deliberations: Deliberaciones
- counting: Recuento
- census: Creación/Administración de censos
- results_frontend: Frontend de Resultados
- results_view: Visualización de resultados
- verification: Verificación
- polling: Cabina de votación

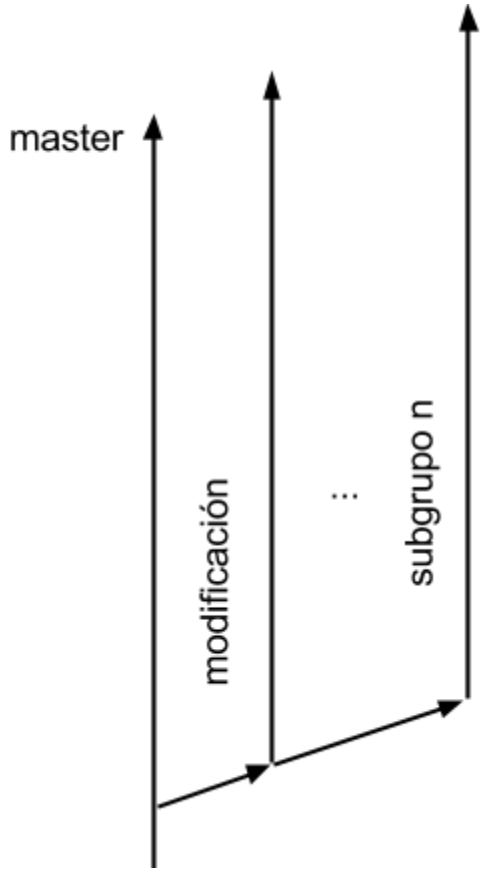
Si en algún momento en una rama hay código de utilidad perteneciente a otra rama, se hará un *merge*[6] de esta última en la rama que puede utilizar el código (sin borrar ninguna de las dos). Si hay algún conflicto no relacionado con el código que se quiere pasar de una rama a otra, permanecerá siempre el código de la rama original. De esta manera, se pretende evitar la repetición de trabajo cuando es menos obvio la utilidad de este para todo el grupo, de manera que no se haya usado la branch de cambios globales. Para que esto sea posible, se requiere que los miembros del equipo puedan identificar trabajo que es posible que ya haya realizado otro miembro del grupo. Se actuará de igual manera cuando dos funcionalidades estén relacionadas y una necesite a parte de la otra.

Se realizará un merge que incluya los cambios de una rama en la rama principal cuando se haya terminado la funcionalidad a la que está dedicada cada rama, cerrándose la rama dedicada a la funcionalidad. Con la excepción de la rama de cambios comunes, que nunca se cerrará, pero cuyos cambios se reflejarán siempre en las otras ramas.”

Y qué gestión sobre vuestro propio proyecto? sobre eso no comentáis mucho.

Está confuso qué se hace con los otros subsistemas y qué hacéis vosotr*s en vuestro caso concreto Se podría trabajar mejor este punto.

[6] Git merge: http://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging



BRANCHES

las figuras se deben numerar y referenciar. Se deben además explicar.

2.2 Gestión de roles

En la política común de los grupos no hay roles, ni definidos ni en la práctica. Como esta detallado en el repositorio:

"Todos los miembros del equipo tendrán derechos de escritura y lectura de todas las ramas."

Al ser un proyecto pequeño (en cuanto a líneas de código se refiere) no era necesario definir roles. A esto se le suma el número tan grande de personas que participaban, lo cual hacía muy difícil la decisión de poner a alguien encargado de los "pull request[7]" o peticiones de cambios en el repositorio.

[7] Pull request: https://help.github.com/articles/using-pull-requests/

No obstante esta no es la mejor práctica y que cualquiera que tenga acceso al repositorio podría hacer lo que quisiera con cualquier proyecto, aunque no pertenezca a él.

Para una buena gestión del código deberíamos tener mínimo los siguientes roles:

- Desarrollador: se encarga de desarrollar código fuente y testearlo. Cuando quiera enviar un cambio al repositorio este deberia hacer pull request.

- Desarrollador jefe: se encarga de llevar la organización de todos los elementos del código. Sería el encargado de aceptar o rechazar las peticiones de cambio de los desarrolladores.

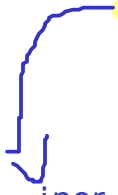
¿Por qué no se ha puesto en práctica? los argumentos no son convincentes.

2.3 Pautas para aplicar un patch (parche)

Esta parte viene explicada en el repositorio común, la pauta a seguir es la siguiente:

"Siempre que se realice un parche sobre el código (una modificación que no añada nuevas funcionalidades), la descripción de esta será:`BUGFIX:` seguido del bug que se pretende corregir, y la plataforma sobre la que se produzca dicho bug, en caso de que haya una concreta."

En cuanto a las políticas de commits[8] o de estilo del código comentar que no hay ninguna establecida. Todos seguimos más o menos las buenas prácticas aprendidas a los largo de los cursos anteriores. Como por ejemplo: no poner abreviaciones en los nombres de variables o métodos.



[8] Git commit: http://git-scm.com/book/es/v1/Fundamentos-de-Git-Guardando-cambios-en-el-repositorio

2.4 Ejercicio propuesto: aplicar un parche

Enunciado

Aplicar un parche en el código del subsistema de modificación de resultados y subirlo a la rama *master*. Imaginar que es la primera vez que vas a realizarlo, pero que si tenemos acceso al repositorio común como desarrollador.

Solución

Lo primero que haríamos es clonar la rama *"modificación"* del repositorio compartido mediante la siguiente instrucción:

```
$ git clone -b modificación --single-branch
git@github.com:EGC-1415-Repositorio-compartido/repvoting.git
```

Con el código necesario ya en nuestra máquina de desarrollo aplicamos el parche al código. Una vez terminado realizamos un *commit* cuyo título y mensaje debe de ser descriptivo con respecto a lo que se ha hecho.

```
$ git commit -a
```

Esto hará un *commit* de todos los archivos a los que ya "*git"* le siga pista, en el caso de que se crearán nuevos archivos habría que añadirlos antes de hacer el *commit* con la instrucción:

```
$ git add NOMBRE_ARCHIVO
```

Después del commit el siguiente paso es hacer un *push*, que consiste en llevar al repositorio online lo que hemos hecho en local. El *push* lo haremos a la rama *modificación* que es donde hemos hecho los cambios.

```
$ git push origin modificación
```

El siguiente paso sería hacer un *merge* en la rama "*master"* de lo que hay en la rama "*modificación"*, que consiste en llevar lo que hay en la rama "*modificación"* a la rama *"master"*. Para esto usamos las instrucciones:

```
$ git checkout master
```

Esto nos posiciona en la rama *master* para que desde esta ejecutar:

```
$ git merge modificación
```

Esto fusionará ambas ramas. El comentario se escribirá con el siguiente formato: "`BUGFIX:` seguido del bug que se pretende corregir."

Por último enviaremos los cambios al servidor con esta instrucción:

```
$ git push origin master
```

Con esto ya tendríamos completado el ejercicio.

Se podría haber buscado un ejercicio más complejo o añadir más ejercicios.

3. Gestión de la construcción e integración continua

Para la gestión y construcción del proyecto hemos utilizado la herramienta de software Maven[9], basándose en un fichero de configuración en XML (project.xml) y una serie de extensiones (plugins), esta herramienta puede compilar el proyecto Java, ejecutar las pruebas unitarias, generar paquetes (jars, wars, ears o distribuciones en zip) y generar una serie de informes, es decir, esta herramienta construye, a partir de ficheros bases, un war de forma automática, teniendo la opción de instalar y desplegar dicho war.

Maven utiliza un Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.

En nuestro proyecto hemos partido del POM creado en la asignatura 'Diseño y Pruebas' añadiendo nuevas dependencias necesarias para la creación de nuestro proyecto en este caso jackson-mapper-asl* y json*.

¿detalles?

Las partes del ciclo de vida principal de nuestro proyecto Maven son:

1. *compile*: Genera los ficheros .class compilando los fuentes .java

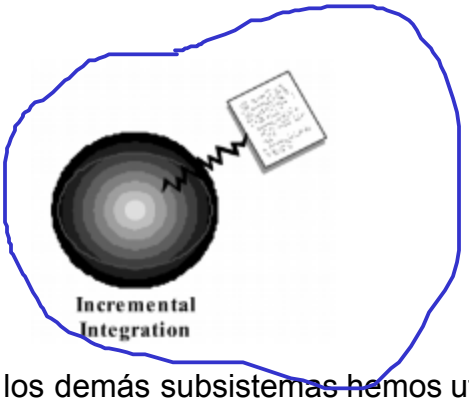
2. *package*: Genera el fichero .war con los .class compilados

En cuanto a integración, hemos usado dos de los tipos estudiados en clase, diferenciándose el usado internamente en nuestro subsistema y el utilizado en conjunto con los demás subsistemas:

- Para la integración en nuestro subsistema hemos utilizado el tipo 'Integración incremental' en el que desarrollamos un esqueleto principal de nuestra aplicación y vamos añadiendo nuevas partes del proyecto probadas y depuradas.

[9] Maven: http://maven.apache.org/

citar las fuentes!!



- Para la integración con los demás subsistemas hemos utilizado 'Integración por fases' en la que cada subsistema trabaja de manera independiente a los demás y periódicamente se combinan todos los subsistemas.

Mediante integración continua tenemos la posibilidad de realizar el proceso de construcción, la construcción de nuestros propios tests y más operaciones, todo de forma automática.

Mediante el uso de la herramienta que nos permite la realización del proceso, configuramos la misma, para que se conecte al repositorio de código usado por el grupo y realice el proceso de construcción.

El proceso consiste en el uso de una herramienta específica, la cual configuramos para que se conecte al repositorio de código de nuestro equipo, detectando los cambios que se producen en el mismo y realizando las operaciones y procesos automáticos que tenga asignados.

Podemos determinar que el proceso comienza con la detección, por parte de la herramienta, de un cambio (commit) en el repositorio de código, en nuestro caso Git. Una vez detectado, la propia herramienta se encarga de ejecutar el script de construcción.

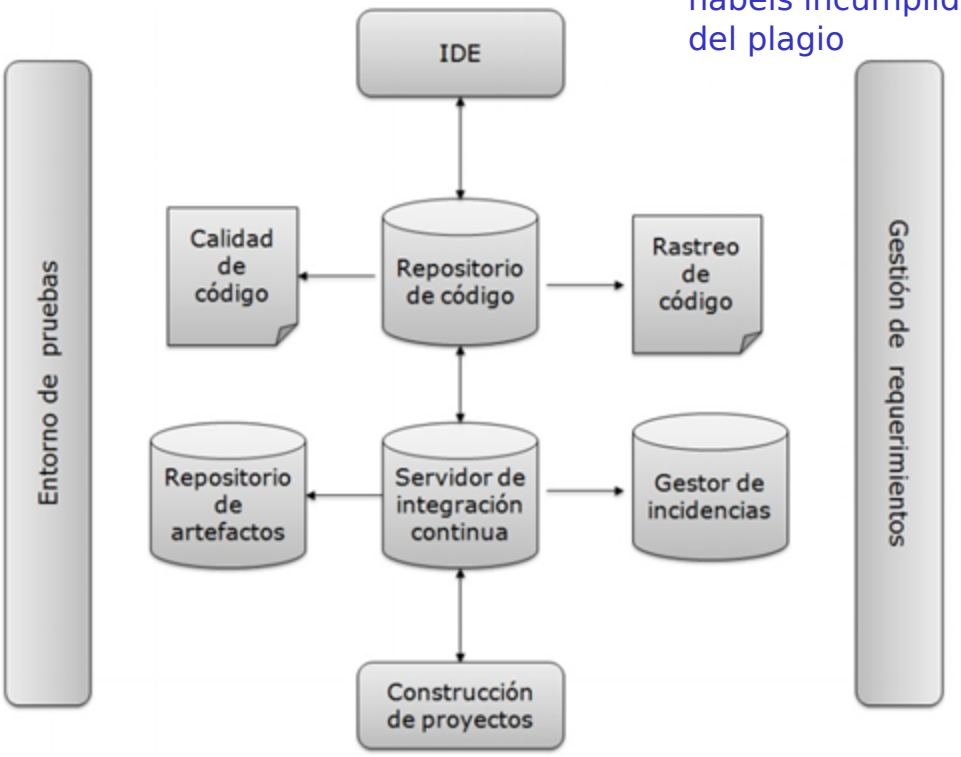
Una vez procesado el script, la herramienta guarda los resultados del mismo mediante un mecanismo de feedback y tiene la capacidad de generar alertas para mantener informados a los desarrolladores, de forma inmediata, de posibles errores.

En un caso ideal se utilizan construcciones de integración, en las cuales se dispone de un servidor dedicado encargado de la misma. El equipo encargado de nuestro subsistema decide simular este entorno mediante construcciones privadas, las cuales, a diferencia de las anteriores, se realizan en nuestras propias máquinas locales.

En la siguiente imagen podemos ver cómo sería la arquitectura de un entorno en el que se usan construcciones de integración.

3.1 Arquitectura

Cita!!! Siento comentaros que habéis incumplido el tema del plagio



Uno de los servicios más atractivos es la posibilidad de programar nuestras construcciones, de forma diaria por ejemplo. De esta manera nos ayuda a realizar un mejor mantenimiento del código, a encontrar bugs [10]y a muchas más tareas.

[10] Bug: http://es.wikipedia.org/wiki/Error_de_software

El uso de integración continua nos beneficia en la gestión del proyecto, mejorando la calidad de trabajo de nuestro equipo; la gestión del código, apoyándonos en la detección de errores en el proyecto; el proceso de construcción, como hemos expresado en este apartado.

Aunque no sea una tarea liviana, pensamos que merece la pena invertir tiempo en ella, ya que la inversión traerá posteriores beneficios.
Para el desarrollo de la integración continua existen numerosas herramientas de las cuales decidimos usar Jenkins para nuestro proyecto. Jenkins es una herramienta de código abierto muy popular, por lo que dispone de bastante soporte y ha sido tratada dentro del contenido de la asignatura, por lo que decidimos que sería la mejor opción.

Podemos descargarla y obtener información de la misma en la siguiente url:
http://jenkins-ci.org/

Tras nuestra experiencia en integración hemos concluido que es un proceso que se debe llevar a cabo desde el primer momento y que es fundamental.

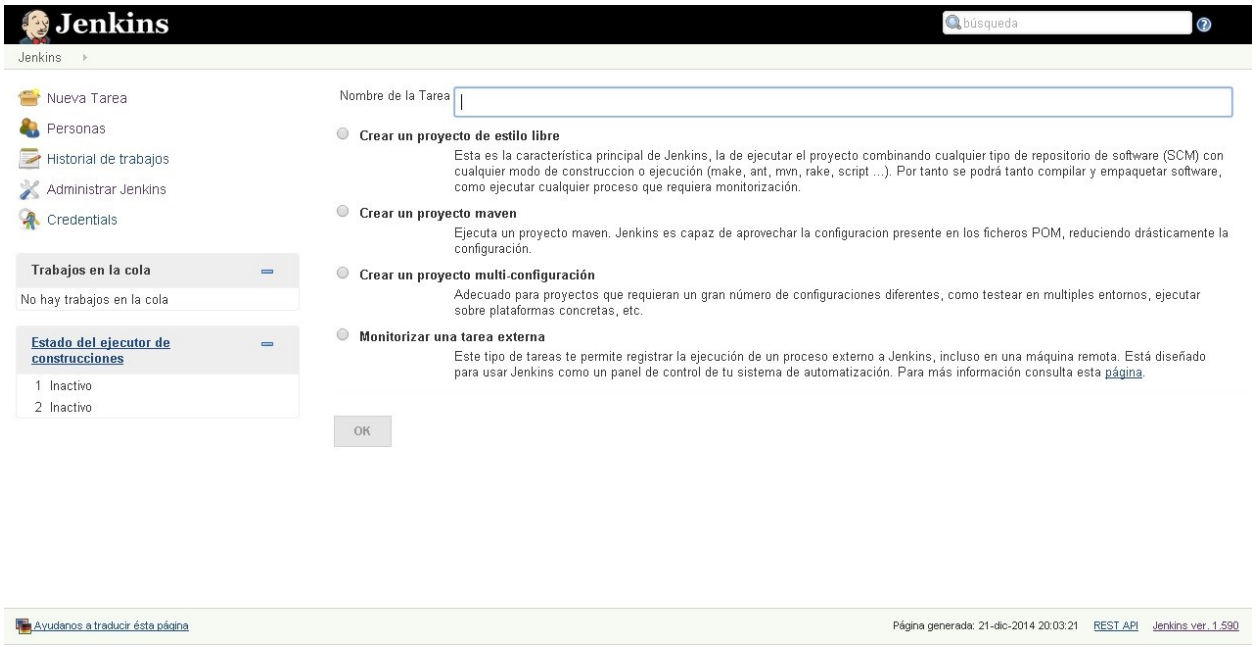
3.2 Ejercicio propuesto: Manejando Jenkins

En este ejemplo vamos a reflejar el proceso básico de creación de tareas de construcción.

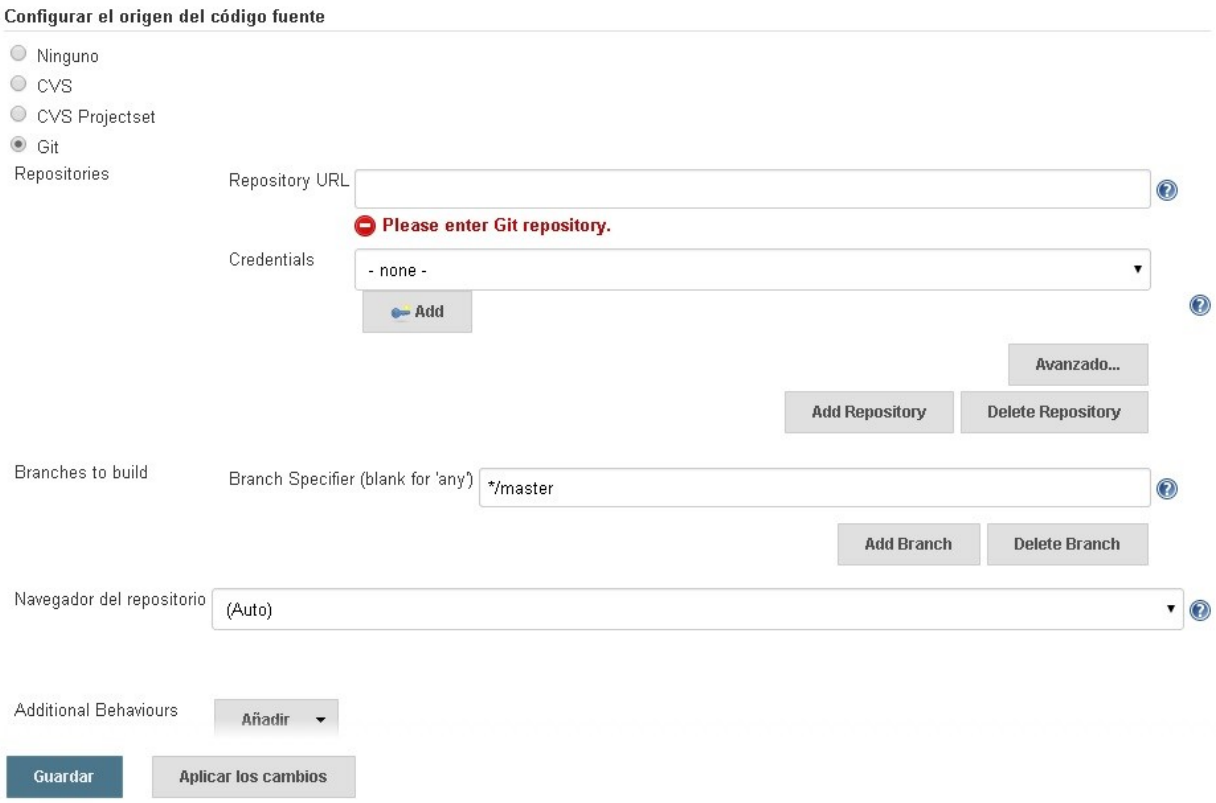
Para resolver este ejercicio realizaremos los siguientes pasos:

En primer lugar abriremos Jenkins y veremos en el menú de la izquierda, en la primera opción, el enlace para crear una nueva tarea.

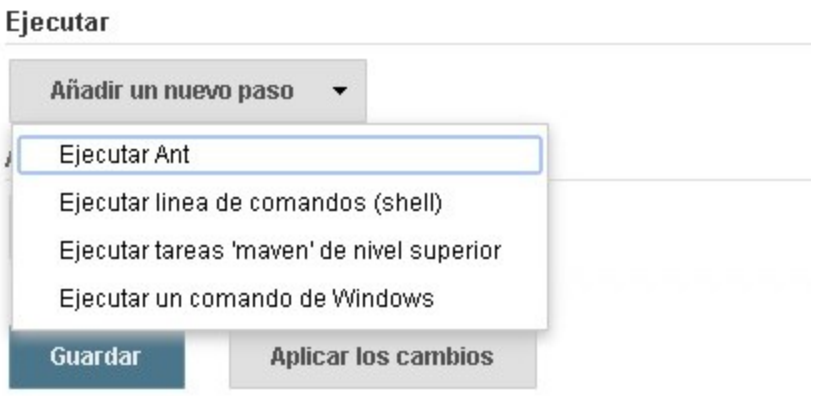
Daremos un nombre a dicha tarea y decidiremos el tipo de proyecto o tarea que queremos instanciar.



Una vez pulsemos en el botón “ok” Jenkins nos dirigirá al menú de configuración del proyecto tarea. Es en este menú donde debemos configurar nuestro repositorio introduciendo la URL del mismo.



En el mismo menú podemos configurar el tipo de ejecución que queremos en la pestaña “Ejecutar”.



Seguimos en configuración y en la pestaña “Disparadores de ejecuciones” podemos añadir periodicidad a nuestras ejecuciones.

De esta manera tendremos configurada nuestra tarea y en la página principal de Jenkins dispondremos de un menú que contendrá información sobre la misma y un código basado en el tiempo atmosférico para representar el estado de dicha tarea.

En nuestro proyecto hemos desarrollado un script de construcción el cual ejecutamos desde una tarea de Jenkins de forma periódica.

Tras nuestra experiencia en integración hemos concluido que es un proceso que se debe llevar a cabo desde el primer momento y que es fundamental.

4. Gestión del cambio

El proceso que hemos ido siguiendo hasta ahora no estaba predefinido por un estándar que teníamos que seguir debido a la falta de información y conocimiento de los miembros del grupo; aunque sin saberlo estábamos siguiendo ya unos mecanismos de depuración.

Estos mecanismos de depuración ya mencionados y utilizados por el grupo son bastante simples, debido al bajo volumen de código con el que debemos trabajar, consisten en la siguiente secuencia de pasos a seguir cada vez que se produce una incidencia o se detecta un fallo:

- Informar de la incidencia
- Reproducir
- Diagnosticar
- Arreglar
- Analizar

A continuación, los iremos explicando cada uno detenidamente:

- Informar de la incidencia: Cada vez que se detecte una incidencia de código difícil de resolver, un cambio en la estructura, o de cualquier otro tipo que sea relevante, debe ser comunicada al resto del grupo mediante la/s vía/s de comunicación grupal ya acordadas,es decir por grupo de Whatsapp y/o Google Drive, que estime oportuna según la importancia de dicha incidencia, haciendo un comunicado en detalle para un posterior análisis por parte de la sección de desarrolladores, que en nuestro caso, como todos tenemos un buen nivel de programación, podemos entender de qué trata dicha incidencia y tomar parte en dicho análisis.

- Reproducir: Para reproducir la incidencia, como ya hemos dicho debido al poco volumen de código con el que trabajamos, nos centramos sobre todo en errores críticos dentro de la ejecución del código o en las interfaces usadas para la integración con el código de otros subgrupos.
  Estos errores son reproducidos dentro del entorno de desarrollo Eclipse en la máquina de cada desarrollador, donde no debería haber diferencias en los fallos.

Dicho entorno nos ofrece además acceso a la herramienta de debug y a la consola de java, que nos facilita información más detallada a la hora de reproducir fallos.

Aparte cabe destacar que adoptamos la política de no incluir un logging dentro del código, debido a los siguientes argumentos:

- Ensucia el código innecesariamente, debido a que tiene poca utilidad frente a la complicación de implantarlo,consultarlo y mantenerlo sin fallos.
- No tenemos tanto volumen de código como para perder la sucesión de actualizaciones al gestor de versiones.
- El gestor de versiones utilizado permite regresión a estados anteriores e información y descripciones de los cambios realizados en cada versión.
- Puede dar los mismos problemas que los comentarios.
- Y debido a la famosa conjetura de: "*No importa la cantidad de información de log*[11] *que añadas, nunca será la que necesitas" de versiones*"

Esta cita viene a decir, que por mucha información que incluyamos en el logging del sistema, es muy difícil que coincida con la necesitada, bien o porque tenemos muy poca y no mostramos lo suficiente, o que si no tendremos mucha y costará discernir bien entre tanta información, dando lugar a no encontrar la parte necesaria, o bien porque que la misma naturaleza del problema básicamente hace que no pueda ser registrado en un sistema de logging.

- Diagnosticar: A la hora de diagnosticar, el desarrollador o equipo de desarrolladores designado/s a dicha parte del código, deberá realizar una serie de experimentos, de uno en uno, según las hipótesis creadas por la entidad encargada del diagnóstico y si estas no resultasen acertadamente, pueden pedir ayuda al resto del equipo indistintamente de su función en el proyecto, debido a, como ya hemos dicho previamente, la buena formación de todas las personas pertenecientes al grupo en temas de programación.

[11] Log: http://es.wikipedia.org/wiki/Log_%28registro%29

- Arreglar: En este aspecto no tenemos un patrón muy formal del estándar a seguir,es decir, como no tenemos preproducción, ni una lista de artículos de trabajo, no es una acción con tanto alcance.
  Debido a ello utilizamos un patrón bastante informal, asumimos que tras la detección del bug que da lugar al fallo de la incidencia, se arreglará el estado de error y se subirá al repositorio con una descripción detallada asociada a la nueva versión creada, esto si fuese una incidencia leve, si fuese algo de importancia, de cambio de estructura o de algún otro tipo debe ser analizado previamente por el equipo programadores, como ya explicamos en la etapa de reporte de incidencias.

- Analizar: Después de localizar el bug y arreglarlo, suele ir acompañado de una conversación entre los miembros del grupo explicando lo sucedido al resto del equipo,un debate abierto y la conclusión obtenida en la experiencia.

En nuestro proyecto la gestión de cambio ha sido bastante importante debido a la multitud de cambios estructurales que hemos tenido que hacer en el código.

Hemos utilizado un enfoque dinámico para poder afrontar todos los cambios sin problemas, al tener que adaptar el proyecto varias veces según los diferentes requisitos de integración y de funcionalidad.

4.1 Procesos de la gestión de cambios

- Monitorización del proceso de cambio: hemos ido utilizando un gestor de versiones añadiendo los motivos de los cambios que hemos ido realizando y mediante reuniones para tratar los diferentes aspectos del cambio.
- Evaluar peticiones: La evaluación depende de un consenso de todo el grupo para aceptar peticiones.
- Reuniones: en las reuniones se tratarán los temas con todo el equipo de desarrolladores
- Implementación: Se encargará a varios programadores la aplicación de los cambios acordados previamente por el grupo.

4.2 Roles

En cada informe de la gestión del cambio, cada uno tiene un rol definido según su forma de intervención en el procesos, los que propongan las modificaciones serán los propietarios del cambio y el resto representa la función de consejo de cambio, que aceptaran los cambios propuestos o los rechazan.

4.3 Estados

Utilizamos unos términos especiales para los estados cada vez que se crea una petición de cambio:

- Cambio propuesto, cuando se propone un cambio a implementar
- Estudio del cambio, el consejo estudia la situación y la viabilidad de dicho cambio
- Adaptación de elementos externos, se va preparando el código que tenga también que ser modificado debido a acciones colaterales.
- Implementación, se implementa en su totalidad en cambio propuesto.

4.4 Políticas

Dentro del proyecto hemos utilizado una política democrática para el consejo de cambio, es decir un debate grupal estudiando las opciones viables, y luego mediante una votación donde interceden todos los miembros del grupo para ver si el cambio será aceptado,aplazado o si no será rechazado y no se aplicará.

4.5 Ejercicio propuesto: petición de cambio

- Petición de cambio: fecha 2/12/2014
- Autor: anónimo

Descripción: ha vuelto a cambiar el formato JSON asignado a los votos, debemos de reestructurar el proyecto actualizando los campos requeridos de la estructura a recibir para la integración los servicios.

- Prioridad:Urgente

Consejo de cambio:

1. Se produce una reunión,con una asistencia del 90% del grupo,luego tienen permiso para la toma de decisiones referente al cambio y se genera una acta de reunión.
2. En la reunión se debate el cambio y si es aceptado,aplazado o denegado.
3. Se evalúa el cambio y es admitido por una mayoría a favor .
4. Se procede a desarrollar la estructura para la futura implementación del cambio.
5. Se acuerda que miembros deben ocuparse de dicha implementación.
6. Se cierra la reunión.

Equipo de desarrollo:
Implementa la solución estipulada por el consejo de cambio, mientras
es monitorizado por este para saber el estado de la gestión del cambio.

Finalmente, el cambio ha sido implementado satisfactoriamente y se cierra
el proceso de gestión del cambio propuesto a dicha petición inicial.

5. Gestión del despliegue

Con respecto a la gestión del despliegue, según lo visto en clase y las vivencias que hemos tenido durante el desarrollo de la asignatura y los correspondientes talleres de integración, decidimos hablar en esta sección sobre cómo integrar todos los subsistemas desarrollados por cada uno de los grupos de la asignatura, debido al campo tan reducido que tienen cada uno de los subsistemas y también haberle preguntado al profesor su opinión en este aspecto.

Cuando hablamos de cómo integrarlos nos referimos a la manera en la cual, cada módulo del proyecto Ágora Voting se van a unir, para conseguir un correcto funcionamiento del mismo.

Después de los intentos fallidos o sin mucho éxito de integración, llegamos a la conclusión de desplegar la mayor parte de los subsistemas en una máquina virtual de ubuntu con las herramientas necesarias para que todos los proyectos de los grupos de la asignatura puedan desplegarse sin ningún tipo de problemas.

Esta decisión se tomó ya que al intentar integrar varios subsistemas a la vez, en la máquina virtual de windows XP, la cual todos los grupos usaban, el funcionamiento no era ni mucho menos el esperado, ya que no funcionaba cuando se introducían más de 3 proyectos dentro de la máquina.

El sistema elegido nos sirve para tener un integración mucho más centrada y óptima ya que al tener todas las herramientas en una misma máquina, bajo consenso de todos los grupos, no se hace tan difícil el hecho de unirse unos con otros.

El objetivo principal de esta decisión es el despliegue de Ágora Voting, ya que hablar de despliegue de un solo módulo del proyecto es por lo tanto todos los proyectos de los subsistemas relacionados tendrán que estar preparados para desplegarse de manera independiente y tras importar las librerías adecuadas, funcionar correctamente, comunicándose con los subsistemas relacionados.

Para conseguir este funcionamiento y facilitarlo se decidió usar la misma autenticación en las bases de datos.

Con respecto a las herramientas citadas anteriormente, son las necesarias para asegurar el correcto funcionamiento, como ejemplo: Git, Xampp, Eclipse (adaptado para proyectos con Maven) y Jenkins.

6. Gestión de la variabilidad

La gestión de la variabilidad busca que el software sea personalizable y que a su vez sea producido en masa para que el producto se ajuste a cada tipo de cliente lo más fácilmente posible.

Es un nuevo paradigma que ha aparecido dentro del desarrollo de software y se basa en SPL [12](Software Product line) o línea de producto software que consiste en tomar uno o más artefactos realizados como parte de un desarrollo y utilizarlos nuevamente en el desarrollo de otro sistema para la creación de un nuevo producto enfocado a distintos clientes, usuarios u objetivos. Este nuevo sistema es desarrollado a partir de un conjunto común de bienes del núcleo (core assets). Los bienes del núcleo son la base de la línea de productos e incluyen entre otros la arquitectura, componentes reutilizables, modelos de dominio, requerimientos, documentación, planes de prueba, etc. Un aspecto importante a considerar dentro de la línea de productos es que se debe establecer un alcance en donde se describe qué productos son parte de la línea.

Al partir siempre de un mismo sistema, estos nuevos productos siempre tendrían la misma base pero con características especiales o distintas que los diferencian unos de otros. Si tenemos en cuenta este método, Ágora Voting podría desplegar nuevas versiones distintas de su aplicación web, como por ejemplo una aplicación oficial para móvil o tablet en Android, iOS o Windows Phone así como distintas aplicaciones para distintos clientes tales como Ágora Voting Enterprise, Ágora Voting developer, etc.

Por ahora no se conoce otra versión de Ágora Voting, sólo la versión web, por lo que la variabilidad que tiene ahora mismo el sistema es escasa y está enfocada para un único tipo de usuario.

[12] SPL: http://en.wikipedia.org/wiki/Software_product_line

Para que Ágora Voting pudiera modelar correctamente la futura variabilidad que se incluiría en el sistema, habría que definir primeramente los “core assets” que serían útiles para su uso, con el fin de utilizar el paradigma SPL. Para ello hay que utilizar un modelo de variabilidad que consiste en un diagrama con los siguientes elementos:

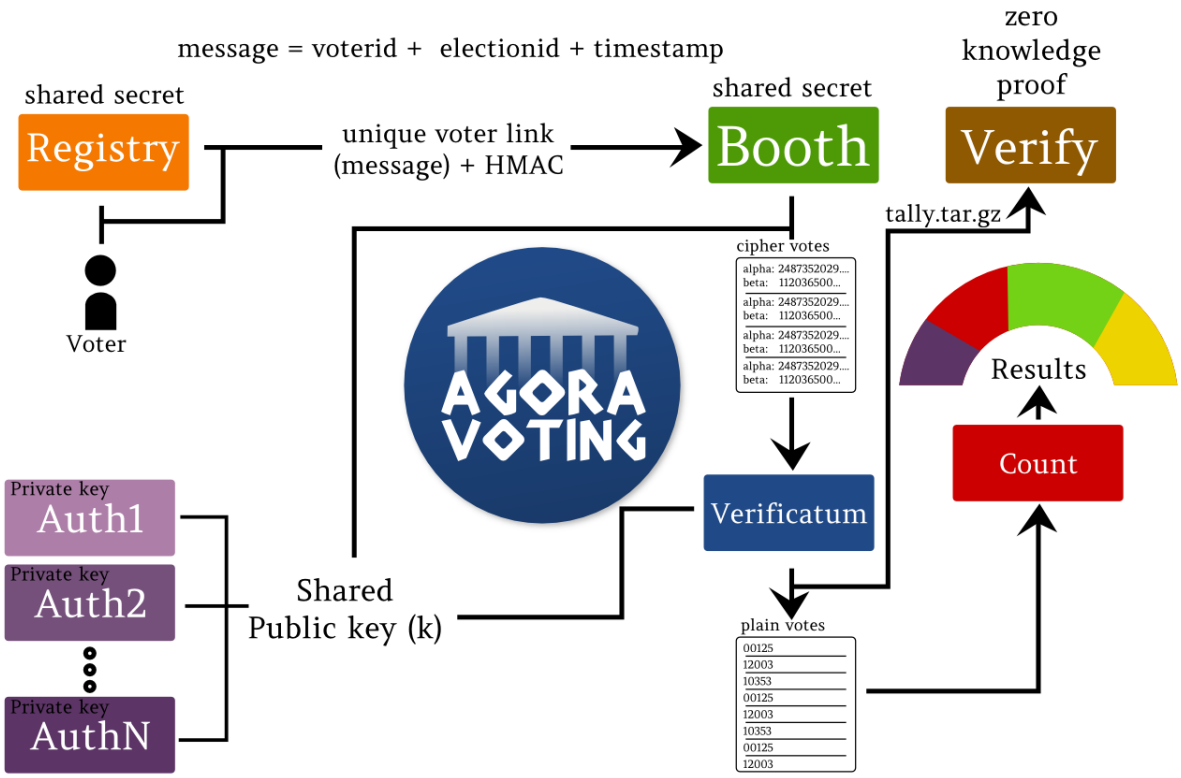
1. Identificar qué productos son parte de la línea base
2. Identificar la variabilidad a implementar
3. Definir sus posibles restricciones
4. Implementar la variabilidad
5. Administrar la variabilidad implementada

Con respecto a la futura variabilidad de Ágora Voting se han encontrado pocos resultados ya que hay poca documentación al respecto. Se espera que en fechas futuras, Ágora Voting se prepare para ser una plataforma puntera en el sector de votaciones, por ejemplo, para partidos políticos de cara a las elecciones primarias por lo que lanzarán una nueva versión en la que tendremos acceso desde cualquier dispositivo (ordenador, tablet, móvil), que sea accesible para que todos los usuarios puedan llegar a entender, usar y desarrollar.

Con respecto a los niveles en donde se gestiona la variabilidad, se espera que sean a nivel general partiendo desde una base, empezando en el código fuente y terminando en el despliegue del sistema.

Para el ejemplo propuesto anteriormente (Ágora Voting Enterprise, Ágora Voting developer, etc) la gestión del código para las distintas versiones se llevarán a cabo partiendo de la base de la versión web, haciendo los cambios pertinentes en el código para poder producir esa versión. Con respecto al despliegue, es exactamente lo mismo, tenemos que partir del despliegue de la base y enfocarlo a las nuevas versiones, haciendo los cambios que sean necesarios.

6.1 Core asset o núcleo de Ágora Voting

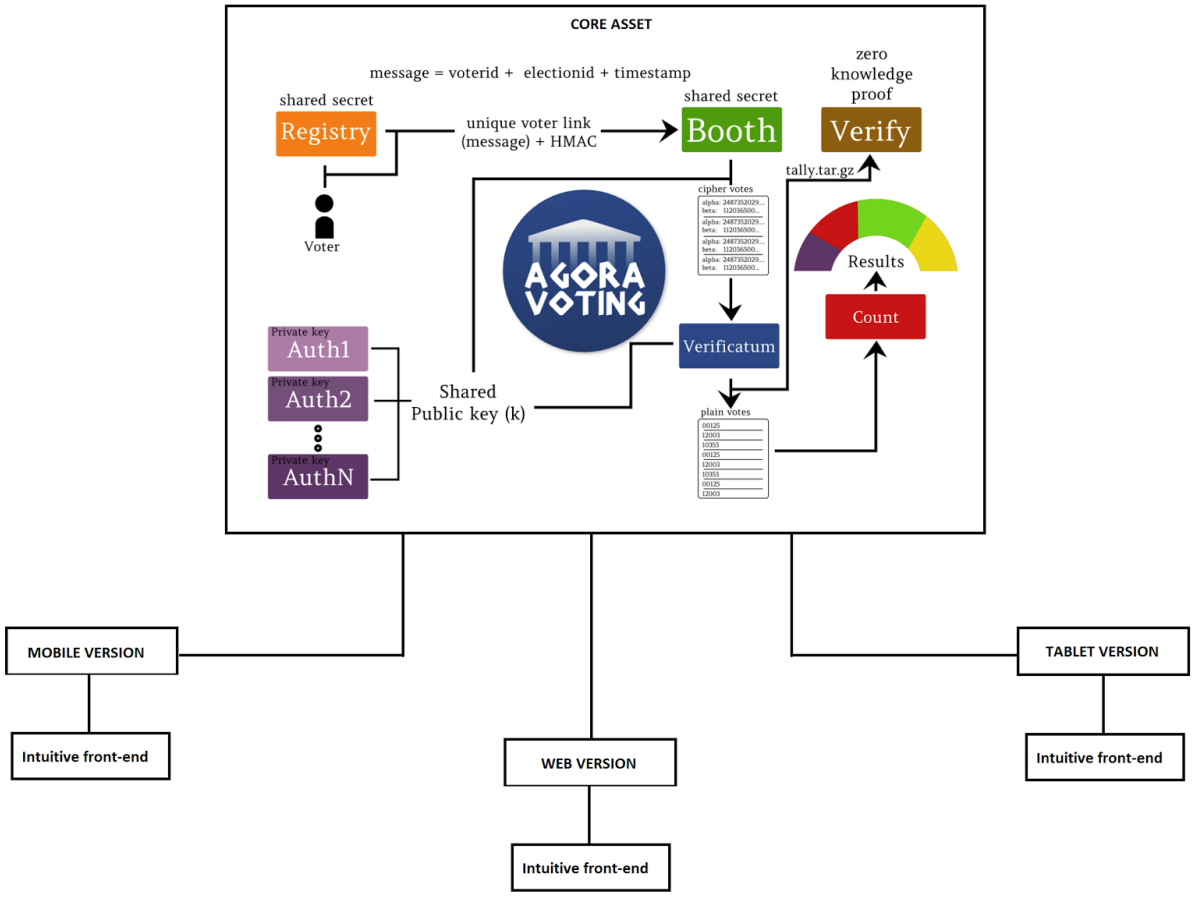


Esta imagen que vemos contiene el núcleo o la base de Ágora Voting y los subsistemas que componen el sistema o aplicación web. Cada subsistema está directamente relacionado con otro subsistema que necesita información de otro para llevar a cabo su función. La aplicación no podría funcionar si le quitamos alguna funcionalidad a cualquier subsistema ya que pertenecen a la base o núcleo.

Para llevar a cabo la variabilidad del sistema, tendríamos que partir de esta base y añadir mejoras u otras funcionalidades al sistema que no alteren el funcionamiento del núcleo.

6.2 Ejercicio propuesto: variabilidad para Ágora Voting

A continuación mostramos una propuesta de variabilidad simple para Ágora Voting:



En esta imagen podemos ver una propuesta muy simple de variabilidad para Ágora Voting que nos muestra dos nuevas versiones de aplicación (móvil y tablet) partiendo de la que ya existe (core asset) en las que el único requisito que tienen que tener estas versiones es que tengan una interfaz gráfica (front-end) intuitiva para que cualquier usuario que quiera acceder a la aplicación le resulte fácil usarla y no tenga problemas.

7. Mapa de herramientas

Lista de las herramientas usadas en nuestro subsistema:

- **Sistema operativo linux:** Concretamente Ubuntu 12.04.

- **Java JDK 7:** Es el lenguaje de programación que hemos utilizado para el desarrollo de nuestro subsistema.

- **ProjEtsii:** Sistema de gestión de proyectos usado para la creación del primer repositorio del grupo con SVN.

- **SVN:** Se utilizó como primera herramienta de gestión de versiones, y se alojó el repositorio en ProjEtsii, herramienta de gestión de proyectos comentada anteriormente.

- **Git 1.9.1 :** Hemos utilizado este sistema de control de versiones, ya que ha sido uno de los temas impartidos en la asignatura.
  Concretamente se han usado dos repositorios, uno en el cual hemos desarrollado nuestro subsistema, y el segundo, en el cual se han subido todos los proyectos de los subsistemas correspondientes a AGORA VOTING.

- **GitHub:** Aplicación Web donde se han alojado los dos repositorios de los que hemos hablado anteriormente.

- **IDE Eclipse Luna :** Entorno de desarrollo en el cual hemos unido todas las herramientas para la implementación del subsistema.

- **Tomcat 7:** Hemos elegido esta herramienta debido al conocimiento que teníamos sobre ella. Usada para desplegar nuestro subsistema.

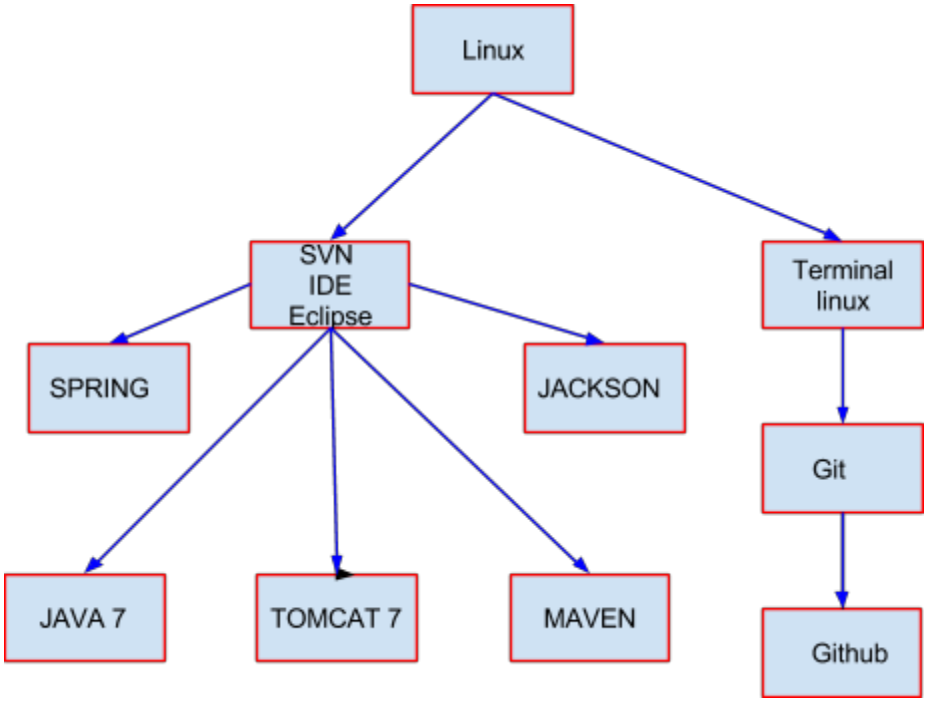
- **Terminal de linux:** Se ha utilizado para temas de configuración y de administración, concretamente para la gestión del cliente Git.

- **Maven:** Herramienta que nos da la funcionalidad para construir un proyecto en java.

- **Spring:** Framework utilizado para la ejecución de nuestro subsistema.

- **Jackson** : Librería en la que transformamos desde JSON a objetos Java, se ha usado principalmente para la integración con los subsistemas correspondientes.

7.1 Gestión del mapa de herramientas



En esta imagen mostramos cómo se relacionan las herramientas que hemos utilizado para realizar nuestro subsistema.

En primer lugar vemos la relación entre el sistema operativo utilizado, con su terminal y el entorno de desarrollo elegido.

El terminal de linux lo usamos para interactuar con el cliente de git y poder configurar de manera óptima nuestro código.

Con respecto al entorno de desarrollo vemos las relaciones entre las herramientas y cómo se han integrado cada una de ellas en nuestro eclipse.

Como se puede observar faltan dos de las herramientas nombradas en la lista, ya que estas herramientas (ProjEtsii y SVN) han sido usadas al principio de la implementación del proyecto, pero posteriormente se cambió de sistema de control de versiones.

8. Conclusiones

En esta memoria se presenta el análisis del subsistema de modificación de resultados de Agora Voting (que nuestro grupo ha tenido que desarrollar) desde el punto de vista de la asignatura de Evolución y Gestión de la Configuración. Por tanto se han estudiado los puntos de gestión del código fuente, gestión de la construcción e integración continua, gestión de los entregables, gestión del despliegue…

Durante el desarrollo de nuestro subsistema nos hemos dado cuenta que en un grupo como el nuestro de 9 desarrolladores es necesario llevar a cabo una buena gestión de la configuración. Pues aunque los miembros del equipo sean muy buenos en sus respectivas áreas sin una buena gestión el proyecto puede volverse caótico y difícil de desarrollar, así como una gestión mal aplicada puede incluso empeorar dicho proyecto.

Aunque es cierto que el llevar a cabo estas gestiones implica dedicarle más tiempo al proyecto sobre todo al inicio de este, a la larga el tiempo dedicado al desarrollo del código se verá reducido notablemente. Son unas buenas prácticas que debería de llevarse a cabo tanto en grandes como en pequeños proyectos.