E.T.S Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería de Software

Documentum e Integración Continua

Autor:
Dña.Tania Marciel Mateos

Tutor:
D. Benjamín Sahelices Fernández

Agradecimientos

A mis padres Darío y Concepción, por hacer posible mi formación como ingeniera informática. Ellos junto a mis hermanos han sido un gran apoyo.

A Ramón, por compartir junto a él las alegrías y tristezas que conlleva el esfuerzo de sacarse este tipo de carreras.

A mi tutor Benjamín, por haberme guiado y aconsejado todo este último año.

A mis compañeros más íntimos con los que he llegado hasta la meta y de los que me llevo una gran amistad.

A mis compañeros de Everis por haberme apoyado estos últimos meses a pesar de la brevedad de nuestra relación.

Gracias.

RESUMEN

En el contexto de la gestión documental, se ha implantado el proceso de integración continua sobre un proyecto de desarrollo software. La plataforma de gestión documental sobre la que se desarrolla el proyecto es *Documentum,* por lo que inicialmente se ha realizado un estudio de su arquitectura y de las herramientas empleadas que se integran en esta plataforma, como son: *Composer*, *Documentum Foundation Classes(DFC)*, *Web Development Kit (WDK)* y *Webtop*.

Una vez situados en el ámbito de desarrollo en el que nos movemos comenzamos el estudio y el análisis de la integración continua. Este proceso implica el uso de varias herramientas software, sobre las que se ha detallado su estructura y sus funcionalidades, las cuales permiten mantener la organización y minimizar el tiempo de desarrollo. Concretamente se ha empleado *SubversionSVN* como sistema de control de versiones*, Maven* y *Ant* como herramientas de automatización de tareas y *Jenkins* como servidor. Actualmente hay diversos servidores dedicados a la integración continua y por ello se ha realizado un análisis de algunos de los más populares en la actualidad, junto a Jenkins, con el fin de conocer las competencias que existen.

A continuación es necesario adaptar el servidor Jenkins a las especificaciones de nuestro trabajo, es decir, proceder a la instalación de los plugins necesarios y configurar los credenciales. Cada componente que forma el software del proyecto se corresponde con un job o tarea en Jenkins. Todos los jobs se han configurado para llevar a cabo la construcción, ejecución y despliegue de los componentes en particular y con ello, la integración y ejecución del proyecto en general.

Finalmente se pretende optimizar la calidad del código. Partiendo de los resultados obtenidos en el análisis de calidad realizado por *Sonar*, se procede a la corrección del código y a la revisión de los nuevos resultados.

ABSTRACT

In the context of document management, it has been implemented the continuous integration process on a software development project.

The document management platform on which the project is focused is Documentum, which initially carried out a study of its architecture and the tools used are integrated into this platform, such as: *Composer, Documentum Foundation Clases(DFC), Web Developement Kit(WDK)* and *Webtop*.

Once located in the area of development in which we move in starts the study and analysis of continuous integration. This process involves the use of various software tools, on which detailed structure and its functions, which allow to maintain the organization and minimize development time. Specifically it has been used Subversion SVN as version control system, Maven and Ant as task automation tools and Jenkins as a server.

There are currently several servers dedicated to continuous integration and therefore it has been performed an analysis of some of the most popular today, with Jenkins, in order to learn the existing competences.

Then is necesary to adapt the Jenkins server to the specifications of our work, that is, proceed to install the necessary plugins and configure credentials. Each component forming the software project corresponds to a job or task at Jenkins. All jobs are configured to carry out the construction, implementation and deployment of components in particular and thus, integration and project implementation in general.

Finally it aims to optimize the quality of the code. Starting from the results of quality analysis conducted by Sonar, it proceeds to correction code and to the review of new results.

Tabla de contenidos

Capítulo 1 1

1. Introducción 1
2. Objetivos. 1
3. Metodología. 2
4. Resumen de la memoria. 2
5. Planificación 3

Capítulo 2 4

1. Gestión documental. 4
2. Documentum 6.7 6
   - Arquitectura 8
   - Docbase y tipos documentales 12
   - Content Server 13
   - Documentum Composer 16
   - Documentum Foundation Classes (DFC) 17
   - Webtop 19
   - Web Development Kit (WDK) 20
3. Integración continua (IC) 21
4. Sistemas de Control de Versiones (SCV) 22
   - SubversionSVN 25
5. Herramientas de construcción. 28
   - ANT 28
   - MAVEN 28
6. Servidor de integración continua 29
   - Continuum 30
   - Solano CI 31
   - Bamboo 32
   - Cruise Control 34
   - Jenkins 36

Capítulo 3 38

Elicitación y Análisis de requisitos. 38

Capítulo 4 40

1. Implantación de la Integración Continua con Jenkins. 40
2. Configuraciones generales de Jenkins 40
   - 2.1. Seguridad 42

2.2. Credenciales 43
2.3. Sistema 44
2.4. Administración de Plugins 47
2.5. Administración de Nodos. 48

3. Configuración de proyectos en Jenkins 49
3.1. Creación de tareas. 49
3.2. Configuración de tareas. 53

4. Ejecución de tareas. 62

5. Optimización del código. 63

6. Cambios en la planificación. 65

Conclusiones. 66

Bibliografía 67

Capítulo 1

1. Introducción

El desarrollo de proyectos software de grandes dimensiones implica la participación de grupos de trabajadores desarrollando código o documentación bajo entornos de trabajo comunes. El trabajo desarrollado por cada uno se realiza de forma independiente en un entorno local, lo que supone una posterior unión de las partes elaboradas. Esta integración no suele ser trivial y presenta numerosos conflictos relacionados con las versiones de las herramientas utilizadas, discrepancias en el código y pérdida de versiones de documentos entre otras. Lo que puede convertir el proceso de integración y pruebas en una tarea larga y tediosa. Esto se puede evitar aplicando la integración continua.

La integración continua es un proceso de desarrollo software que consiste en el seguimiento diario del proyecto e integración periódica del código para capturar los errores lo antes posible y corregirlos en su inmediatez, así como mantener informados a los miembros implicados en el desarrollo en todo momento. Todo esto con el fin de conseguir un código de calidad en el menor tiempo posible y evitar costes tanto a nivel de desarrollo como a nivel del ámbito laboral de los trabajadores.

2. Objetivos.

El principal objetivo de este trabajo es comprender el proceso de integración continua y aplicarlo sobre un proyecto de desarrollo software. Implícitamente existen otros dos objetivos primordiales para su desempeño. Uno de ellos es introducirnos en el contexto en el que se desarrolla el proyecto, la gestión documental, lo que conlleva el estudio del entorno y de las herramientas utilizadas como son, *Documentum* y los componentes que lo integran. El otro es comprender los beneficios de la integración continua así como analizar las herramientas más avanzadas necesarias para aplicar el proceso y obtener los mejores resultados.

Tras comprender el campo en el que nos encontramos, el trabajo se centrará en el servidor de integración utilizado en este proyecto, Jenkins. A partir de este punto podemos diferenciar otros objetivos. Por un lado, conocer las competencias que existen entre servidores de integración continua, para lo que será necesario realizar un análisis de algunos de los servidores más populares en la actualidad. Por otro lado, será necesario adaptar Jenkins a las necesidades de nuestro trabajo con el fin de realizar la ejecución y el despliegue de los componentes que forman el proyecto software. Esto implica configuración de servidor, configuración de tareas y ejecución de proyectos en Jenkins.

Finalmente, para conseguir un software de mayor calidad, se optimizará el código a través de Sonar, lo que implica adquirir un conocimiento mínimo de la herramienta así como el propio código que conforma el proyecto para poder corregirlo y con ello mejorar los indicadores de calidad.

3. Metodología.

La integración continua en sí es un método empleado en proyectos de desarrollo software, por lo que el trabajo se ha desarrollado utilizando el método de integración continua. Para ello se han empleado las siguientes herramientas software:

- Sistema de control de versiones: SubversionSVN 1.9.3 e interfaz gráfica TortoiseSVN 1.9.3
- Herramientas de construcción: Maven 4.0.0 y Ant 1.7.0
- Servidor de integración continua: Jenkins 1.576

Respecto al entorno en el que se ha desarrollado el proyecto, la gestión documental, se han empleado las siguientes herramientas:

- Plataforma de gestión documental: Documentum 6.7.
- Herramientas para el desarrollo de aplicaciones: Documentum Composer 6.7 SP1 y Eclipse Kepler Service Release 2.

4. Resumen de la memoria.

En la memoria se pueden diferenciar tres amplios bloques. Tras el resumen de los contenidos que se van a tratar, las partes a desarrollar se encapsulan en la Gestión Documental, la Integración Continua y la implantación del proceso con Jenkins.

El primer bloque presenta una introducción al campo de la gestión documental así como al estudio de la plataforma sobre la que se desarrolla el proyecto, *Documentum.* Se describe su historia, su arquitectura y las herramientas que integra.

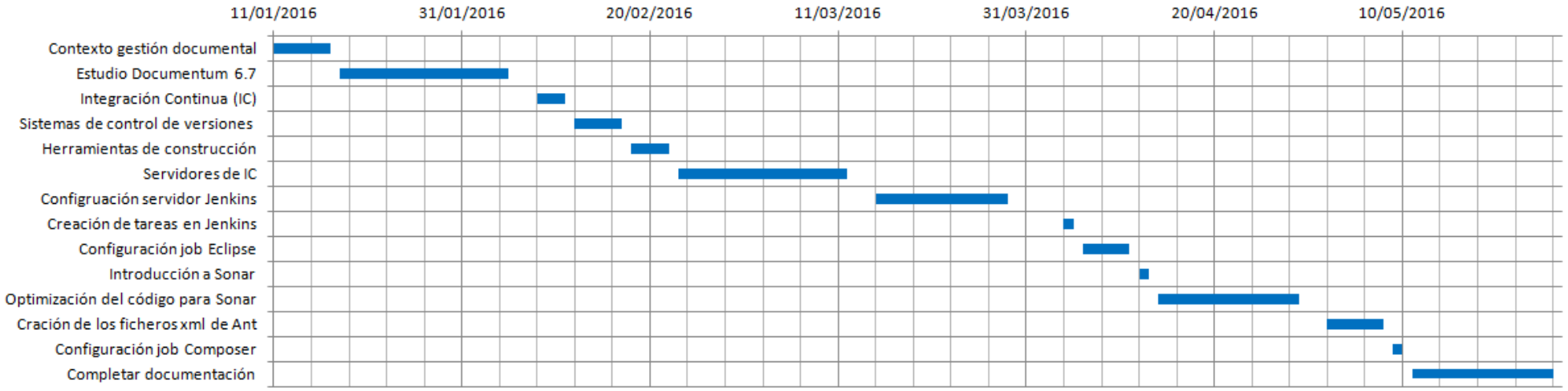
En el segundo bloque se introducen los fundamentos y los objetivos de la integración continua. Además se detalla la estructura y funcionalidad de las herramientas software que se deben utilizar en el proceso. Una de dichas herramientas son los servidores de integración continua, por lo que encontramos en esta sección la descripción de algunos servidores populares incluido Jenkins.

El tercer bloque se centra en el servidor Jenkins y en el proceso de adaptación del mismo a los componentes software que forman el proyecto. Este incluye las configuraciones generales del servidor, las configuraciones de las tareas para la ejecución de cada componente y el análisis de calidad.

Finalmente existe un apartado en el que se exponen las conclusiones a las que ha derivado el desarrollo del trabajo.

5. Planificación

La planificación está estimada en días, pero las horas dedicadas no serán equitativas. No se incluyen los domingos ni los días 24, 25 y 26 de Marzo por ausencia. De lunes a viernes la estimación de horas dedicadas cada día es de 4 horas. Los sábados la estimación es de 7 horas. A continuación se muestra el diagrama de Gantt correspondiente a la estimación inicial.



	Completar documentación	Configuración job Composer	Cración de los ficheros xml de Ant	Optimización del código para Sonar	Introducción a Sonar	Configuración job Eclipse	Creación de tareas en Jenkins	Configuración servidor Jenkins	Servidores de IC	Herramientas de construcción	Sistemas de control de versiones	Integración Continua (IC)	Estudio Documentum 6.7	Contexto gestión documental
Fecha Inicio	11/05/2016	09/05/2016	02/05/2016	14/04/2016	12/04/2016	06/04/2016	04/04/2016	15/03/2016	23/02/2016	18/02/2016	12/02/2016	08/02/2016	18/01/2016	11/01/2016
Duración (días)	15	1	6	15	1	5	1	14	18	4	5	3	18	6
Fecha Fin	27/05/2016	10/05/2016	07/05/2016	30/04/2016	13/04/2016	11/04/2016	05/04/2016	02/04/2016	14/03/2016	22/02/2016	17/02/2016	11/02/2016	06/02/2016	16/01/2016

Gráfica 1 – Diagrama de Gantt.

Capítulo 2

1. Gestión documental.

Los documentos se han utilizado a lo largo de los años en todos los ámbitos que nos rodean. Todo documento tiene un contenido y un ciclo de vida que comienza en su creación y termina en su eliminación o en su estado de permanente conservación. En los últimos años con la llegada de las nuevas tecnologías el uso del documento digital[1] se ha convertido en un concepto habitual y necesario. La masiva cantidad de documentos generados hasta nuestros días ha creado la necesidad de buscar soluciones para su correcta organización, garantizar un almacenamiento seguro y obtener fácil disponibilidad a ellos.

La gestión documental se define como un conjunto de normas, técnicas y prácticas para administrar el flujo de documentos de todo tipo en una organización. Se han desarrollado programas capaces de gestionar documentos independientemente de su formato y su contenido. La mayor parte de veces esto conlleva a tratar con información no estructurada ya que un gran porcentaje del contenido de las empresas no está estructurado. Los datos no estructurados son aquellos que no tienen un tipo predefinido ni una estructura uniforme, dificultando por ello su almacenamiento en una base de datos racional. Con este tipo de sistemas se pueden realizar las siguientes acciones:

- Estructurar y catalogar la información.
- Controlar el versionado de los documentos.
- Gestionar los posibles estados de un documento.
- Controlar el acceso a los documentos mediante la gestión de los permisos de usuario.
- Realizar búsquedas exhaustivas.
- Distribuir la documentación.
- Realizar conversiones de formato.
- Gestionar flujos de trabajo.
- Integrar con herramientas de generación de contenido.
- Establecer políticas de retención y almacenamiento.
- Escanear.
- Controlar la autenticidad a través de encriptación y firmas digitales.
- Auditar.

[1] Un documento digital es “el que contiene información codificada en forma de dígitos binarios que puede ser capturada, almacenada, analizada, distribuida y presentada por medio de sistemas informáticos (La voz del Bibliotecario, [en línea] https://fterrazas.wordpress.com/glosario/) ”

Claramente todo esto conlleva una seria de ventajas, como por ejemplo:

- Reducción de tiempo en búsquedas y en el acceso a la información.
- Reducción del riesgo ante las versiones de un documento.
- Mayor accesibilidad desde cualquier ubicación.
- Reducción de los costes referentes al archivado.

Hasta la actualidad se han desarrollado múltiples herramientas cuyo objetivo principal ha sido la gestión de contenido empresarial. Entre las más conocidas se encuentran: *Alfresco, Nuxeo, Sharepoint, Documentum y OpenText.*

Todas ellas ofrecen funcionalidades básicas relacionadas con la gestión documental como: gestión de grupos y usuarios, drag and drop, versionado de documentos, previsualización online de documentos, historial de acciones de usuario, integración Microsoft Office, sistemas de notificaciones y exportación en distintos formatos.

- **Alfresco**

Gestor de contenidos empresariales basado en Java de software libre. Dispone de una plataforma accesible desde cualquier navegador. Es extensible y personalizable permitiendo también la integración con otras aplicaciones. Dispone de herramientas colaborativas como calendarios de equipo o tableros de discusión.
Las cuotas de *Alfresco* pueden calcularse según la cantidad de CPU utilizada (Alfresco One) o el número de usuarios (Alfresco Cloud).

- **Nuxeo**

Herramienta de gestión documental de software libre desarrollada en Java. Disponible para sistemas operativos como Windows, Linux y Mac OS x.
Además de las funcionalidades básicas mencionadas *Nuxeo* permite, entre otras cosas, la creación de tipos documentales, la gestión de plantillas de documentos y la búsqueda de objetos por tipo y contenido.
Respecto al coste de sus servicios *Nuxeo* ofrece tres posibles subscripciones: Silver, Gold y Platinum. El precio de estas varía por la diferente cobertura de funcionalidades, siendo Silver la más barata y Platinum la más costosa.

- **SharePoint**

Gestor de contenidos empresariales de software propietario creado por *Microsoft*. Disponible sólo para Windows.
Presenta ediciones que ofrecen diferentes servicios a distintos precios. Por un lado se encuentra *SharePoint Fundation* la cual ofrece las funcionalidades básicas y forma la base

del resto de productos. Por otro lado esta *Server Sharepoint* para grandes volúmenes de contenido y funcionalidades empresariales más específicas. Este producto tiene dos versiones, el *Estándar* y el *Enterprise,* la segunda versión se diferencia de la estándar en la posible integración con herramientas de inteligencia empresarial, es decir, desarrollo de contenido para el análisis del rendimiento.

- **OpenText**

Alchemy OpenText es una herramienta para la gestión de documentos empresariales basado en software propietario creado por *OpenText Corporation*. Este se complementa con *OpenText IX* el cual dispone de productos destinados a la protección del intercambio de información como, *Secure Mail* integrado con Microsoft Outlook para correo certificado o, *Secure MFT* para aportar seguridad al intercambio de grandes volúmenes de información tanto interno como externo.

2. Documentum 6.7

Documentum es una plataforma de gestión de contenido empresarial, propiedad de EMC Corporation. Mediante software propietario ofrece servicios de creación, gestión, distribución y almacenamiento, es decir, una solución completa a las necesidades empresariales de gestión documental.

La empresa Documentum fue fundada en 1990 por Howard Shao y John Newton con el objetivo de buscar una solución para la gestión de información no estructurada. Desarrollaron un par de sistemas personalizados para la compañía de aviones Boeing y para la empresa farmacéutica Syntex.

En 1993 Documentum presentó su Sistema Electrónico de Gestión de Documentos (EDMS) el cual consta de un repositorio compartido ejecutado en un servidor central permitiendo el acceso a la información a usuarios de distintas aplicaciones cliente. Más adelante se le incorporó a este sistema controles de acceso y flujos de trabajo para la revisión de documentos y aprobación de procesos. Cinco años más tarde fue posible la incorporación de EDMS a la web.

Con el lanzamiento de Documentum 4i en el año 2000 se podía disfrutar por completo del llamado Webtop (Web Aplicatión Enviroment) producto que permite la integración con aplicaciones externas y la distribución de contenido a portales, servidores de aplicaciones y sitios web.

En 2002, con Documentum5, se convirtió en un sistema unificado de gestión de contenidos empresariales (EMC). La plataforma permite el almacenamiento de una gama prácticamente ilimitada de tipos documentales y además integra el módulo de gestión de procesos de negocio

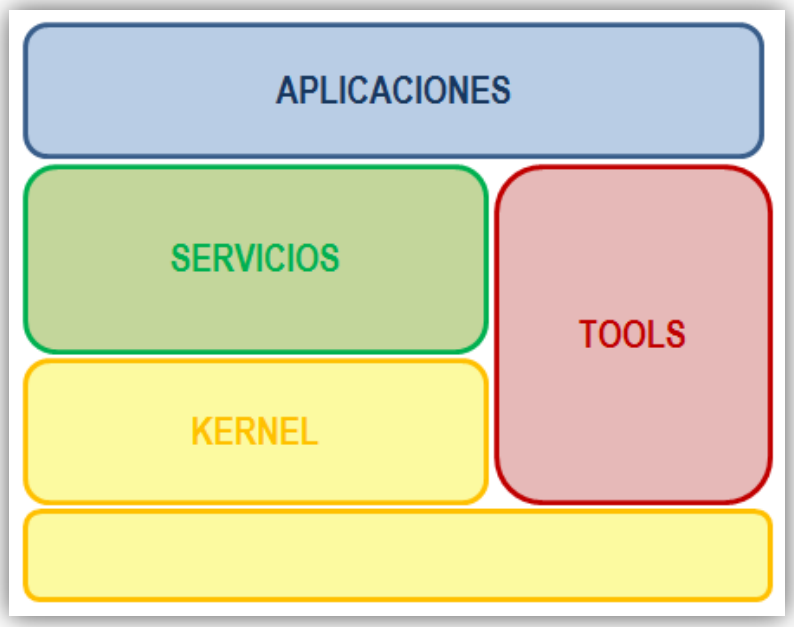
(BPM). Con el tiempo agrega nuevas funcionalidades como: gestión de activos digitales, integración de contenidos empresariales, Documentación Imaging…

En el año 2003 Documentum fue adquirido por EMC Corporation. Desde esta fecha hasta la actualidad han lanzado cinco versiones, siendo Documentum 6.7 la última existente.

Arquitectura

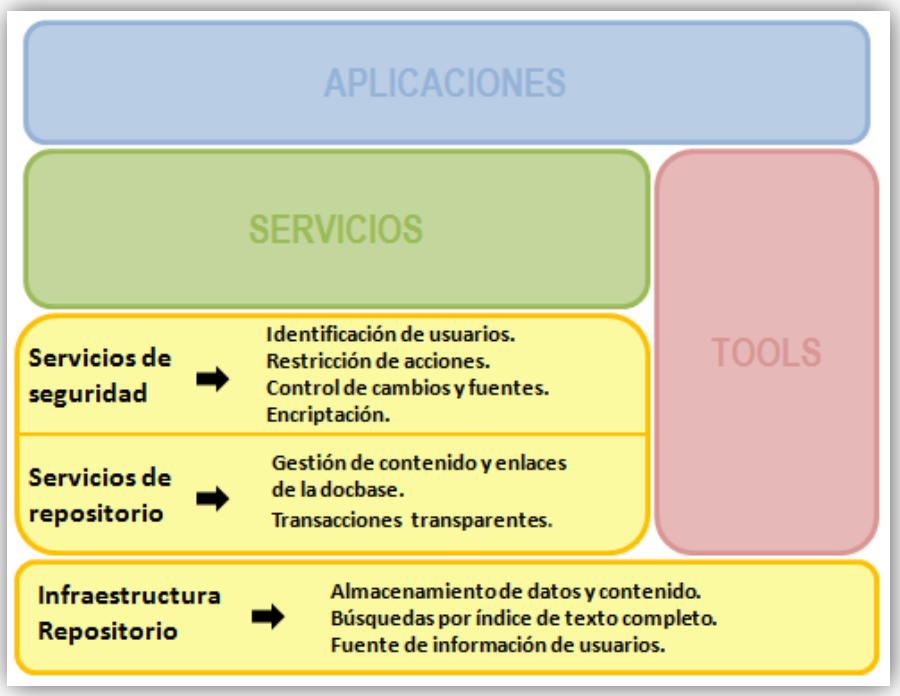
Documentum presenta una arquitectura sólida, escalable, tolerante a fallas y extensible. Se distinguen cuatro capas:

Figura 1 – Capas de la Arquitectura Documentum.



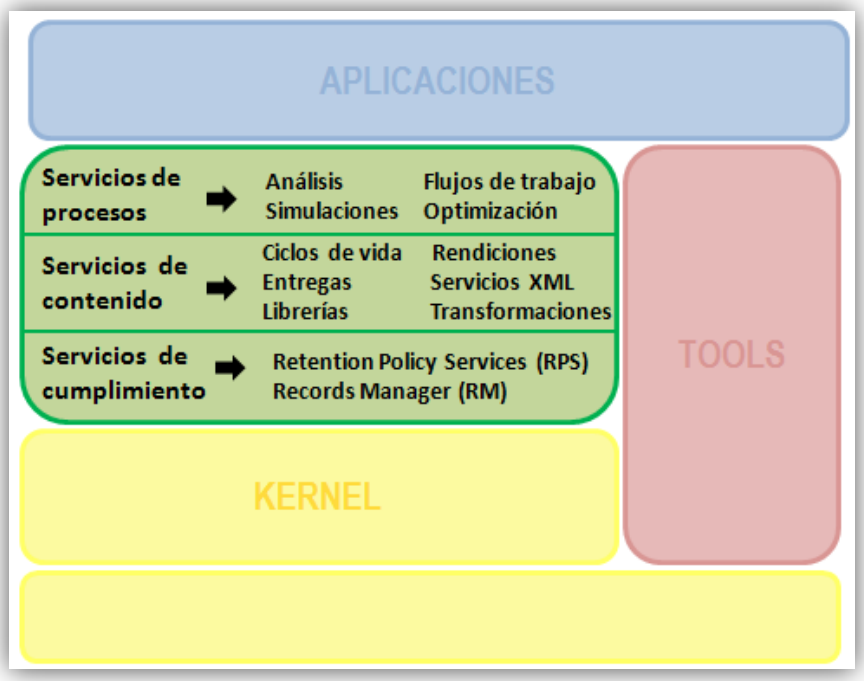
1) **Kernel:** es el elemento principal para la gestión del almacenamiento, acceso y seguridad del contenido.
   - Infraestructura del repositorio:
     - Sistema de ficheros: SAN, NAS, Centera.
     - Base de datos: Oracle, SQL server.
     - Índices full-text: Lucene, Enterprise Search Server.
     - Servicios de directorio (LDAP).
   - Servicios de acceso al repositorio:
     - Content Server: gestión del acceso y los contenidos del repositorio.
   - Servicios de seguridad: autenticación, control de acceso, auditoría y encriptación de comunicaciones.

Figura 2 – Kernel Documentum.



2) **Servicios de aplicación:** proporciona la capacidad de organizar y controlar el intercambio de datos.
   - Servicios de cumplimiento de normativa legal.
     - Políticas de detención.
     - Seguridad y control de accesos.
     - Gestión de registros: gestión de los documentos considerados reflejo de las actividades de una empresa (registros) desde su creación a su eliminación.
     - Políticas de nomenclatura: seguir las convenciones de las nomenclaturas.
   - Servicios de contenido: gestión básica de las funcionalidades del repositorio (flujos de trabajo, ciclos de vida, versiones, búsqueda y transformaciones).
   - Servicios de procesos: gestión de procesos de negocio y entornos colaborativos.

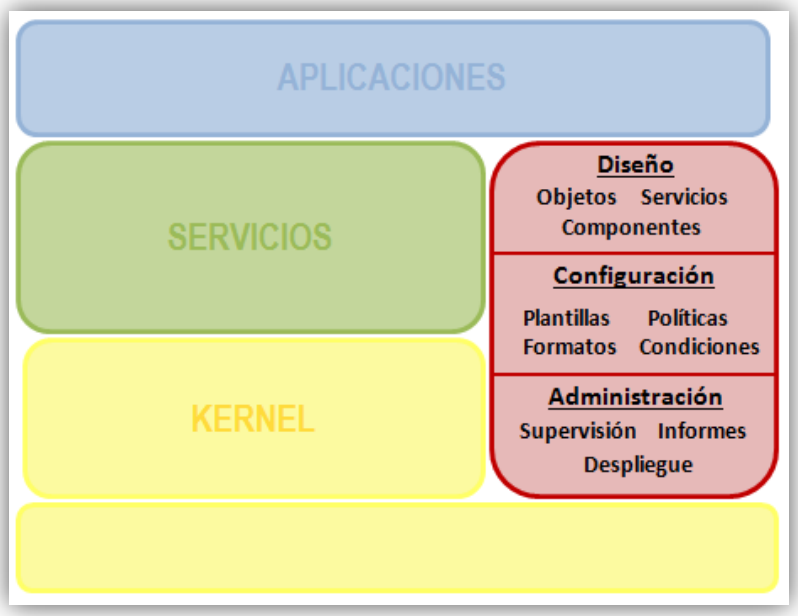
Figura 3 – Servicios de aplicación Documentum.



3) Tools

- Herramientas de desarrollo y diseño:
  - *Documentum Composer*: IDE basada en Eclipse.
  - *Documentum Foundation Services* (DFS): API de desarrollo orientado a servicios.
  - *Documentum Foundation Classes* (DFC): API de desarrollo orientado a objetos.
- *Business Object Framework* (BOF): Type Based Object (TBO) y Service Based Object (SBO).
- Herramientas de configuración:
  - Webtop Presets: configuraciones realizadas por los usuarios a través de la interfaz.
  - Forms: creación de interfaces
  - Templates: contenido inicial para nuevos documentos.
  - Policies: control de comportamiento de los contenidos.

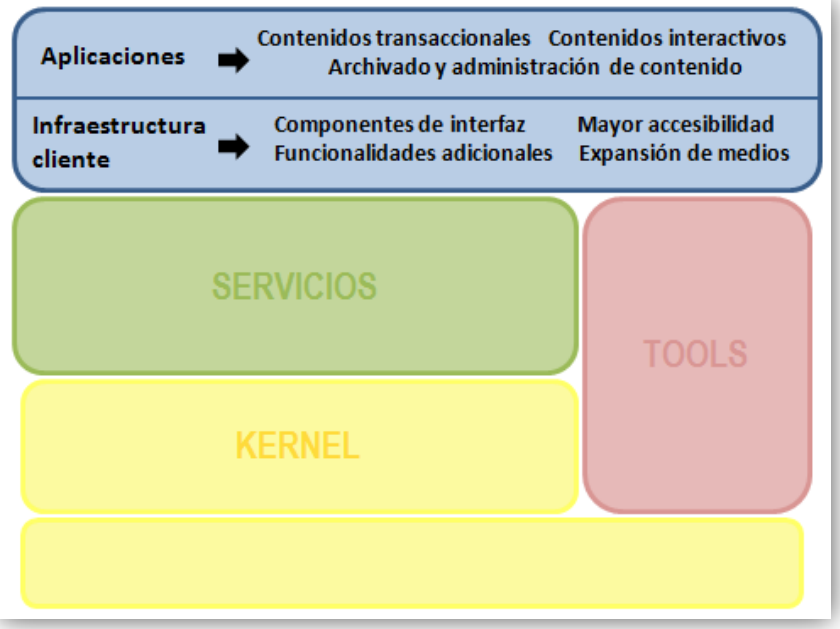
Figura 4 – Herramientas de Documentum.



4) Aplicaciones

- Infraestructura cliente:
  - UFC: transferencia de contenido entre cliente, servidor de aplicaciones y el *Content Server*.
  - WDK: framework de desarrollo web.
  - Conectores con Office.
  - Protocolos: JDBC, ODBC, WebDav, FTP, File Share, DFS, CMIS
- Contenido:
  - Gestión documental: *Webtop*, *Content Management Portlets*, *Content Services for SharePoint*, *Documentum Client for Outlook*, *Documentum Transformation Services* y *PDF Annotation Services.*
  - Búsquedas y clasificación: *Federated Search Services* y *Content Intelligence Services*.
  - Colaboración: *Documentum CenterStage* y *Documentum eRoom.*
- Transacciones: *Captiva*, *Process Builder*, *Forms Builder* y *TaskSpace*.
- Gestión de contenido interactivo:
  - WCM: *WebPublisher* y *Content Delivery Services*.
  - DAM: *Digital Asset Management* y *Media Transformation Services.*
  - Publicación editorial: *Content Services for WoodWing*.
  - Archivo: *email archiving*, *SAP Archiving*, *Archive services for reports* y *archive services for SharePoing*.

Figura 5 – Aplicaciones de Documentum.



Docbase y tipos documentales

El repositorio de Documentum o *Docbase*, utiliza una tecnología orientada a objetos. En Documentum existen tipos predefinidos, los denominados *tipos documentales*. Cada tipo documental posee una plantilla predefinida de propiedades y métodos. Todos los objetos pertenecen a un tipo, es decir, son instancias de los tipos predefinidos. Se puede crear nuevos tipos, a los que se denominan *Custom Types*, a partir de un tipo documental (dm_document) o a partir de otro *Custom Type*.

La organización de los tipos es jerárquica, de forma que los subtipos heredan todas las propiedades del supertipo y junto a estas presentan otras propiedades adicionales.

Como muestra la *Figura 6,* en el repositorio se distingue el sistema de ficheros, donde se guarda el contenido de los objetos, y la base de datos relacional, la cual contiene las propiedades (metadatos) que describen los objetos.

Los objetos pueden ser referenciados por su nombre interno o por una etiqueta asociada. El nombre interno es como viene representado en la base de datos y la etiqueta es utilizada en las aplicaciones cliente.

Figura 6 – Docbase Documentum.

Content Server

Content Server es una plataforma que proporciona servicios de gestión de contenidos, administración de procesos y seguridad.

Gestión de contenidos

La gestión del contenido del repositorio y el acceso al mismo se realiza gracias al *Content Server.* Como lenguaje de consulta se utiliza Document Query Language (DQL), su sintaxis y estructura está basado en SQL pero con capacidades de *Documentum* específicas. Puede recuperar, actualizar, eliminar, crear objetos y realizar búsquedas de contenido indexado y metadatos.

Administración de procesos

Existen dos conceptos importantes en este campo, los *workflow* y los *lifecycle*. Un *workflow* o flujo de trabajo, es una red de actividades en las que participan uno o varios objetos. Las tareas llevadas a cabo pueden alterar los objetos existentes en el repositorio o crear nuevos objetos. Las actividades pueden definirse en serie o en paralelo. Si una actividad depende de que otra finalice estas estarán conectadas en serie, en el caso de que puedan ejecutarse simultáneamente estarán conectadas en paralelo. Las actividades pueden reutilizarse para más de un proceso pero esta debe tener un nombre único para cada proceso.

Un *lifecycle* o ciclo de vida es una sucesión de estados linealmente conectados por los que va circulando un documento. Cada estado define una serie de acciones a realizar y el paso de un estado a otro depende de las reglas de negocio que se hayan impuesto. El documento puede ser promovido a un estado posterior, degradado a un estado anterior o incluso degradado al estado inicial del ciclo de vida. Se distinguen dos tipos de estados: los estados normales, a los que forman parte el estado inicial (base), los estados intermedios y el estado

final, y los estados de excepción, en los que el ciclo de vida puede suspenderse de forma temporal.

Seguridad

Los usuarios tienen limitaciones de acceso tanto al repositorio como a la aplicación web. Un usuario es una instancia de tipo dm_user y este tiene configuradas propiedades que determinan el nivel de acceso.

El acceso a la aplicación cliente, a lo que se denomina *Client Capability,* tiene cuatro niveles distintos y cada uno incluye las funcionalidades del nivel anterior.

Tabla 1 – Niveles de capacidades de un usuario sobre la aplicación cliente.

Client Capability	
Consumidor	Búsqueda, lectura y copia de documentos.
Contribuidor	Creación de documentos/carpetas. Modificación y borrado de documentos. Inicio de worlflows y tareas en ellos.
Coordinador	Creación de archivadores. Creación de documentos virtuales. Creación de ciclos de vida.
Administrador del Sistema	Mantenimiento del repositorio. Gestión de usuarios y grupos. Configuración del Content Server.

El acceso al repositorio depende de los privilegios que tenga asignados cada usuario. Se distinguen los privilegios básicos y los extendidos descritos en la *tabla 2*. En el caso de que el usuario sea *Sysadmin* tiene permisos para realizar tareas básicas de administración y en el caso de que el usuario sea *Superuser* tiene capacidad para llevar a cabo todo tipo de tareas administrativas.

Tabla 2 - Acciones de un usuario sobre los objetos de la Docbase.

Privilegios básicos	
Valor por defecto. Sin funcionalidad adicional	
Creación de nuevos tipos documentales	
Creación, modificación y borrado de carpetas	
Creación, modificación y borrado de grupos	
Sysadmin	**Superuser**
Creación de tipos, grupos y carpetas	Dar y revocar privilegios a Sysadmin y Superuser
Activación/desactivación de usuarios	Propietario de todos los objetos del repositorio
Modificación de usuarios y grupos	Desbloquear objetos
Dar y revocar privilegios	Modificación y borrado de tipos documentales de otro usuario
Creación y modificación de ACLs	Modificación y borrado de ACLs de otro usuario
Gestión de ciclos de vida	Modificación del propietario de un objeto
Modificación de workflows	Borrado de ACLs del sistema
	Registro,borrado y consulta de las tablas de repositorio
Privilegios extendidos	
Sin funcionalidad sobre la auditoría	
Configuración de eventos de auditoría	
Borrado de entradas auditoría	
Acceso a entradas de auditoría	

Para acceder al repositorio cada usuario debe autenticarse introduciendo sus credenciales. La autenticación puede realizarse mediante lo siguientes métodos:

1. Cuenta del sistema operativo: compara la contraseña interna del usuario contra la del sistema operativo.
2. Servidor LDAP.
3. Inline password: contraseña como atributo del objeto.
4. Plugin de autenticación.

En la protección a nivel de objeto toman protagonismo las *Access Control Lists (ACLs).* Son un conjunto de permisos que determinan cómo actúa cada usuario con los objetos.

Un objeto sólo puede tener un ACL y un ACL puede estar asociado a 'n' objetos. Se puede aplicar un ACL al propietario del objeto, a un usuario, a un grupo o al mundo (usuarios no especificados en la ACL).

El ACL es un objeto en sí en el repositorio (dm_acl) formado por una lista de usuarios/grupos junto con un conjunto de permisos básicos y extendidos para cada uno de ellos.

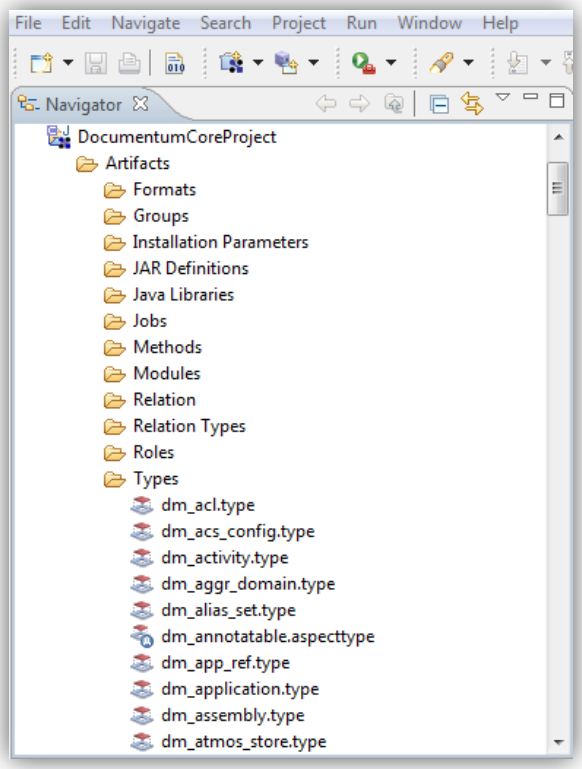
Documentum Composer

Composer es la herramienta básica para el desarrollo, implementación y configuración de aplicaciones en *Documentum* cuya IDE está basada en Eclipse.

Los recursos que utiliza se denominan artefactos, estos se corresponden con los objetos del repositorio. Existen varios tipos de artefactos: formatos de fichero, ciclos de vida, listas de acceso (ACLs), tipos documentales, servicios web, etc.

Los proyectos se agrupan en espacios de trabajo o workspace. Cada workspace tiene un proyecto denominado DocumentumCoreProyect, este es de sólo lectura y actúa como proyecto de referencia porque contiene todos los artefactos correspondientes a los tipos predeterminados iniciales creados en el repositorio.

Figura 7 – Estructura del proyecto DocumentumCoreProject.



El artefacto 'Modules' recoge la implementación de los TBO y SBO. Estos forman parte de la herramienta *Business Object Framework* (BOF) la cual proporciona el marco de la lógica de negocio. En *Documentum* la lógica de negocio está totalmente separada del cliente y BOF permite crear componentes lógicos reutilizables. El TBO está orientado a los tipos documentales y con él se puede aplicar funcionalidades a todas las instancias de un tipo de objeto concreto. El SBO está orientado a servicios y permite crear módulos para diferentes aplicaciones. Además del artefacto 'Modules' también es necesario los artefactos 'Jardef' lo cuales contienen el paquete jar donde se encuentran las clases con las nuevas funcionalidades.

En los proyectos se personalizan los objetos que se van a utilizar para la aplicación web, es decir, se crea el tipo de artefacto correspondiente al objeto a personalizar y se configura. Posteriormente son instalados en el repositorio indicado a través de los archivos DAR. Los DAR son archivos que contienen el formato binario del proyecto preparado para ser desplegado en uno o varios repositorios. Este archivo puede modificarse e instalarse las veces que sea necesario hasta conseguir la customización deseada.

Documentum Foundation Classes (DFC)

Las DFCs son una librería de componentes clave para la plataforma software de *Documentum* ya que actúa como framework para el desarrollo de aplicaciones web. El conjunto de clases e interfaces que lo forman están desarrolladas en Java y se localizan en el paquete *dfc.jar*. Entre las funcionalidades que proporciona destacan las siguientes:

- Conexión y desconexión a un repositorio.
- Ejecución de consultas a un repositorio.
- Modificación de objetos y transacciones.
- Customización o extensión de productos *Documentum*.
- Trabajar con *workflows* o *lifecycles.*
- Unión entre el *Content Server* y las aplicaciones cliente.

Los detalles de configuración se encuentran en el fichero *dfc.properties*. En él se especifican datos como el puerto, el host, los credenciales del repositorio… en la siguiente figura se muestra un ejemplo.

Figura 8 – Código de configuración para las DFC.

```
##############################
###        Entorno         ###
##############################
dfc.docbroker.host[0]=server1
dfc.docbroker.port[0]=1489
dfc.globalregistry.repository=REPO1
dfc.globalregistry.username=dm_bof_registry
dfc.globalregistry.password=+h1PIKGRpU4\=
```

En la *tabla 3* se mencionan algunas de las interfaces más utilizadas junto a su utilidad.

Tabla 3 – Interfaces que ofrece las DFC.

Interfaz	Funcionalidad
IDfSessionManager	Gestión de sesiones y transacciones. El acceso a un repositorio implica iniciar sesiones para la comunicación con el mismo.
IDfSession	Gestión de información referente al servidor, al repositorio y al usuario que ha establecido la sesión.
IDfClient	Obtención de información sobre el repositorio y el servidor en el que se ubica.
IDfLoginInfo	Validación de usuarios que desean acceder al repositorio.
IDfPersistentObject	Creación y eliminación de objetos. Obtención de información y modificación de atributos.
IDfTypedObject	Acceso a los atributos de los tipos documentales.
IDfId	Obtención de los identificadores (r_object_id) de los objetos.
IDfACL	Gestión de las listas de accesos (ACLs).
IDfQuery	Gestión de consultas a la base de datos.
IDfCollection	Acceso a los elementos almacenados en una colección cuyo contenido es el resultado de una query.

Webtop

Webtop es la principal aplicación de *Documentum*. Proporciona la base a partir de la cual se desarrollan otras aplicaciones ajustadas a las necesidades de las empresas.

El acceso se realiza vía URL a través de cualquier navegador. En la página de inicio de sesión además de los credenciales del usuario es necesario indicar la base de datos a la que se desea acceder. En la interfaz de la aplicación se pueden diferenciar cinco partes:

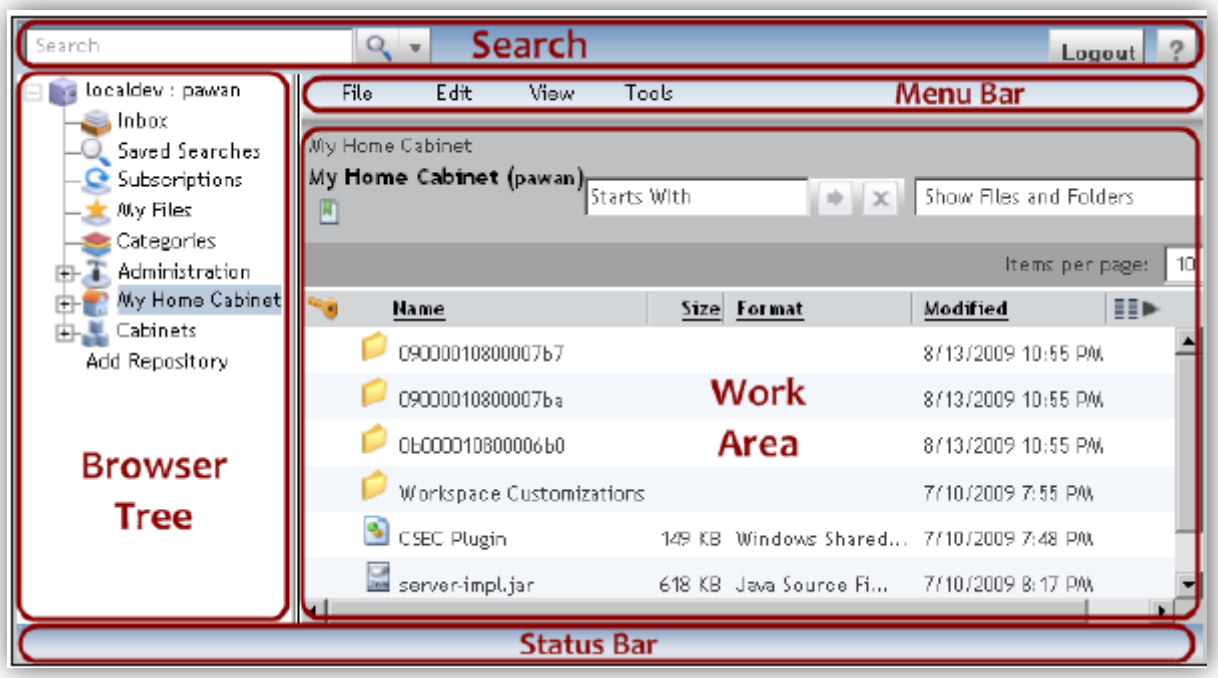
- Árbol de navegación.
  El nodo raíz indica el nombre del repositorio al que estamos accediendo, de él parten otros nodos en los que podemos encontrar, por ejemplo, notificaciones, ficheros recientes, búsquedas guardadas o configuración. El contenido de esta última dependerá de los privilegios del usuario.
  Un nodo importante es el denominado 'Cabinets', en él se encentra la estructura de ficheros que contiene los documentos y otros objetos.

- Buscador.
  Dispone de una barra superior en la que se pueden realizar búsquedas simples y avanzadas, es decir, búsquedas más específicas en las que es necesario incluir parámetros adicionales como operadores relacionales, fechas o tamaños entre otros. Los resultados de las búsquedas pueden guardarse para tener acceso directo a ellas. En esta sección se encuentra además el acceso a la ayuda y al cierre de sesión.
- Área de trabajo.
  Es la parte de la interfaz en la que se muestra el contenido con el que el usuario está trabajando, es decir, los objetos que contiene cada apartado en función del nodo por el que esté navegando. Se puede visualizar directamente alguna de las propiedades de los documentos como el tamaño, la versión y su ACL.

- Menú.
  Barra situada en la zona superior del área de trabajo la cual permite llevar a cabo diferentes acciones. En función del objeto al que se haga referencia o del nodo en el que nos situemos sólo ciertas funciones estarán habilitadas.

- Estado.
  En la parte inferior de la interfaz existe una barra en la que se indica el proceso o estado de ciertas acciones que lleva a cabo el usuario como, por ejemplo, la edición de un documento, la importación, los cambios guardados, etc.

Figura 9 – Interfaz correspondiente a la aplicación Webtop.



Web Development Kit (WDK)

WDK es un framework para el desarrollo de aplicaciones web de *Documentum* que permite modificar la apariencia, acciones de menú, lenguaje y etiquetas así como añadir nuevas funcionalidades.

Está basado en el patrón de arquitectura software Modelo-Vista-Controlador (MVC), es decir, la lógica de negocio está separada de la interfaz de la aplicación.

Cada componente está formado por:

- Ficheros XML que definen la configuración del componente.
- Ficheros JSP para el diseño visual del componente.
- Clases JAVA que definen el comportamiento ante un evento.
- Ficheros NLS que contienen las propiedades de las cadenas.

3. Integración continua (IC)

El proceso de desarrollo de software "Integración Continua" (IC) fue promovido a los largo de los años 90 por Martin Fowler y Kent Beck. Tras la publicación del libro *Extreme Programming Explained 1991* obtuvo mayor importancia. En él se expone la idea de la creación de sistemas basada en una continua construcción y control del código fuente. A lo largo de la siguiente década ya se estableció como una práctica estándar en el entorno industrial.

La integración continua consiste en poder observar y corregir lo antes posible fallos que surjan en el desarrollo de un sistema complejo, es decir, llevar a cabo integraciones automáticas del proyecto en periodos cortos de tiempo. Tras cada modificación del software, el sistema completa una construcción, prueba, despliegue e integración. Si algo ha fallado, todos los usuarios son informados de dónde se encuentra y de cuál ha sido el error. Aplicando este método el software generado con cada nuevo cambio queda probado y está verificado que funciona correctamente.

La integración continua proporciona las siguientes ventajas:

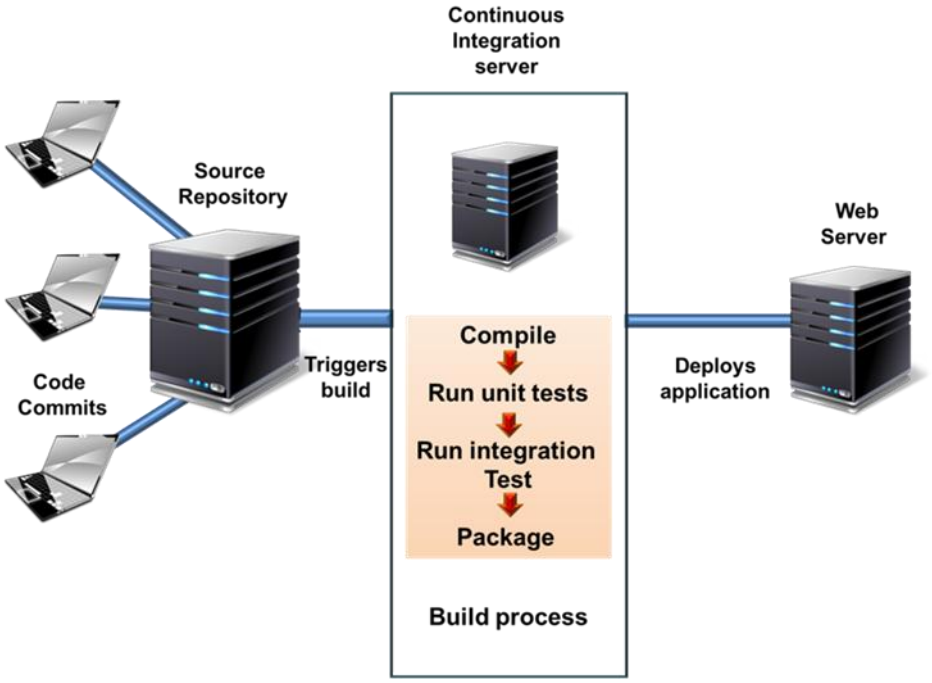
- Reducción del riesgo. La detección del origen del fallo tras pequeños cambios permite su corrección inmediata, por lo que una nueva modificación partirá de un conjunto de código estable.
- Aumento de la visibilidad. Los usuarios implicados en el proyecto pueden ver en todo momento con detalle el estado del mismo.
- Mejora de calidad del software. Mediante la automatización para pruebas unitarias se asegura que el código elaborado hasta ese momento es de calidad.
- Reducción de costes generales. En proyectos complejos la realización de pruebas y la corrección de errores una vez finalizado el proyecto es mucho más costoso.

Para poner en práctica la integración continua es necesario seguir los siguientes puntos:

- Establecer un único repositorio de código fuente.
- Automatizar la construcción.
- Incluir las pruebas unitarias como parte del proceso de construcción.
- Subir frecuentemente al repositorio los cambios realizados en el código.
- Reparar inmediatamente la rama principal del proyecto en el caso de que existan errores y nunca actualizar código sobre una rama con errores.
- Automatizar el despliegue.
- Desplegar en un servidor de pruebas, es decir, en una réplica del entorno de producción.

Todo esto conlleva el empleo de una serie de herramientas que facilitan el proceso de integración y que detallaremos más adelante. Se trata de los sistemas de control de versiones (SCV), los sistemas para la automatización de la construcción y lo servidores de integración continua.

Figura 10 – Continuous Integration (CI).



4. Sistemas de Control de Versiones (SCV)

Una herramienta de control de versiones permite tener uno o varios proyectos con todas las versiones de sus fases de desarrollo en un único repositorio.

Conceptos básicos para trabajar con los SCV:

- *Repositorio*: base de datos con una estructura dada donde se almacena el contenido del proyecto.
- *Check out:* realizar una copia local de un conjunto de archivos desde el repositorio.
- *Commit:* almacenar cambios realizados en el proyecto al repositorio. Se guarda como una nueva versión.
- *Update:* actualizar copia de trabajo local, es decir, obtener los cambios realizados en la última versión.
- *Bloqueo:* bloqueo de carpeta o archivo para garantizar que sólo un usuario puede modificarlo.

- *Rama (branch):* existe la rama principal, también conocida como “trunk”, y otras ramas denominadas “líneas de desarrollo” en las cuales se puede trabajar y realizar pruebas sin que estas afecten al resto del proyecto.
- *Merge:* proceso de introducir los cambios de una rama a otra, normalmente la unión se suele realizar con la rama principal.
- *Conflicto:* fragmentos del archivo en la que el código insertado no coincide. Suelen darse al realizar un Update o un Merge.

Ofrecen las siguientes ventajas:

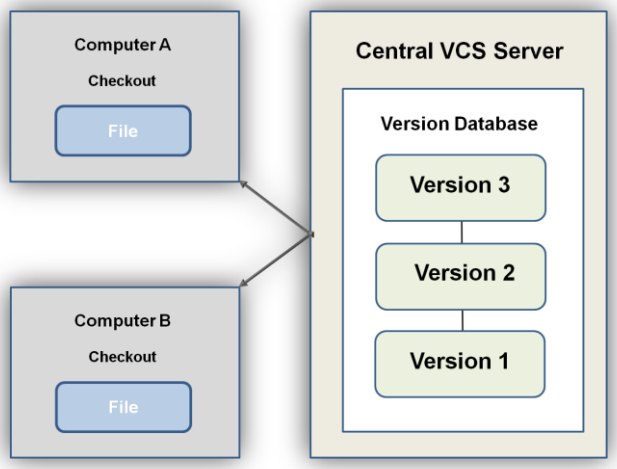
- Revertir archivos o un proyecto entero a un estado anterior.
- Historial de cambios. Saber quién y en qué momento hizo un cambio así como dónde se encuentra un determinado error para su corrección.
- Crear una rama para probar o solucionar una parte del código sin comprometer lo que ya se ha desarrollado.
- Acceso remoto. Es posible acceder al repositorio donde se encuentra el proyecto desde cualquier equipo capaz de conectarse a la red.
- Reforzar la seguridad. Existen diferentes permisos que condiciona el acceso a las distintas ramas del proyecto.

En base a la ubicación del repositorio y el acceso al mismo, existen tres tipos:

- CVS local: una base de datos simple como ubicación del repositorio en la que se lleva registro de todos los cambios realizados sobre los archivos.

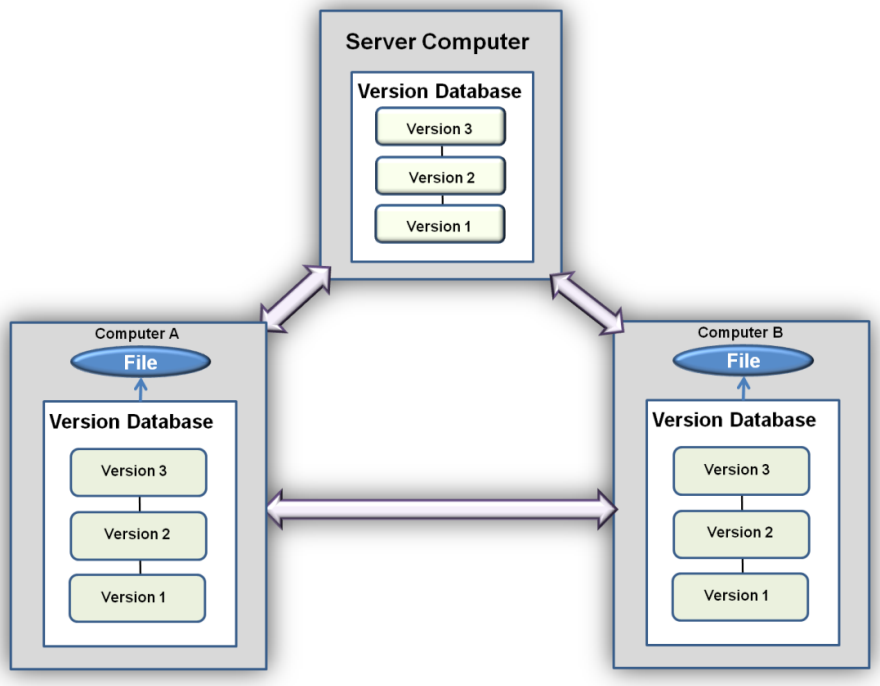
- CVS centralizado: servidor como ubicación del repositorio y orientado a proyectos en los que participan desarrolladores en diferentes sistemas. Los usuarios se descargan la última versión de los archivos en local para realizar los cambios. La desventaja de este tipo es que si el servidor falla nadie puede trabajar sobre el proyecto.

Figura 11 – Control de versiones centralizado.



- CVS distribuido: presenta las mismas características que el centralizado pero con la excepción de que el usuario puede replicar completamente el repositorio en local para trabajar sobre él evitando así el bloqueo de trabajo en el caso de que se produzca un fallo en el servidor y la posible restauración del repositorio.

Figura 12 – Control de versiones distribuido.

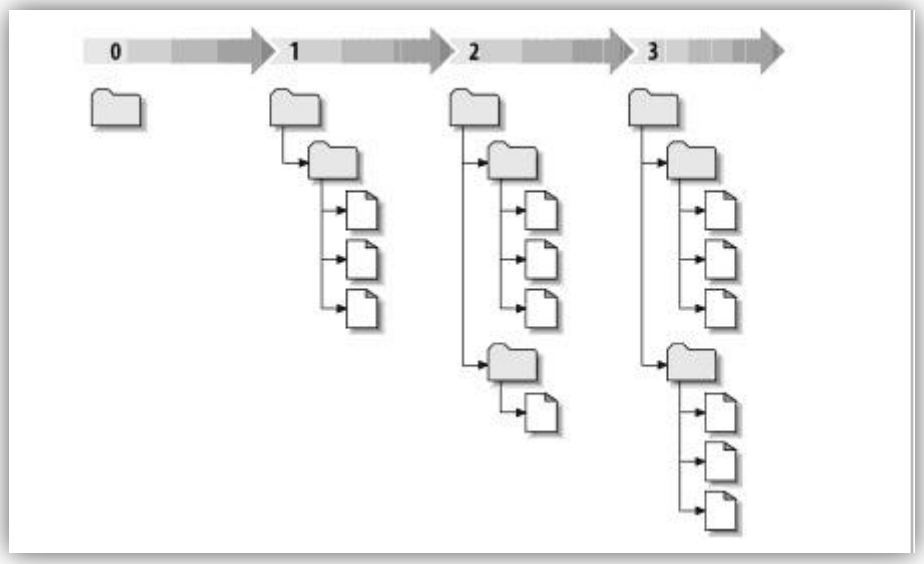


SubversionSVN

La herramienta de control de versiones utilizada a lo largo del desarrollo del proyecto es *Subversión SVN.* Es de tipo centralizado y sus archivos están organizados en forma de árbol dentro de un repositorio central.

El repositorio de *SVN* guarda todos los cambios que alguna vez se hayan escrito en él, cada cambio en cada archivo e incluso los cambios en el propio árbol de directorios. Cada vez que el repositorio recibe nuevos datos se crea un estado específico enlazado al sistema de ficheros denominado "Revisión". Cada revisión tiene asignado un identificador único (el cual es 0 en el caso de que el repositorio este recién creado) y se incrementa de manera automática en cada actualización realizada sobre el repositorio.

Figura 13 – Revisiones Subversion SVN.



Cada proyecto presenta una estructura a la que se denomina estructura TTB la cual está organizada en tres líneas de trabajo:

- Trunk: línea de desarrollo principal.
- Tags: rama de gestión de versiones. Reservado para versiones cerradas, por lo que no son ramas de desarrollo.
- Branches: ramas con evoluciones paralelas a Trunk. Almacena copias de los archivos del árbol principal sobre los que se desarrollan pruebas, correcciones…

Entre otras ventajas destaca su bajo coste a la hora de crear nuevas ramas, esto se debe a que se crea un vínculo interno apuntando a una revisión y árbol específicos en vez de hacer una copia completa en el repositorio. Como resultado, las ramas y las etiquetas son muy rápidas de crear, y casi no conllevan espacio extra en el repositorio.

El método seguido por SVN para la compartición de archivos es copiar-modificar-fusionar. Este método evita sobrescribir un archivo tras ejecutar un *commit* en el caso de que dos usuarios estén realizando modificaciones del mismo archivo en sus copias locales. Esto es posible porque en el momento en que un usuario actualiza el repositorio con los cambios que he realizado en el archivo, *SVN* avisa a otros usuarios que tengan copia local del mismo archivo de que esta desactualizado. De esta forma cuando otro usuario vaya a subir sus modificaciones sabe que antes debe fusionarlo con el que ya se encuentra en el repositorio.

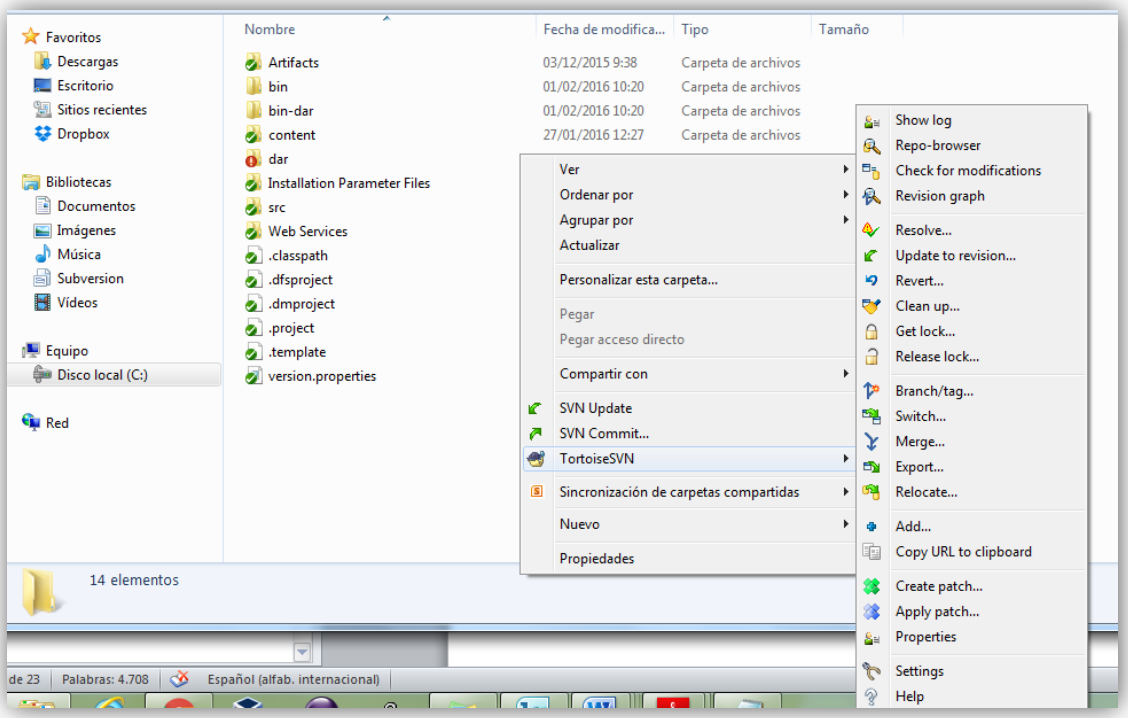
TortoiseSVN

Es un cliente gráfico implementado como una extensión del Shell del Windows por lo que proporciona una interfaz de usuario sencilla para *SubversionSVN*. Algunas características de *TortoiseSVN* son:

- Uso sencillo.
- Admisión de todos los protocolos de *SubversionSVN.*
- Finalización automática de palabras clave o rutas y corrector ortográfico incorporado.
- Obtención de gráficos de todas las revisiones.
- Flexibilidad para la integración de sistemas de seguimientos de fallos basados en la web.
- Disponibilidad en varios idiomas.

Como se puede observar en la *Figura 14*, la copia local de los archivos del repositorio está etiquetada por un símbolo que nos indica el estado de dicha copia. El símbolo verde nos indica que la copia local tiene el mismo contenido que en el repositorio y el símbolo rojo nos advierte de que el contenido es distinto ya sea porque algún usuario a actualizado el repositorio con cambios o porque hayamos realizado nosotros modificaciones en nuestros archivos. En este último caso *Tortoise* permite ver las diferencias que existen entre los ficheros para que el usuario decida qué debe hacer antes de llevar a cabo la operación commit. Con solo un click sobre la carpeta podemos ver de forma intuitiva las opciones que nos ofrece *Tortoise* para llevar a cabo una correcta integración.

Figura 14 – Opciones de Tortoise en carpeta de Windows.



5. Herramientas de construcción.

Durante el desarrollo de un proyecto hay que tener en cuenta aspectos como: añadir las librerías necesarias, crear el proyecto en la estructura correcta y tener los recursos necesarios para construir y desplegar. Para reducir el tiempo de desarrollo y evitar fallos es conveniente automatizar estas tareas pero de forma eficiente. Dos herramientas muy conocidas y con gran uso en la actualidad para llevar a cabo una construcción automatizada son *Ant* y *Maven*. Ambas son proporcionadas por Apache, están destinadas para proyectos basados en Java y permiten automatizar tareas repetitivas de un desarrollo software como la compilación, el testeo, la empaquetado y la distribución.

ANT

Herramienta de construcción basada en un lenguaje específico del dominio de la familia XML. El proceso de automatización de las tareas se realiza con un script Ant, al que se suele denominar *build.xml*, cuya estructura está formada por tres elementos importantes:

- Project: el proyecto para el que está destinado el script. Está formado por al menos un objetivo.
- Targets: representan los objetivos a realizar (inicializar, compilar, testear…). Cada target puede estar formado por una o más tareas.
- Task: unidad de código que se ejecuta para realizar las tareas.

Un archivo *build.xml* no es reutilizable, sólo puede pertenecer a un único proyecto. Algunas de las ventajas que ofrece Ant son:

- Portabilidad. Si tenemos instalado Ant en diferentes sistemas operativos el script *build.xml* funciona correctamente en todos sin necesidad de modificaciones.
- Capacidad de creación de tareas y establecimiento de dependencias entre ellas de forma sencilla.
- Gran variedad de entornos la integran (Eclipse, Netbeans, IntelliJIDEA…).

MAVEN

Herramienta de gestión de proyectos cuyo núcleo se encuentra en el fichero *pom.xml (*Proyect Object Model). Este contiene detalles de la configuración del proyecto como: versión del modelo POM, versión del proyecto, grupo al que pertenece, nombre que se le dará a la biblioteca del proyecto y forma de empaquetado del proyecto (jar o war).

En Maven se distinguen dos elementos, los artefactos y los grupos. Un artefacto es un componente software que podemos incluir en un proyecto como dependencia. Estos artefactos se organizan en grupos.

Una de las características de Maven es que presenta ciclo de vida, es decir, el proceso de construcción y distribución está definido por un conjunto de fases. Las fases del ciclo de vida básico son las siguientes:

- *Validate:* validar si el proyecto y la información necesaria del mismo es correcta.
- *Compile:* compilar fuentes.
- *Test:* pruebas del código.
- *Package:* empaquetar el código en su formato distribuible.
- *Integration-test:* procesar y desplegar el paquete en un entorno donde las pruebas de integración se pueden ejecutar en el caso de que sea necesario.
- *Verify:* comprobar si el paquete es válido y cumple todos los criterios de calidad.
- *Install:* instalación del paquete a nivel local.
- *Deploy:* desplegar el paquete instalado en un entorno específico.

Las fases pueden estar formadas por metas o *goals*, es decir, comandos que realizan tareas específicas.

Ejemplo: limpiar todas las clases compiladas mvn clean:clean

Las metas pueden añadirse al proyecto mediante artefactos denominados plugins. Estos se configuran en el proyecto indicando las metas a ejecutar.

Maven busca las dependencias en el repositorio, el cual está formado por un conjunto de paquetes Jar. Existen tres tipos de repositorios:

- Repositorio local: creado por Maven tras ejecutar cualquier comando mvn en el sistema local.
- Repositorio central: creado por la comunidad Maven Apache se encuentra en la web y contiene una gran cantidad de bibliotecas.
- Repositorio remoto: se encuentra en la web con bibliotecas que posiblemente no se encuentren en el repositorio central.

Si la dependencia no se encuentra en estos repositorios, Maven detiene el proceso y lanza un error. Algunas ventajas que ofrecen Maven son:

- Adaptación de un proyecto a nuevas características Maven sin apenas configuraciones.
- Extensible. Fácil creación de plugins con Java o lenguajes de script.
- Gestión de dependencias gracias al repositorio central.
- Reutilización del archivo *pom.xml* para proyectos diferentes.

6. Servidor de integración continua

Un servidor de integración continua permite automatizar y programar la integración completa del código de un proyecto. Una automatización básica incluye los siguientes procesos:

1. Descargar la última versión del sistema de control de versiones.
2. Compilar el proyecto.

3. Ejecutar las pruebas unitarias.
4. Realizar el despliegue.

Puede incluir otros procesos durante la ejecución como elaboración de documentación, pruebas de análisis estático de código y pruebas de cobertura, entre otras.

La programación de la ejecución puede configurarse con diferentes parámetros: a una hora concreta del día, tras la realización de un commit, tras la construcción de otro proyecto…

El uso de este tipo de herramientas evita discrepancias entre versiones ya que, al realizarse siempre en el mismo entorno, las versiones utilizadas por los elementos de compilación serán siempre las mismas. También permite la detección de fallos porque, a pesar de ser necesaria la comprobación del código modificado antes de realizar un commit, a veces esto no se lleva a cabo de la forma correcta y siguen existiendo errores. Además podemos obtener un histórico de resultados de compilaciones.

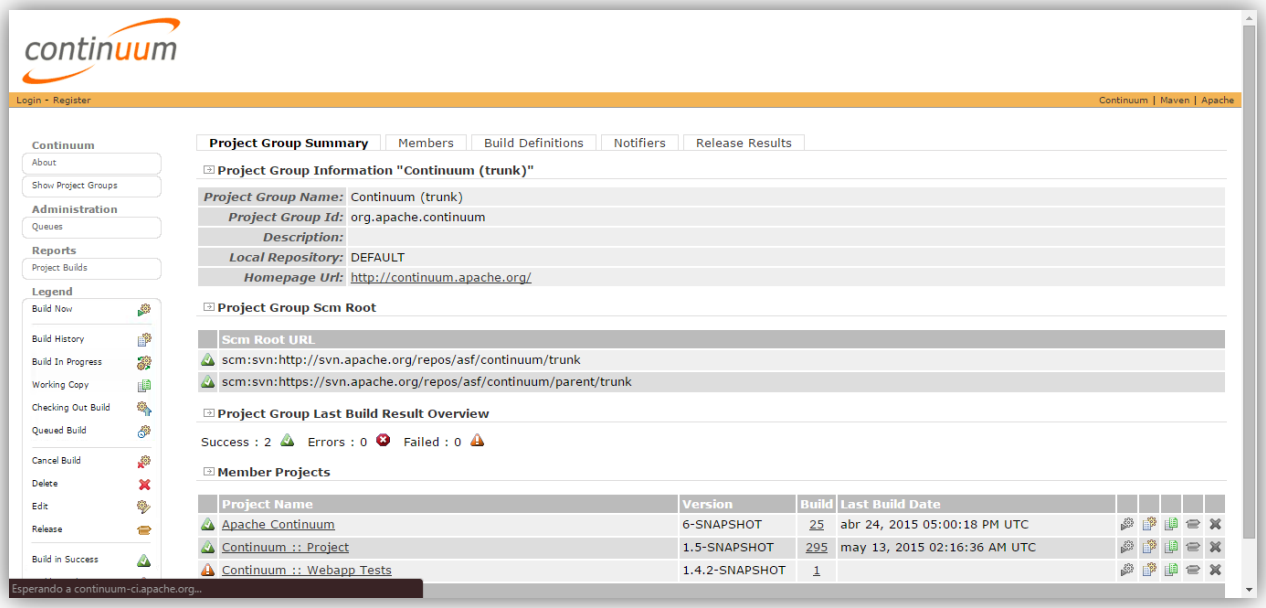
A continuación vamos a describir algunas de las herramientas más conocidas en este campo.

Continuum

Servidor apto para diferentes sistemas operativos. La base de datos por defecto es Apache Derby, pero es posible configurarlo para otras bases de datos de las soportadas por Continuum como MySQL, Postgres, MS SQL Server y Oracle.

Las tareas de construcción se realizan fundamentalmente con Maven pero también permite utilizar scripts de Ant. Las tareas administrativas básicas se realizan a través de la interfaz web (*Figura 15*), para tareas más avanzadas es necesario modificar ficheros de configuración.

Figura 15 – Interfaz web servidor Continuum.



Es potente pero no presenta líneas de ayuda para el usuario. No ofrece mecanismos de extensión o plugins para la integración con sistemas externos. En el caso de querer obtener funcionalidades distintas a las que están establecidas se debe hacer a través de Maven con su correspondiente configuración del fichero *pom.xml* aunque, aun utilizando este método, los resultados no los obtendríamos directamente por pantalla. Las notificaciones a los usuarios se realizan vía e-mail y su configuración puede realizarse mediante la interfaz o a través del *pom.xml* en versiones posteriores a Maven 2.0.

El control de versiones puede llevarse a cabo con Git, Subversion, Perforce y Bazaar entre otros.

Respecto a la seguridad, Continuum soporta LDAP para la autenticación y existen usuarios, roles y permisos para operaciones concretas.

Solano CI

Herramienta para la integración continua distribuida sólo para Linux. Compatible con una gran variedad de bases de datos (Sqlite 3, Mysql, Postgres, Redis, RabbitMQ, CouchDB, NSQ…). No es integrable en entornos de desarrollo integrado (IDE).

Soporta varios lenguajes, entre ellos están Java, Javascript, PHP, Phyton, Scala y Ruby. Las tareas de construcción pueden realizarse mediante Ant, Maven, Gradle y Android. También trabaja con diferentes sistemas de control de versiones como Git, Mercurial, Azure, Github, etc.

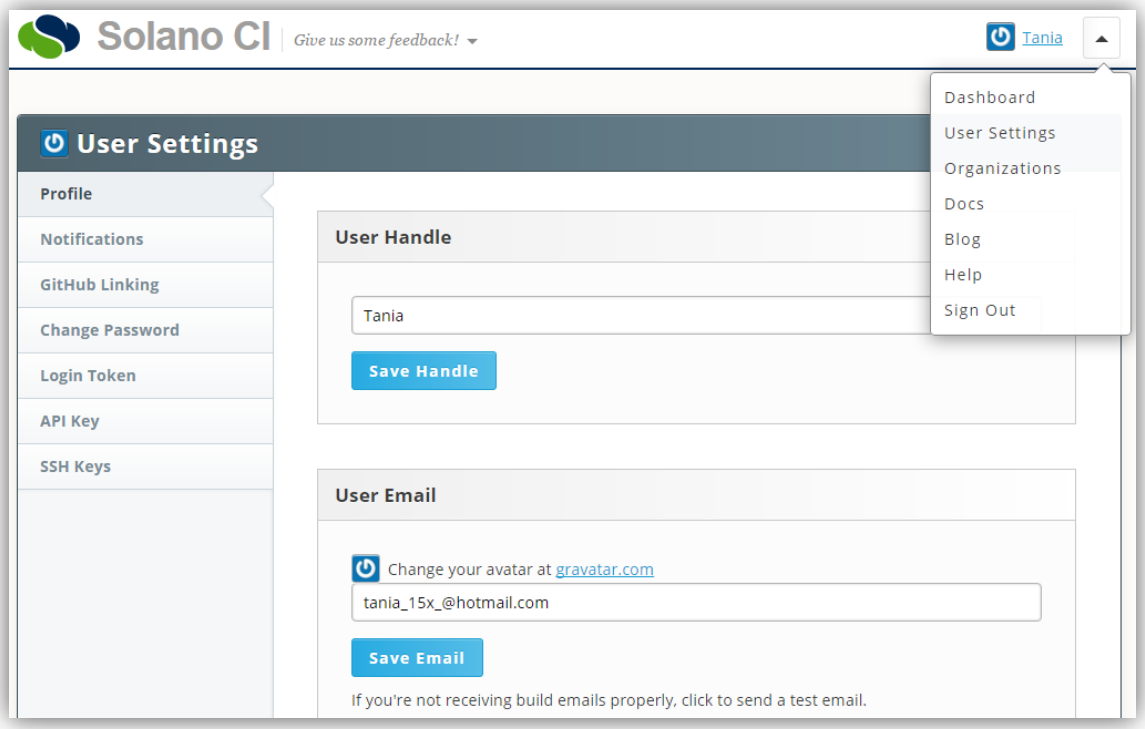
Para realizar la instalación de Solano es necesario tener también instalado Ruby. La configuración está centrada en el archivo *solano.yml*. Este archivo es necesario para controlar la instalación, el arranque y la configuración de la aplicación así como listas de comandos y pruebas a ejecutar. Dispone de un interfaz de línea de comandos (CLI).

Los usuarios en Solano están divididos en organizaciones, estas determinan los permisos que poseen los usuarios pertenecientes a la organización sobre el acceso al código y a los resultados de las pruebas. Un usuario debe pertenecer a al menos una organización y puede tener uno de los siguientes roles:

- Owner: propietario de la organización y control total de la misma.
- Admin: administrador de los usuario de la organización.
- Colaborator.

Iniciando sesión para acceder a Solano tenemos, como se puede observar en la *Figura 18*, la interfaz que nos permite llevar a cabo tareas relacionadas con la modificación del perfil de usuario, la administración de organizaciones y el acceso a repositorios y a documentación.

Figura 16 – Interfaz web servidor Solano CI.



Bamboo

Servidor de integración continua desarrollado por Atlassian que incluye la funcionalidad de administrar entregas dentro de diferentes ambientes es decir, diferentes entornos, dando paso a la entrega continua.

Es potente cuando funciona junto a otras aplicaciones de Atlassian como por ejemplo JIRA, una aplicación basada en web para el seguimiento de errores, de incidentes y para la gestión operativa de proyectos. Es integrable en entornos Eclipse, IntellJ IDEA y Visual Studio.

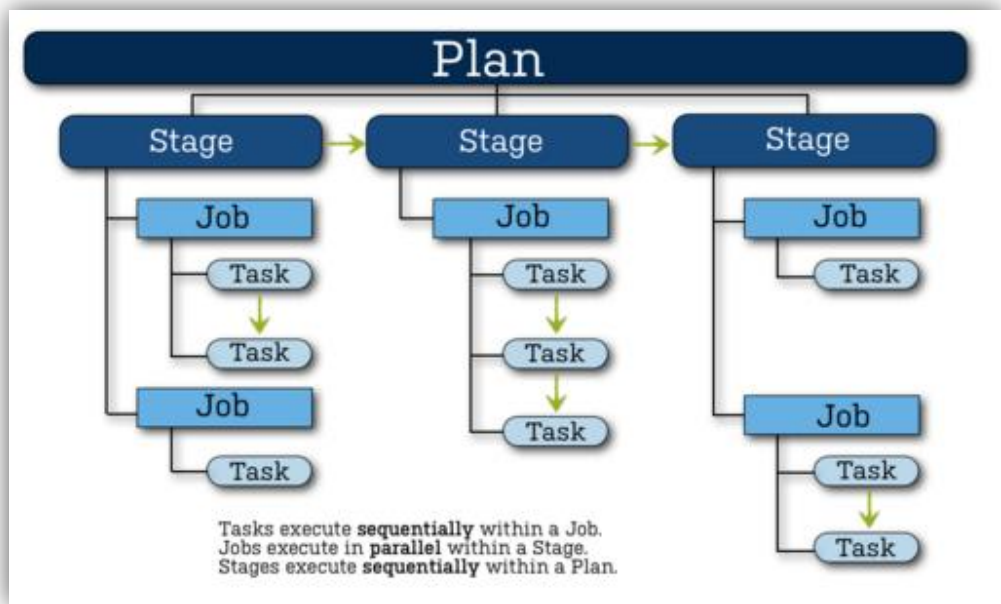
Dispone de interfaz de línea de comandos (CLI) para los sistemas operativos Windows, Linux y OSX. También es posible gestionar los proyectos a través de su API REST.

Trabaja con Subversion, Git, Mercurial o Perforce como control de versiones y con proyectos desarrollados en Java, .Net, PHP y Ruby. En base al lenguaje del proyecto podemos automatizar la construcción con varias herramientas.

- Para Java: Ant, Maven o Grails.
- Para .Net: MSBuild, NAnt o Visual Studio.
- Para PHP: PHPUnit

En Bamboo un proyecto está dividido en planes. Un *plan* está formado por *etapas* y estas incluyen flujos de trabajo (*job*) compuestos por *tareas*, es decir, el plan contiene todo lo necesario para llevar a cabo un proceso.

Figura 17 – Esquema de la organización de un proyecto en Bamboo.



Las tareas son la unidad más pequeña de trabajo, estas corresponden, por ejemplo, a la ejecución de una meta Maven o a la ejecución de un script. El job es una unidad de construcción que controla la secuencia de las tareas, los requisitos necesarios para las mismas y define los artefactos que se producirán en la construcción. Las etapas procesan en paralelo sus job en diferentes agentes (servicios destinados a la ejecución de un job). Es necesario que finalicen correctamente todos los job para poder iniciar una nueva etapa. También pone a disposición de futuras fases los artefactos generados. Por último, el plan especifica la ubicación del repositorio y los permisos de configuración del mismo, así como las dependencias entre los planes y el orden de ejecución de las etapas. Tras la ejecución del plan se notifica al usuario los resultados de la compilación.

Los *artefactos*, mencionados anteriormente, son los paquetes listos para instalar y ejecutar en otro entorno. Una vez creados y probados, los artefactos se empaquetan junto a metadatos relacionados con la presentación de informes y el seguimiento a través de sus entornos formando una *release*, es decir, un conjunto de artefactos en un momento dado en el tiempo. Gracias a esto se puede observar el comportamiento del despliegue en cada entorno.

En Bamboo existen usuarios, autores y grupos. Los autores son aquellos que pueden modificar código que forma parte de algún plan, pero estos no tienen por qué ser usuarios. La seguridad está dividida en dos conjuntos:

- Permisos globales: pueden incluir acceso a Bamboo, creación de un plan, configuración del repositorio o cualquier tipo de operación relacionada con la administración (Admin).

- Permisos de plan: pueden incluir visualización, edición de plan y tareas, construcción, clonación o cualquier tipo de operación sobre el plan incluido la modificación de permisos (Admin).

Cruise Control

Herramienta desarrollada por ThoughtWorks disponible en dos versión, la versión Java y la versión .Net.

Su arquitectura está compuesta por tres módulos principales.

Bucle de construcción: es un proceso que comprueba periódicamente el estado del código y realiza una construcción en el caso de que haya sido modificado en el repositorio. El intervalo de tiempo en el que actúa la define el usuario.

Resultados de construcción JSP: este componente se encarga de mostrar los resultados del bucle de construcción. Se puede visualizar detalles del proyecto como las últimas modificaciones de sus archivos o el estado actual, por otro lado se muestra por pantalla el proceso de compilación y resultados de pruebas.

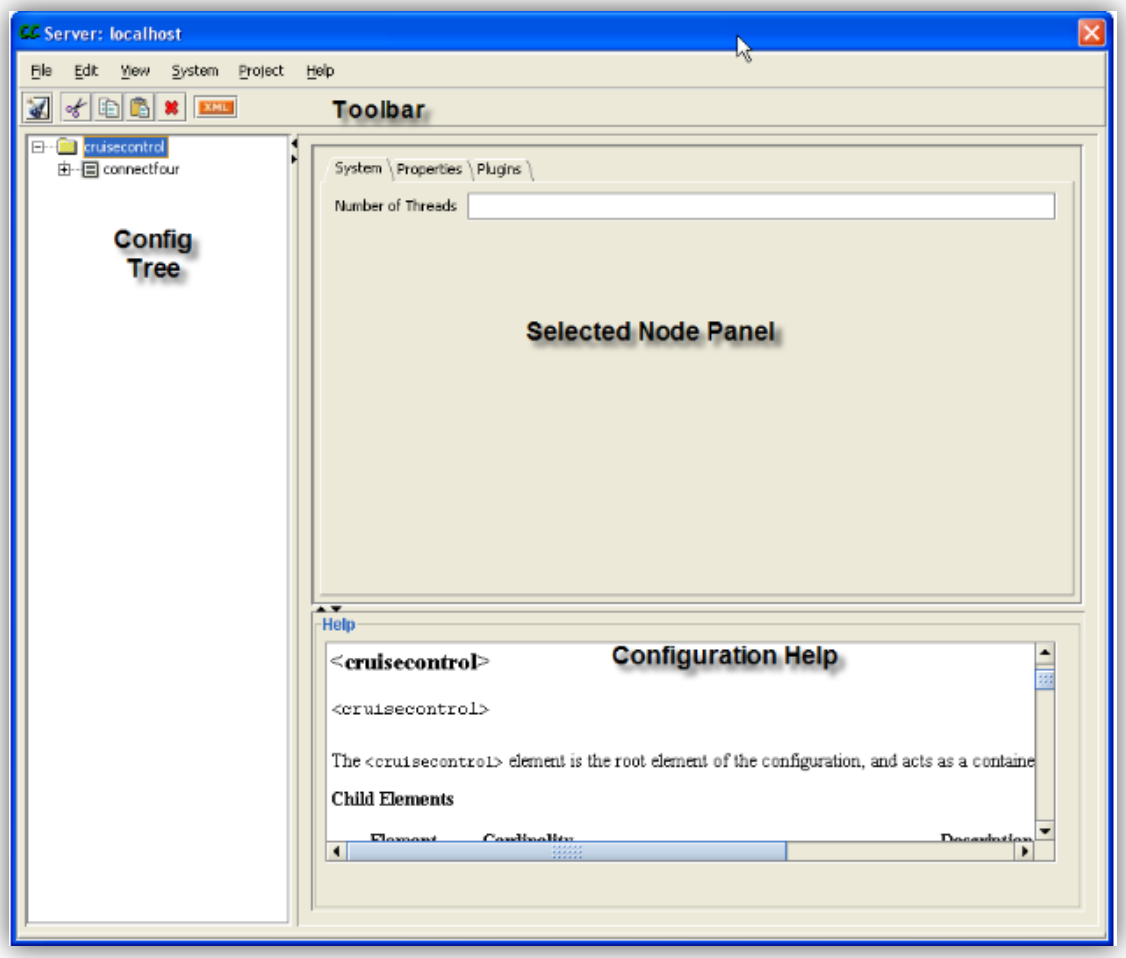
Tablero: es una sencilla interfaz que ayuda a visualizar el conjunto de proyectos que están presentes y detalles de los mismos como nombre, servidor, estado, tiempo desde la última construcción, etc.

Figura 18 – Tablero de Cruise Control.

La configuración de Cruise Control está centrada en un archivo denominado *config.xml*. En este, se encuentra toda la información necesaria como propiedades, atributos, plugins, proyectos y elementos de representación, lo cual indica que apenas se realizan configuraciones a través de su interfaz web. El resultado de esto es una interfaz sencilla, pero dependiendo de la cantidad de proyectos incluidos, se puede llegar a tener un archivo xml demasiado extenso. Para una gestión más simple de esto se facilita la herramienta “cc-config”

la cual permite una visualización y gestión del archivo xml más sencillas como se muestra en la *Figura 19*.

Figura 19 – Interfaz del estor de configuración cc-config.



Cruis Control es apta para los sistemas operativos Windows y Linux. Puede emplear Ant, Natn, Maven, Phing, Rake y Xcode como herramientas de construcción. Presenta flexibilidad para añadir o quitar funcionalidades ya que la mayor parte de ellas están basadas en plugins.

Los permisos necesarios para realizar unas acciones u otras sobre los proyectos dependen del rol del usuario. Existen cuatro roles:

- *Cruise* o Usuario: visualización de resultados y propuesta de mejoras.
- *Desarrollador*: contribución de soluciones para el código o la documentación.
- *Commiter*: acceso para la modificación del código.
- *Administrador*: todos los privilegios sobre los proyectos y usuarios.

Jenkins

Herramienta desarrollada en Java que actúa como servidor de integración continua. Distribuida para diferentes sistemas operativos e integrable en entornos como Eclipse, Netbeans e IntelliJ IDEA.

Abarca gran variedad de herramientas y lenguajes de programación:

- Proyectos desarrollados en Java, .Net, PHP, PL/SQL, Phyton, C++, Drupal…
- Control de versiones Git, Subversion, Visual SourceSafe, ClearCase, Bazaar ...
- Procesos de compilación con Ant, Maven, MSBuild y Gradle entre otros.

Uno de sus puntos fuertes se basa en su extensibilidad, pudiendo añadir nuevas funcionalidades y compatibilidades con otros servicios a base de plugins.

La configuración de los proyectos puede realizarse de forma sencilla mediante la interfaz que presenta. Permite, entre otros, configurar procesos como: obtención de código de diversos repositorios, automatización de comandos, análisis estático de código y notificación a los usuarios a través de correo. Todo ello apoyado de líneas de ayuda. El lanzamiento de las ejecuciones puede realizarse manualmente desde la interfaz o programarse para que se realice tras otra acción o en un instante en el tiempo. Si el proceso falla en alguno de los pasos, Jenkins informa al usuario sobre el error.

Las tareas, también denominadas jobs, pueden organizarse en vistas. Cada vista muestra una lista de tareas, de esta forma, cuando el número de tareas es demasiado grande, se facilita el trabajo con los proyectos.

Respecto a la seguridad de Jenkins se distinguen dos niveles: seguridad a nivel de herramienta y seguridad a nivel de proceso. Todos los usuarios acceden al sistema mediante autenticación, aunque esto debe configurarse puesto que Jenkins, por defecto, permite total acceso a cualquier usuario. Cada proyecto debe tener asignado un propietario el cual puede dar acceso a otros usuarios o grupos de usuarios para que puedan participar en él. La capacidad de participación estará condicionada por los permisos que tengan asignados casa usuario. Existen tres niveles de permisos:

- Administrador: acceso total al proceso y a su configuración. Habilita el acceso a los miembros del grupo del proyecto.
- RW (lectura y escritura): visualizar y ejecutar procesos.
- R (sólo lectura): visualizar el proceso.

Tabla 4 – Características de los cinco servidores de integración continua mencionados.

Servidores	Soporte	Sistema Operativo	Lenguajes	SCV	Builders	Entornos integración	Interfaz	Notificaciones	Últimas versiones
continuum	Apache	Windows Linux OS x	Java	Git SubversionSVN Bazaar Perforce	Ant Maven	No hay datos	Interfaz web	Disponible	1.4.2 (13 Jun 2014)
Solano	Solano Labs	Linux	Java PHP Phyton Ruby C C++	Git Mercurial Azure Github	Ant Maven Gradle Android	No es integrable	Interfaz web para configuraciones básicas. Interfaz CLI.	Disponible	1.28.4 (20 Abr 2016) Versión de gem
Bamboo	Atlassian	Windows Linux OS x	Java .Net PHP Ruby	Subversion Git Mercurial Perforce	Ant maven MSBuild Visual Studio PHPUnit	Eclipse IntellJ IDEA Visual Studio	Interfaz CLI.	Disponible	5.11 (28 Abr 2016)
cruisecontrol	ThoughtWorks	Windows Linux	Java .Net	Subversion	Ant Maven Xcode Nant Rake	Eclipse	Interfaz web para configuraciones básica.	Disponible	2.8.4 (15 Sep 2010)
Jenkins	Comunidad de desarrolladores. Autor: Kohsuke Kawaguchi	Windows Linux OS x	Java .Net PHP Phyton C++	Git SubversionSVN Bazaar	Ant Maven MSBuild Gradle	Eclipse NeatBeans IntellJ IDEA	Interfaz web para todo tipo de configuraciones.	Disponible	1.651.1 (14 Abr 2016)

Capítulo 3

Elicitación y Análisis de requisitos.

El proceso de extracción de requisitos se ha llevado a cabo a través de reuniones con los denominados 'stakeholders' o participantes en el proyecto, es decir, cliente, desarrolladores técnico, encargados y programadores.

El desarrollo software del proyecto se ha llevado a cabo en dos plataformas distintas: *Composer,* para los proyectos relacionados con la configuración y desarrollo de los objetos que conforman la base de datos, y *Eclipse* para los proyectos destinados a la funcionalidad de la aplicación. Ambos con el objetivo de customizar la aplicación web *Webtop.*

La construcción de los proyectos de *Eclipse* se realiza mediante Maven, la cual incluye el proceso de compilación y empaquetado. En caso de que sea necesario se deberán corregir y establecer los pom.xml comunes para todos los entornos de trabajo.

La construcción de los proyectos de *Composer* se realiza mediante Ant, porque esta reconoce una serie de tareas específicas de *EMC* y *Composer*. La construcción incluye el proceso de compilación, empaquetado y generación del fichero de instalación DAR. Además para estos proyectos se llevará a cabo el despliegue del DAR. Se deberán crear los ficheros xml de Ant para la construcción y despliegue automáticos.

Todos estos procesos deberán realizarse de forma automática mediante el servidor Jenkins, además de la necesidad de generar informes de calidad del software para establecer un control y corrección en el caso de que sea necesario.

- Requisitos no funcionales.
  1. El servidor deberá tener un job para cada componente que conforma el proyecto software.
  2. El servidor deberá contener una vista para cada plataforma de desarrollo empleada, es decir, en una vista se ubicarán los proyectos desarrollados en *Eclipse* y en otra vista se ubicarán los proyectos desarrollados en *Composer*.
- Requisitos funcionales.
  1. Todos los proyectos que conforman el desarrollo software del trabajo deberán construirse de forma automática.
  2. Los proyectos desarrollados en *Composer* deberán poder ser instalados de forma automática en la base de datos correspondiente.
  3. El servidor deberá obtener la última versión funcional de SubversionSVN para obtener el contenido con el que debe trabajar.
  4. El servidor deberá ejecutar todos los job periódicamente de forma automática.
  5. El servidor deberá ejecutar los job que conforman la vista de *Eclipse* en el orden necesario para que se respeten las dependencias entre componentes.

6. El servidor deberá generar informes de calidad de software para cada job tras cada ejecución.

- Requisitos de producto.

1. El servidor deberá tener instalada la versión 1.6_0_27 para el JDK.
2. El servidor deberá tener instalada la versión 1.7 para Ant.

- Requisitos externos.

1. Los desarrolladores implicados en el proyecto deberán actualizar sus entornos locales con la última versión funcional de *SubversionSVN* al menos una vez al día, en el comienzo de su jornada laboral, antes de realizar nuevos cambios.
2. Los desarrolladores implicados en el proyecto deberán realizar 'commit' de sus cambios locales a *SubversionSVN* al menos una vez al día antes de finalizar su jornada laboral.
3. Los desarrolladores implicados en el proyecto deberán corregir de forma inmediata los errores generados en la rama de desarrollo en caso de que se produzcan, es decir, en todo momento la rama de desarrollo principal debe ser funcional.

Capítulo 4

1. Implantación de la Integración Continua con Jenkins.

El método de integración continua es conveniente implantarla para cualquier tipo de desarrollo software, pero para proyectos de grandes dimensiones con un número alto y variable de desarrolladores es realmente imprescindible si se desea obtener un producto de calidad en el menor tiempo posible.

En este trabajo se ha implantado la integración continua sobre un proyecto de gestión documental. Este está basado en la plataforma de *Documentum 6.7* y su objetivo es la personalización de una aplicación para una gran empresa. Está desarrollado en Java y para su composición se han utilizado las siguientes herramientas: *Eclipse Kepler* , Documentum *Composer 6.7 SP1*, *Subversión SVN* y *Maven 4.0.0*, entre otras. Todas ellas detalladas en apartados anteriores.

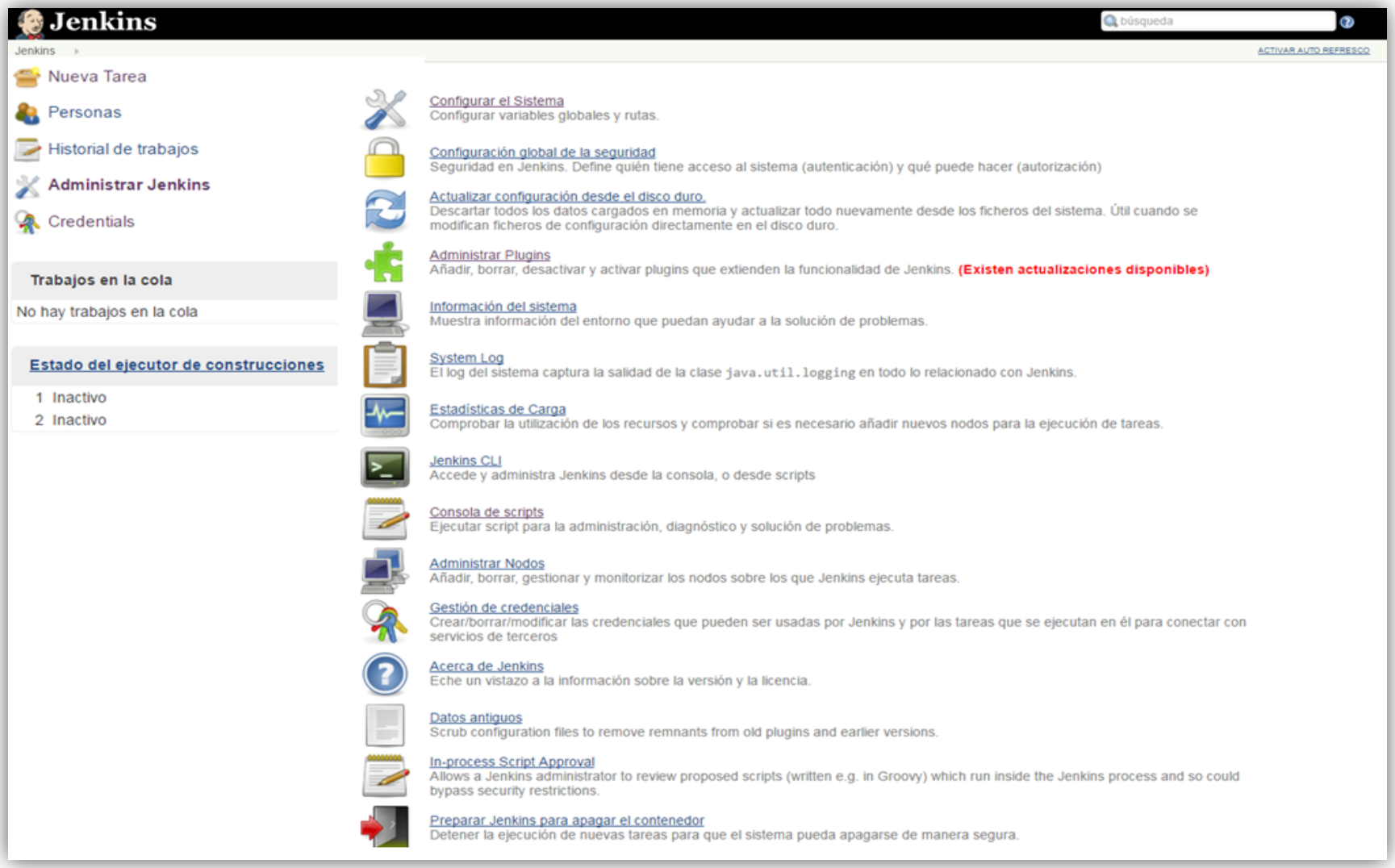
Documentum Webtop es el cliente web con el que se accede a las funcionalidades implementadas en el proyecto, por ello, gran parte de los módulos que lo forman están destinado a la customización de esta aplicación. El software desarrollado hasta el momento ha concluido en diecinueve componentes, catorce de ellos realizados en *Eclipse* y diez en *Composer*.

El empleo de una herramienta de control de versiones como *Subversion* ya es un inicio para llevar a cabo el proceso de integración continua, siempre y cuando la coordinación entre desarrolladores se realice de forma adecuada. Sin embargo es necesario incluir un servidor para poder explotar por completo los beneficios que aporta esta técnica. La simplicidad de la interfaz web, la cantidad de funcionalidades que ofrece y la continua actualización de la herramienta son alguno de los factores por los que se ha elegido Jenkins como servidor de integración continua.

2. Configuraciones generales de Jenkins

En este apartado se van a mostrar las competencias y configuraciones que Jenkins pone a disposición del usuario. Existen cuatro bloques importantes que se deben entender para utilizar el servidor de forma correcta: Seguridad, Sistema, Nodos y Plugins.

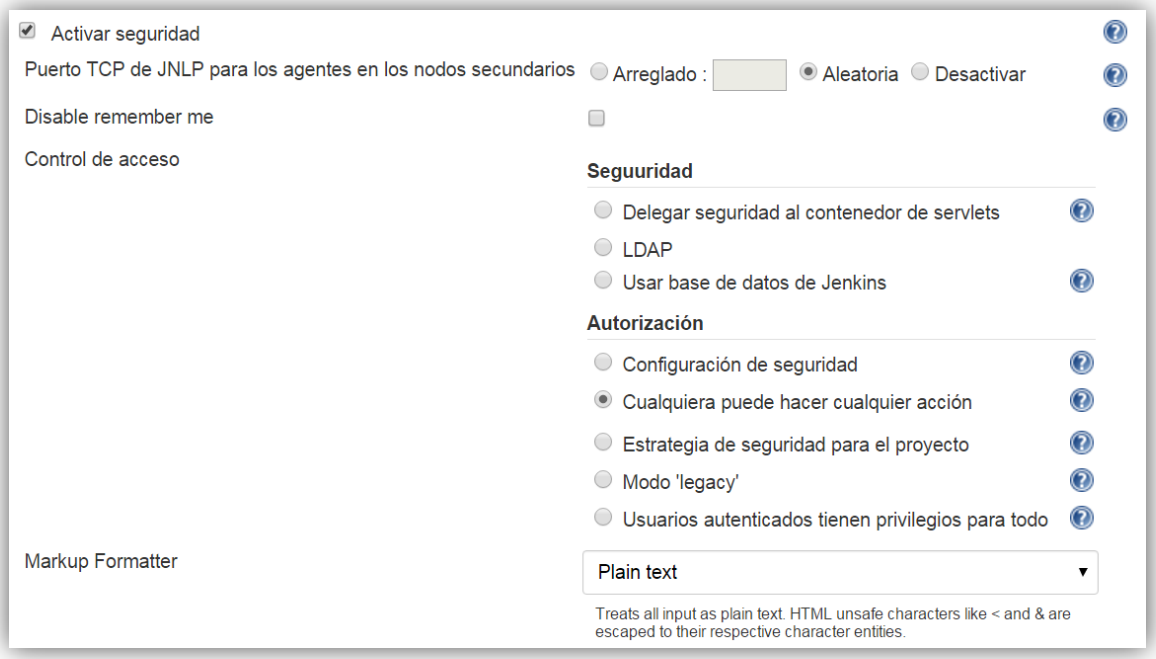
Figura 20 – Interfaz principal para la administración de Jenkins.



2.1. Seguridad

El acceso a Jenkins se realiza vía web introduciendo la URL correspondiente a la ubicación del servidor. Por defecto, este es público, por ello es recomendable priorizar la configuración de la seguridad para establecer unos credenciales de acceso y controlar los permisos administrativos. Como se puede observar en la siguiente *Figura 21*, activar la seguridad implica detallar aspectos como:

Figura 21 – Interfaz para activar la seguridad en Jenkins.



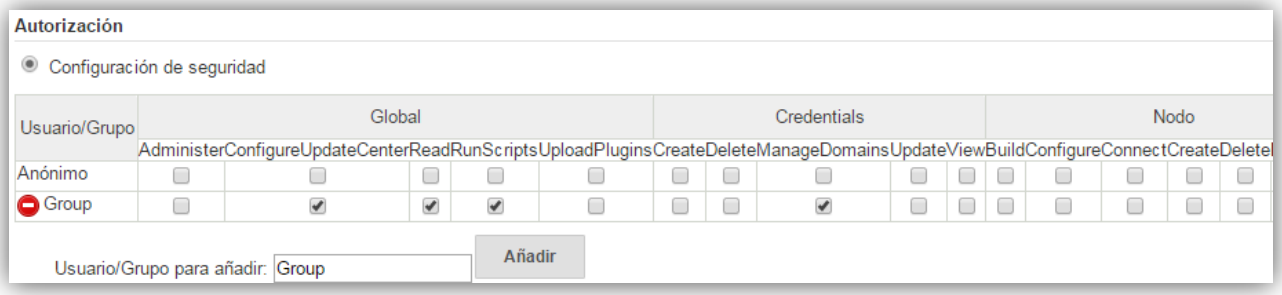
- Puerto TCP: la elección del puerto suele elegirse de forma aleatoria para evitar colisiones, pero también es posible fijar un puerto concreto para obtener mayor seguridad. Si no se utiliza JNLP esta función debe deshabilitarse.

- Control de acceso: La delegación de seguridad a un contenedor de servlets es útil si existen usuarios o grupos autenticados en otras aplicaciones que se encuentran en el mismo servidor.
  En el caso de tener un servidor LDAP es posible utilizarlo para autenticar a los usuarios introduciendo los datos necesarios para que Jenkins pueda conectar con él.
  El acceso también puede verificarse a partir de la lista de usuarios de Jenkins[2] evitando así dependencias externas.

[2] La línea de ayuda de Jenkins describe esta lista como la formada por las identidades de acceso que se pueden enumerar en el dominio de seguridad actual, así como las personas mencionadas en los mensajes de confirmación en los registros de cambios registrados.

- Autorización: en las opciones "*Configuración de seguridad"* y "*Estrategia de seguridad por proyecto"* se presenta una matriz similar a la de la *Figura 22* en el que podemos indicar qué tipo de permisos y a quien (usuario o grupo) se los asignamos.

Figura 22 – Matriz de asignación de permisos a usuarios Jenkins.



- Markup Formatter: HTML puro y texto plano son las dos opciones que disponemos para la estructuración de los archivos.

Jenkins también permite prevenir ataques *Cross Site Request Forgery (CSRF)*[3].

2.2. Credenciales

Todos los credenciales necesarios tanto para el acceso al servidor como para el acceso a herramientas con las que tiene que interactuar, deben estar registrados en Jenkins. Para ello en el panel de control principal (*Figura 20*) se encuentra el apartado 'Credenciales', donde se deben introducir los datos necesarios para la autenticación, como por ejemplo la dirección a la que hace referencia y el tipo de autenticación. Es aconsejable que, una vez finalizada la configuración de seguridad, este apartado sólo esté disponible para el administrador del servidor.

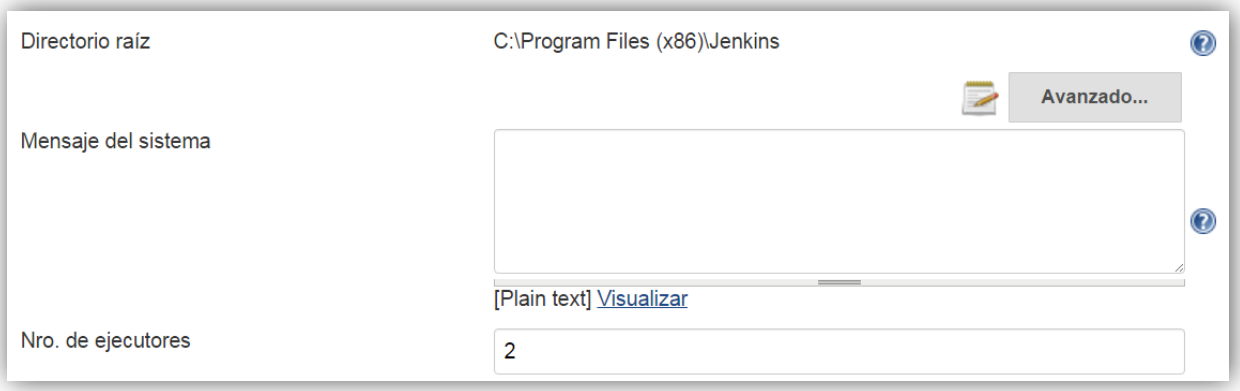
[3] Técnica de falsificación que fuerza al navegador web utilizado por el usuario que hace uso de algún servicio, a enviar una petición maliciosa aparentemente lanzada por el usuario logueado.

2.3. Sistema

En la administración del sistema, además de configurar parámetros referentes a la propia aplicación de Jenkins, es dónde se encuentra la configuración de las herramientas que se van a dar uso en los proyectos. Las configuraciones del servidor son las que se pueden ver en las siguientes figuras, como por ejemplo:

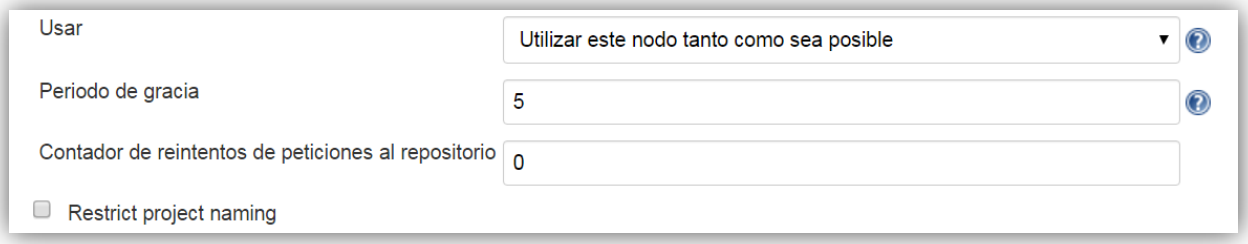
- Ubicación de la aplicación.
- Número de ejecutores.

Figura 23 – Interfaz para configuraciones básicas del sistema.



- Uso: indica el momento en el que Jenkins puede proceder a ejecutar un proyecto, esto puede ser en cualquier momento en el que la aplicación este libre o cuando el proyecto cumpla ciertos requisitos para ser ejecutado.
- El tiempo de espera antes de la ejecución de un proyecto con el fin de evitar fallos o colas de espera.

Figura 24 – Interfaz para la configuración básica del sistema.

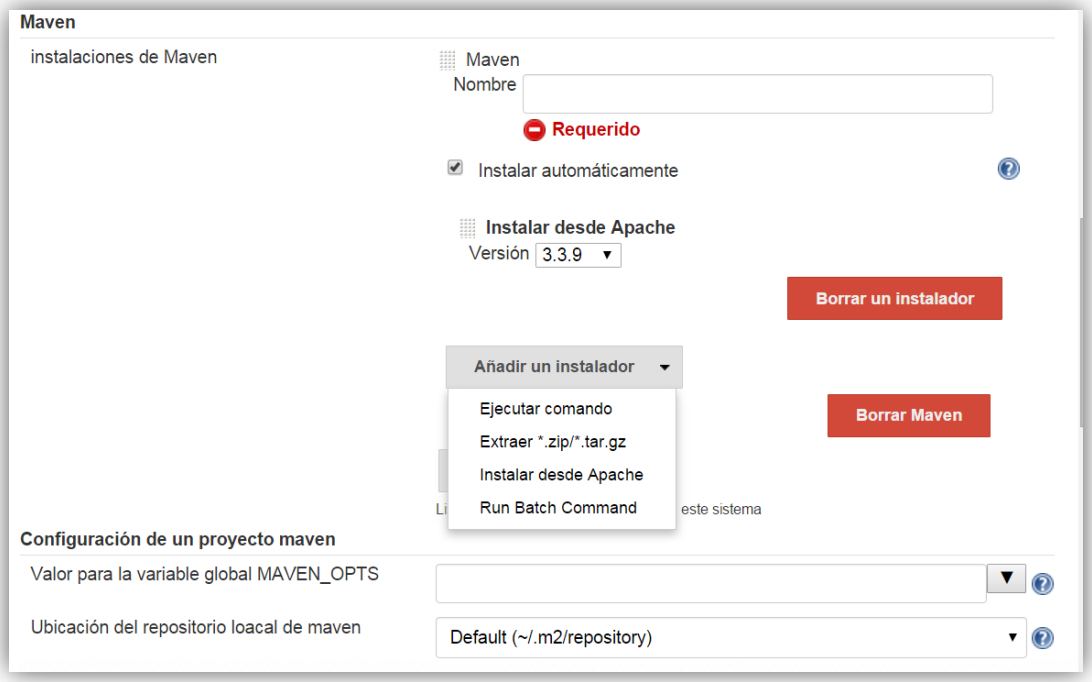


- Dirección web de Jenkins.
- Email del administrador para notificación de sucesos.

A continuación se encuentra la sección donde el usuario tiene la opción de configurar las herramientas de construcción. Puesto que vienen instaladas por defecto, se visualizan en la interfaz de configuración Maven y Ant pero, en el caso de que se instale otro tipo de herramienta, también habría un apartado reservado para ellas.

La instalación de Maven puede llevarse a cabo de tres formas diferentes: por medio de comandos, mediante archivo comprimido o a través de Apache. En cualquiera caso, como se muestra en la *Figura 25*, el usuario sólo debe seleccionar la opción deseada y posteriormente indicar los datos demandados como la versión y la ubicación del repositorio local de Maven.

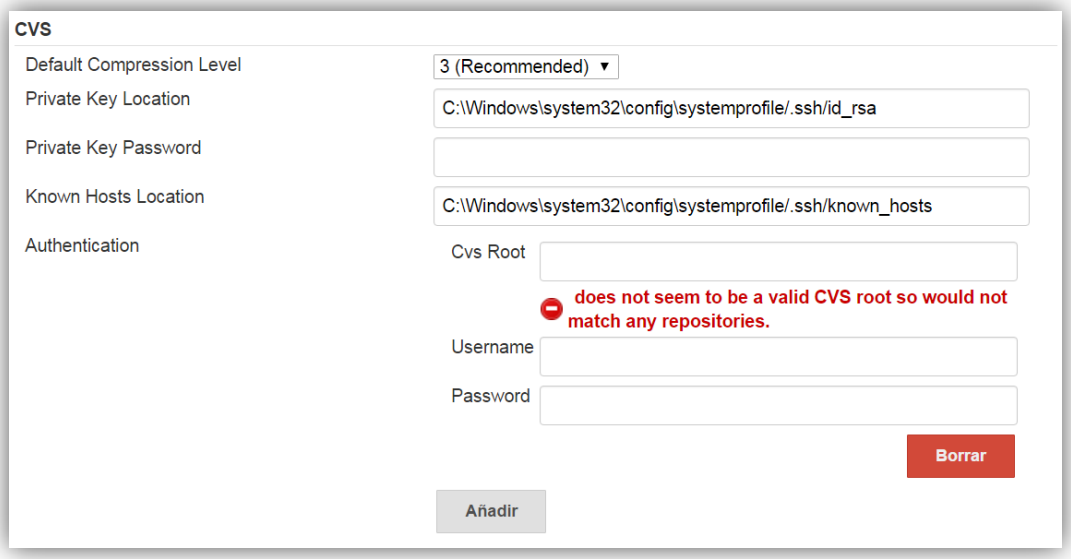
Figura 25 – Interfaz para la configuración de Maven en Jenkins.



Es recomendable utilizar la misma versión de JDK, o al menos la más cercana, a la que se ejecuta en el servidor del proyecto. Si este apartado no es configurado Jenkins utiliza la misma versión de Java sobre la que él se ejecuta, pero esto puede causar problemas de compatibilidad respecto al servidor de producción donde termina desplegándose el proyecto. La instalación de JDK también es a elección del usuario, al igual que en el caso de Maven, pero si se desea instalar JDK desde *java.sun.com* es necesario añadir lo credenciales de una cuenta Oracle.

Por último se encuentra la configuración de la herramienta de control de versiones (CVS) a utilizar. Especificando la ubicación y los credenciales como se indica en la *Figura 26* es posible disponer de varios CVS configurados en Jenkins.

Figura 26 – Interfaz para la configuración de CVS en Jenkins.



2.4. Administración de Plugins

La funcionalidad de Jenkins está basada en plugins, por ello, vienen instalados por defecto algunos de ellos, como por ejemplo: *CVS plug-in, LDAP Plugin, Credentials Plugin, SSH Slaves plugin, Maven Integration Plugin*, etc.

El usuario puede ver toda la información de los plugins instalados así como actualizar a nuevas versiones. También hay a su disposición una lista con los plugins existentes en el repositorio central. Jenkins sólo muestra las versiones compatibles con la del servidor, es decir, en ocasiones la versión indicada de un cierto plugin puede no ser la última existente.

Además de los plugins que ofrece el servidor, el usuario tiene la posibilidad de crear plugins personalizados. Tras el desarrollo del plugin se obtiene un fichero con extensión *.hpi* el cual deberá cargarse e instalarse en el servidor Jenkins para poder explotar su funcionalidad. Estos últimos dos pasos se lleva a cabo de forma sencilla mediante la interfaz web, concretamente, en 'Opciones Avanzadas' de la 'Administración de Plugins'.

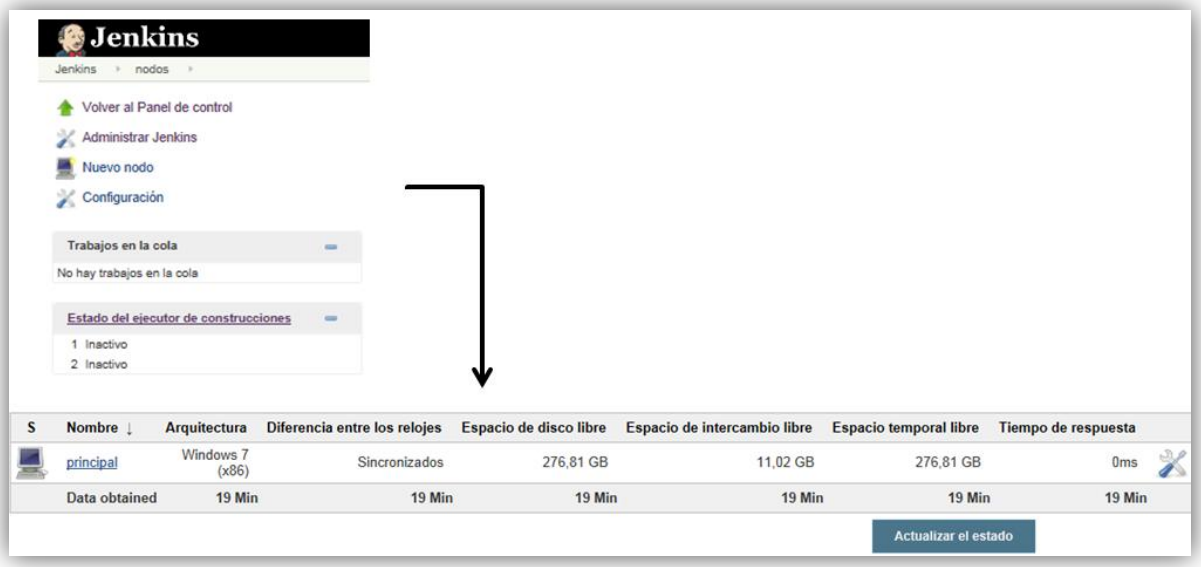
Existen entornos en los que el servidor no tiene conexión directa con Internet porque se encuentra detrás de un cortafuegos, por ello también es posible configurar el proxy.

Figura 27 – Interfaz para la configuración del proxy en Jenkins.

2.5. Administración de Nodos.

La ejecución de las tereas en Jenkins se lleva a cabo sobre nodos. Por defecto utiliza como nodo de ejecución la máquina sobre la que tenemos montado el servidor. Como ejemplo, en la *Figura 28* se muestra el apartado de 'Administrar nodos' de un servidor Jenkins montado en un entorno local. Normalmente se realiza una ejecución tras otra, formando una cola de trabajos en el caso de lanzar varias a la vez, pero es posible configurar varios ejecutores para realizar ejecuciones en paralelo. Para cada nodo configurado puede restringirse su uso sólo para tareas vinculadas a él o para cualquier tipo de tareas, aprovechando así al máximo su disponibilidad.

Figura 28 – Configuración de nodos en Jenkins.



3. Configuración de proyectos en Jenkins

Una vez configurados los parámetros generales del servidor, se procede a crear las tareas, también denominadas *job* en Jenkins, que corresponderán a los componentes que forman el proyecto. Posteriormente será necesario configurarlos para adaptarlos a la estructura y especificaciones de nuestro trabajo.

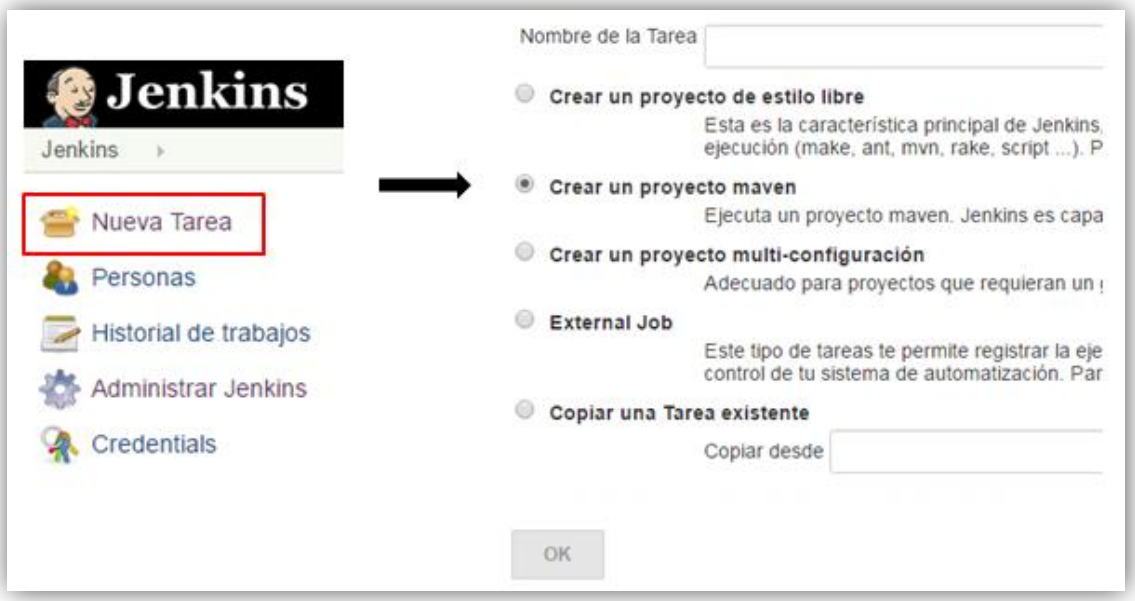
3.1. Creación de tareas.

Jenkins ofrece cuatro opciones diferentes para la creación de tareas en base a las características del componente, con el objetivo de reducir lo más posible las configuraciones sobre ellas. Estas opciones son:

- Proyecto de estilo libre: combinación de diferentes repositorios y métodos de construcción.
- Proyecto Maven: obtiene parte de la configuración directamente de los ficheros *pom.*
- Proyecto multi-configuración: para proyectos que deban probarse en diferentes entornos o ejecutarse en plataformas concretas.
- Job externo: permite registrar la ejecución de un proyecto externo a Jenkins.

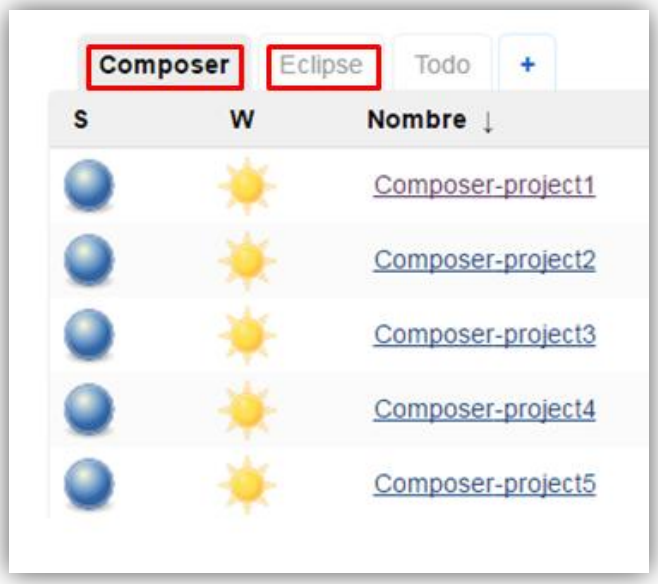
La mejor opción para la integración de las tareas que componen nuestro trabajo son proyectos Maven, puesto que parte de los proyectos desarrollados ya contienen sus propios ficheros *pom.* Una vez se tiene claro el tipo que se le va a asignar a la tarea, se procede a crear para cada componente un job. En la interfaz web de Jenkins, situados en el menú principal del usuario, se selecciona la opción 'Nueva tarea' y en la ventana a la que nos dirige se introduce el nombre y el tipo del proyecto.

Figura 29 – Creación de una tarea en Jenkins.



Puesto que necesitamos crear un total de diecinueve jobs, que son el total de componentes software que forman el proyecto, será necesaria la creación de dos vistas diferentes para obtener una perspectiva más clara y organizada. De manera que en la vista denominada 'Eclipse' tendremos los proyectos desarrollados en *Eclipse* y en la vista denominada 'Composer' tendremos los proyectos desarrollados con *Composer*.

Figura 30 – Vistas de Jenkins.



Tras la creación de todas las tareas, la interfaz de Jenkins se presenta al usuario de forma similar a la de la *Figura 31.*

Figura 31 – Vista principal de los componentes del proyecto en Jenkins.

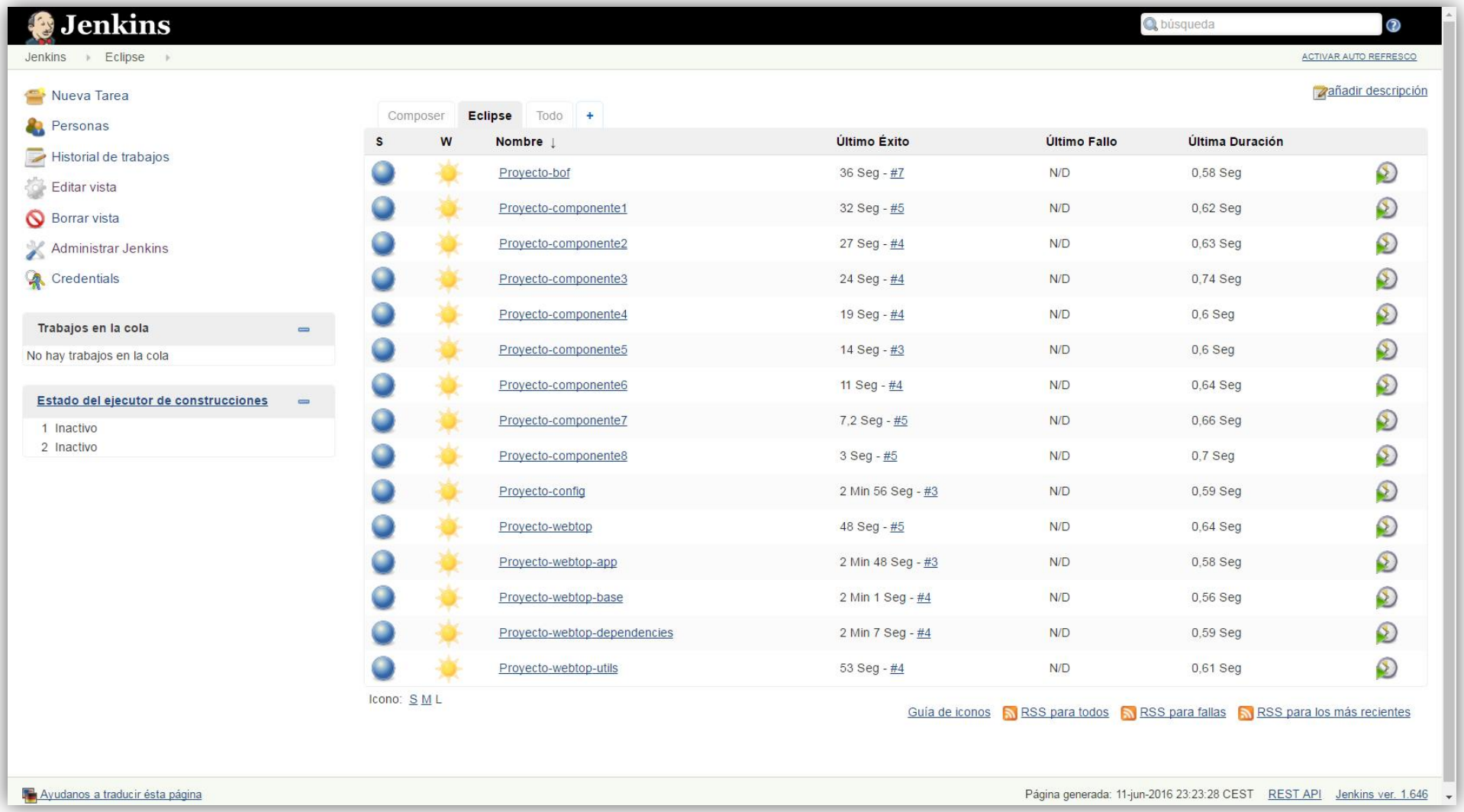
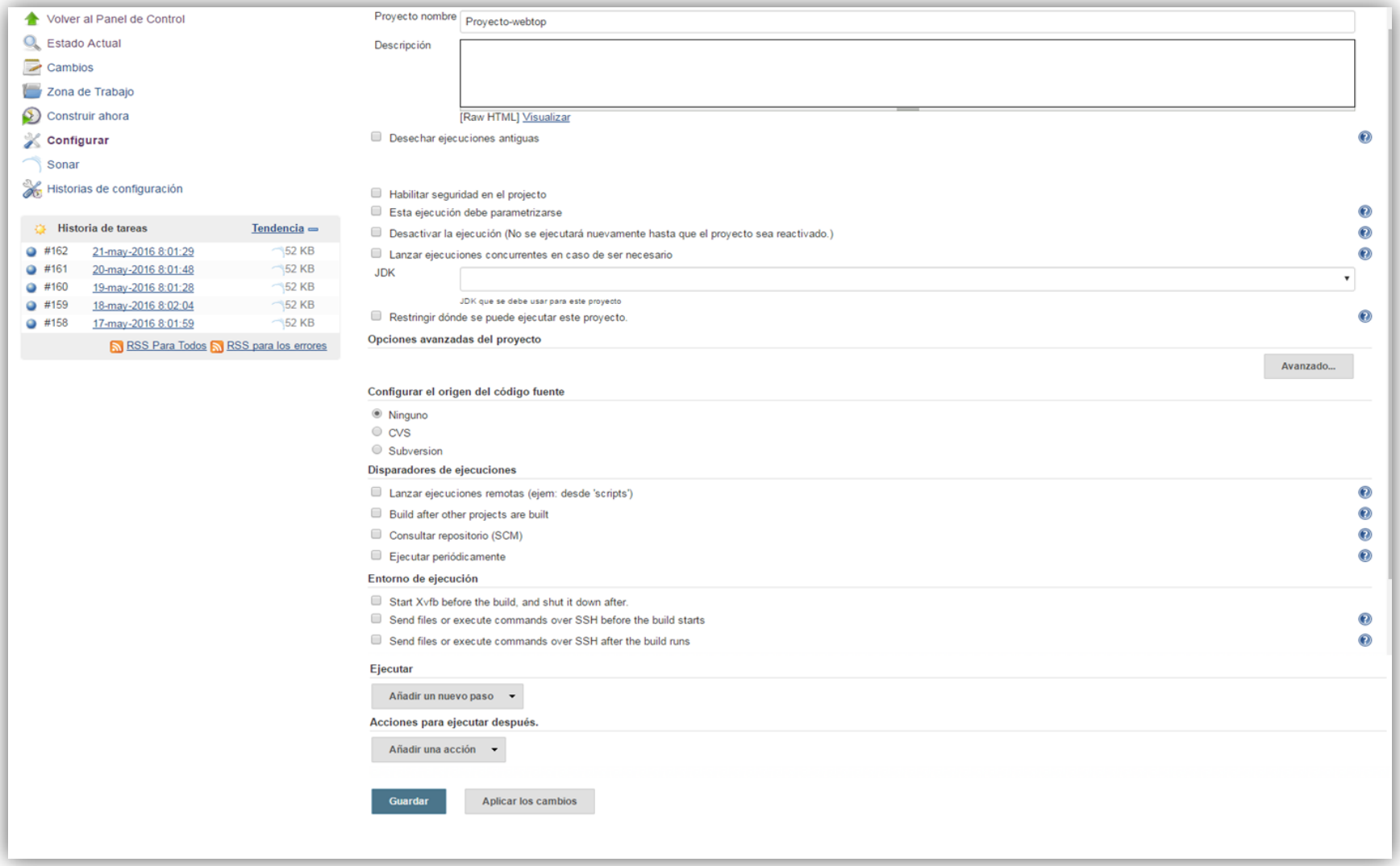


Figura 32 – Interfaz de configuración básica de un job en Jenkins.



3.2. Configuración de tareas.

La interfaz de configuración de las tareas varía según el tipo de proyecto del que se trate. En la *Figura 32* se muestra la interfaz con los puntos básicos de configuración para que el lector pueda hacerse una idea de la secuencia de pasos que se explican a lo largo de este apartado. Para cada tarea se van a configurar los siguientes puntos:

- Historial de tareas.
- Permisos de usuario.
- Versión JDK.
- Origen del código fuente.
- Disparador y orden de ejecuciones.
- Herramienta de construcción.
- Herramientas complementarias como Sonar.

Las configuraciones relacionadas con la ejecución y la construcción del proyecto van a variar en función de si el proyecto es de *Eclipse* o de *Composer*, pero exceptuando estas, el resto de puntos serán iguales en todos los job.

- Historial de tareas.

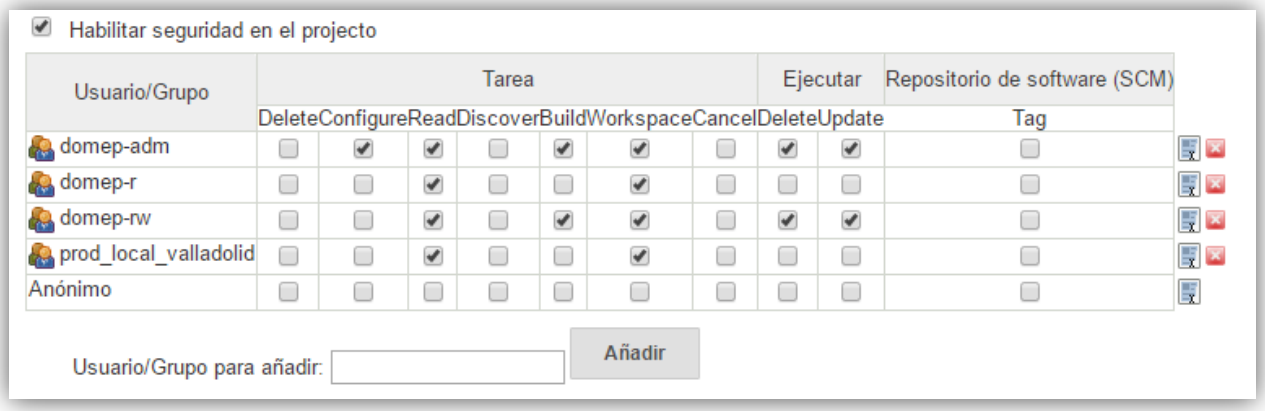
Jenkins almacena el resultado de las ejecuciones llevadas a cabo hasta la fecha. Cada job tiene un historial de tareas de donde se puede obtener, por cada ejecución: fecha y hora de la misma, resultado por consola o en texto plano y los cambios del repositorio respecto a la ejecución previa. En la configuración se permite al usuario desechar ejecuciones antiguas y elegir el número de ejecuciones que desea guardar. En nuestro proyecto se guardarán las cinco últimas ejecuciones de cada job.

Figura 33 – Historial de Tareas de un job Jenkins y salida por consola.

- Permisos de usuario.

Existe una funcionalidad en Jenkins aportada por el plugin 'Matrix Authorization Strategy' que permite al administrador restringir el acceso total o parcial al job a los diferentes usuarios de Jenkins. Este plugin facilita una matriz en la que se pueden añadir los usuarios que deben trabajar con el proyecto y seleccionar las tareas que pueden llevar a cabo.

Figura 34 – Matriz de estrategia de autorización.



- Versión de JDK.

La versión JDK utilizada en este proyecto es la 1.6_0_27 porque es la que requiere *Documentum 6.7* para el desarrollo de la aplicación *Webtop*. Tras la instalación del JDK en las configuraciones generales del sistema (especificado en el apartado 2.3) en el job solo es necesario seleccionar la versión.

- Orden de ejecución.

En Jenkins las ejecuciones pueden programarse tomando diferentes variables:

- Ejecución del job en una fecha y hora concretas.
- Ejecución del job diariamente a una hora concreta.
- Ejecución del job tras la última actualización del código, es decir, tras el 'update' en el sistema de control de versiones.
- Ejecución del job tras la ejecución de otros proyectos.
- Ejecución remota mediante scripts.

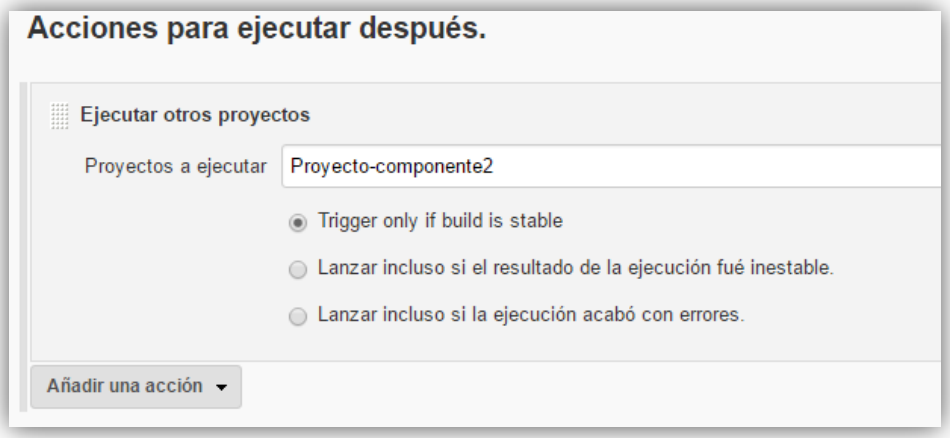
Los componentes del proyecto desarrollados en *Eclipse* necesitan llevar un orden concreto de ejecución puesto que unos componentes dependen de librerías o funcionalidades de otros. El orden es el siguiente:

1. Proyecto-webtop-dependencies.
2. Proyecto-webtop-base.
3. Proyecto-custom.
4. Proyecto-webtop-utils.

5. Proyecto-config.
6. Proyecto-bof.
7. Proyecto-componente1.
8. Proyecto-componente2.
9. Proyecto-componente3.
10. Proyecto-componente4.
11. Proyecto-componente5.
12. Proyecto-componente6.
13. Proyecto-componente7.
14. Proyecto-componente8.

Por ello el disparo de ejecución de todos los job, excepto del job correspondiente a 'Proyecto-webtop-dependencies', será tras la construcción del proyecto antecesor. Como ejemplo vamos a tomar el 'Proyecto-componente1' y el 'Proyecto-componente2'. Primero es necesario añadir un nuevo paso en la configuración, para ello se despliega la pestaña 'Añadir una acción' y se selecciona la opción 'Ejecutar otros proyectos'. Seguidamente aparecerá una nueva sección para indicar qué proyecto o proyectos se desean ejecutar tras la construcción del 'Proyecto-componente1' y en qué circunstancias. En nuestro caso indicamos que el job a ejecutar será el 'Proyecto-componente2' sólo si la construcción del 'Proyecto-componente1' ha sido estable. Finalmente la configuración se muestra como en la *Figura 35*.

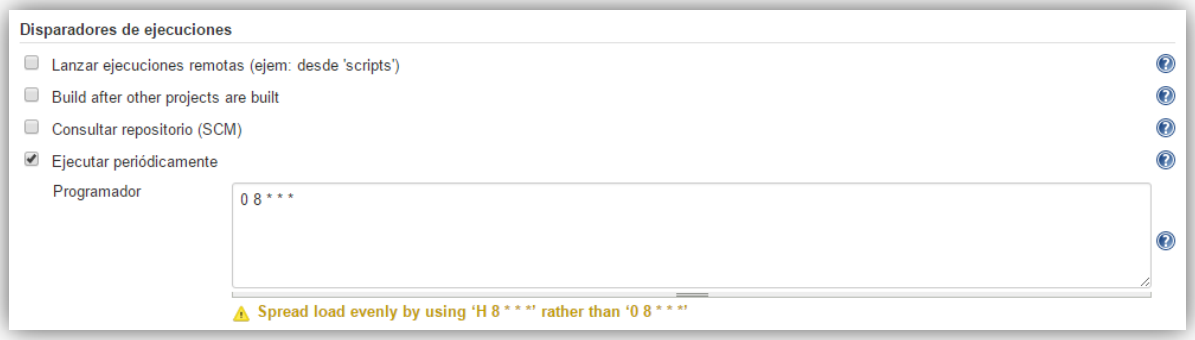
Figura 35 – Disparo de ejecución en Jenkins.



El job correspondiente al proyecto 'Proyecto-webtop-dependencies' es el que desencadena la ejecución del resto de proyectos por ello en su configuración, además de indicar que tras su construcción se dispare la ejecución de 'Proyecto-webtop-base', se debe especificar el momento en el que deseamos que comience la construcción de este. En nuestro caso es conveniente comprobar la estabilidad del proyecto completo diariamente, por lo que en la sección 'Disparadores de ejecuciones', seleccionamos la opción 'Ejecutar periódicamente'. En el cuadro de programador que podemos ver en la *Figura 36* hay que introducir una sentencia para indicar el periodo de tiempo en el que se deba ejecutar la tarea. El patrón que tiene esta sentencia son cinco espacios a completar, donde cada espacio corresponde, en orden:

minutos, horas, días del mes, mes y días de la semana. Nuestro proyecto queremos que se ejecute todos los días a las 8h, por lo que en el programador se introduce *0 8 * * *.* Los asteriscos marcan los parámetros que no tienen valor pero deben constar en la sentencia. También es posible introducir saltos de tiempo con el comando */, de manera que si el usuario desea que se ejecute cada X minutos por ejemplo, la sentencia sería **/X * * * *.*

Figura 36 – Programador de ejecuciones en Jenkins.



Los proyectos desarrollados en *Composer* no deben seguir un orden concreto de ejecución por lo que a cada job se le programará periódicamente para que se ejecute a las 8:30h de la mañana, es decir, en el programador introduciremos la sentencia *30 8 * * *.*

- Origen del código fuente.

El código fuente se obtiene del sistema de control de versiones, por lo que en nuestro caso se procede a la configuración de *SubversionSVN*. Puesto que ya tenemos instalado el plugin '*Subversion Plug-in'* en la interfaz de configuraciones del sistema aparece la sección 'SVN' para realizar su configuración general. En el caso de no tenerlo instalado, tendríamos que dirigirnos al 'Gestor de plugins', buscar el plugin correspondiente al control de versiones que se va a utilizar e instalarlo. Con *Subversion* incorporado en el servidor, el sistema se puede visualizar en la sección 'Configurar el origen del código fuente'>'Subversion' de cada job para su configuración individual.

Hay dos parámetros obligatorios para SVN: la URL del repositorio y la estrategia check-out. La primera indica la ubicación del código fuente y la segunda el método de actualización del mismo. El código fuente a probar se encuentra en la línea de desarrollo principal, es decir, en la rama *trunk*. Por ello, en la especificación de la URL del repositorio, apuntaremos a esta rama. En Subversión la organización de los componentes es similar a la de los entornos de desarrollo, por ello cada componente tiene su propia rama trunk en la cual se encuentran los ficheros y paquetes que forman el componente en sí. Esto implica que la URL especificada en cada job será distinta. Para finalizar la configuración se indica como estrategia check-out 'Use svn update as much as possible', es decir, la actualización del código tan pronto como sea posible.

- Herramientas de construcción.

En el desarrollo del proyecto se han utilizado dos herramientas de construcción, Maven y Ant. Los proyectos desarrollados en *Eclipse* están formados con Maven mientras que los proyectos desarrollados en *Composer* lo están con Ant.

Las funciones que desempeña Maven en nuestros componentes son de compilación y empaquetado, de manera que en cada *pom.xml* se diferencian tres partes:

- La cabecera, donde se indica los datos generales del componente junto a las propiedades.

Figura 37 – Cabecera fichero pom.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.webtop.components</groupId>
    <artifactId>webtop-componente1</artifactId>
    <version>2.1.0</version>
    <packaging>war</packaging>
    <name>Webtop componente1</name>
    <properties>
        <java.version>1.6</java.version>
    </properties>
```

- Las dependencias, en las que se incluyen los paquetes de lo que depende el componente para su correcto funcionamiento.

Figura 38 – Dependencias fichero pom.xml.

```
<dependencies>
    <dependency>
        <groupId>com.documentum.webtop.base</groupId>
        <artifactId>webtop-base</artifactId>
        <version>2.0.0</version>
        <scope>provided</scope>
        <classifier>classes</classifier>
    </dependency>
    <dependency>
        <groupId>com.webtop.utils</groupId>
        <artifactId>utils-webtop</artifactId>
        <version>2.0.0</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>com.webtop.apps</groupId>
        <artifactId>webtop-componente2</artifactId>
        <version>1.0.0</version>
        <scope>provided</scope>
        <classifier>classes</classifier>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>servlet-api</artifactId>
        <version>2.5</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

- Los plugins encargados de la compilación y empaquetado.

Figura 39 – Plugins fichero pom.xml.

```
<build>
    <sourceDirectory>./src/main/webapp/webtop/src</sourceDirectory>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-assembly-plugin</artifactId>
            <version>2.5.2</version>
            <executions>
                <execution>
                    <id>entregable-compilado</id>
                    <configuration>
                        <descriptor>project-assembly.xml</descriptor>
                        <finalName>${artifactId}</finalName>
                    </configuration>
                    <phase>package</phase>
                    <goals>
                        <goal>single</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.3.2</version>
            <configuration>
                <source>1.6</source>
                <target>1.6</target>
                <encoding>CP1252</encoding>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Jenkins realizará una instalación Maven del *pom.xml* de cada job para obtener los paquetes y disponerlos para el resto de jobs. En la interfaz de configuración de cada tarea se deben indicar la versión y los goals que vayan a ejecutarse, por lo que la sentencia para los goals en nuestro caso es *-e install.*

La construcción de los proyecto de Composer se ha realizado mediante Ant porque la herramienta contiene tareas específicas de Composer y EMC que Ant reconoce. Las tareas empleadas en el proyecto son de compilación, empaquetado, actualización e instalación.

- Actualización de los jardef que forman el proyecto <emc.importContent>.
  Los artefactos *jardef* contienen los paquetes jar necesarios para los SBO y TBO que serán configurados en los artefactos *modules*. Es necesaria su actualización para recoger las modificaciones de las clases implicadas en el proceso.

Figura 40 – Tarea Ant para la actualización de los jardef en Composer.

```
<target name="update-jardef" depends="jar"
    description="Update Jardef with most current JAR file">
    <emc.importContent dmproject="${project}"
        artifactpath="${ruta.jardef}/composer-project1.jardef"
        contentfile="${jar.dir}/paquete1.jar" />
</target>
```

- Compilación <emc.buid>.

Figura 41 – Tarea Ant para la compilación del proyecto en Composer.

```
<target name="build-project" depends="update-jardef"
    description="Build the project">
    <emc.build
    dmproject="${project}"
    failonerror="true"/>
</target>
```

- Construcción del fichero dar <emc.dar>.
  El fichero binario DAR contiene las modificaciones realizadas en todos los artefactos del proyecto. Es necesario realizar la construcción de este fichero para su posterior instalación.

Figura 42 – Tarea Ant para la construcción del DAR en Composer.

```
<target name="package-project" depends="build-project"
    description="Package the project into a DAR for installation">
    <delete file="bin-dar/composerProject1.dar" />
    <emc.dar
        dmproject="${project}"
        manifest="bin/dar/default.dardef.artifact"
        dar="bin-dar/composerProject1.dar" />
</target>
```

- Instalación del fichero DAR <emc.install>.

  La instalación del archivo DAR se puede llevar a cabo tras una previa configuración del fichero *dfc.properties* de *Composer* donde se indican los datos del repositorio en el que se desea instalar los cambios. Este fichero debe estar configurado tanto si la instalación se realiza manualmente mediante la interfaz de *Composer* como si se realiza a través de Ant.

Figura 43– Tarea Ant para la instalación del proyecto en Composer.

```
<target name="install-project"
    description="Install the project to the specified repository.">
    <emc.install dar="bin-dar/DocApp_cargador.dar"
        docbase="Repositorio"
        username="username"
        password="password" />
</target>
```

Una vez se han desarrollado todos los archivos xml con las tareas de Ant y se han actualizado las versiones de Subversión, se procede a configurar Jenkins para programar su ejecución.

En los job creados para los proyectos de *Composer*, en la sección de 'Ejecución'>'Añadir un nuevo paso' se selecciona la opción 'Ejecutar Ant'. Una vez incluido Ant sólo es necesario indicar la versión utilizada y el nombre de los ficheros que debe ejecutar.

- Sonar.

La finalidad de la integración continua también es obtener código de calidad. Sonar es una aplicación basada en la web dedicada al análisis y medición de calidad del código de un proyecto. Integra las mejores herramientas de medición de la calidad software (CPD, findbugs, PMD, checkstyle).

Su arquitectura se basa en cuatro componentes:

1. Un servidor formado a su vez de:
   - Servidor web.
   - Servidor de búsqueda.
   - Servidor de informes.
2. Una base de datos.
3. Plugins.
4. Escáneres: para analizar proyectos durante la ejecución del proceso de integración continua.

Cubre los siguientes puntos clave de calidad de software:

- Código duplicado.
- Estándares de codificación.
- Pruebas unitarias.
- Complejidad.
- Evidencias de fallos potenciales.
- Comentarios.
- Diseño y arquitectura.

Gracias a su eficiente navegación del código fuente permite obtener información sobre diversos parámetros con los que obtener métricas que permitan crear un plan de acción en el caso de que los resultados no sean óptimos. Estas métricas se almacenan en su propia base de datos y por ello es posible combinarlas. Entre las métricas que ofrece esta la *complejidad ciclomática*[4], una medida importante puesto que es independiente del lenguaje y nos permite saber el número de pruebas que se deben hacer para abarcar el 100% del código.

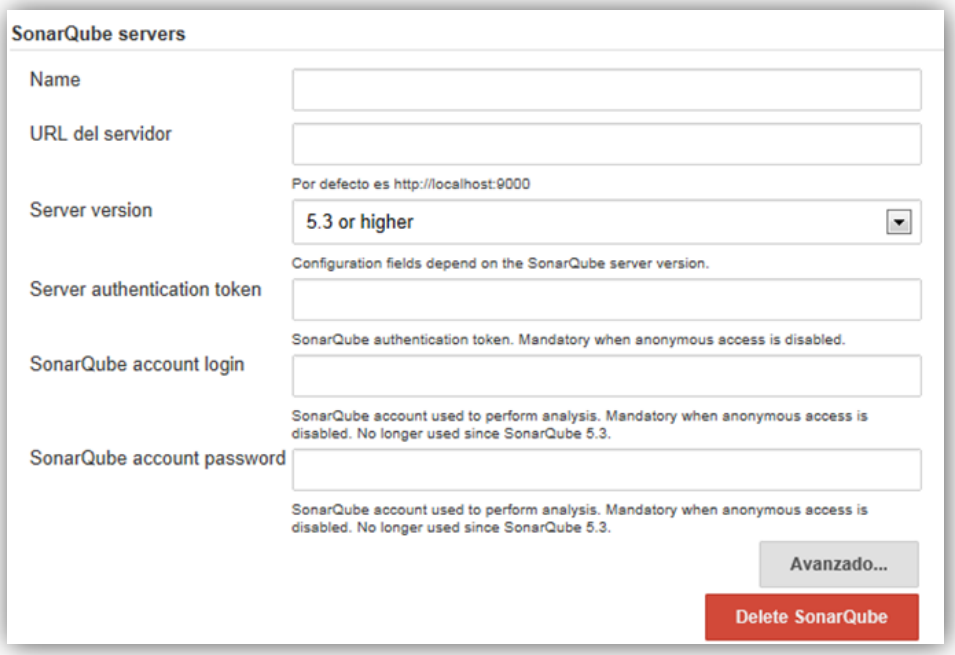
Clasifica las incidencias en base a su severidad y a su naturaleza: Fiabilidad, usabilidad, eficiencia, mantenimiento, portabilidad. Es extensible mediante plugins por lo que se puede ampliar el uso de Sonar, por ejemplo, para medir en diferentes lenguajes o realizar nuevos

[4] Métrica del software basada en grafos que mide el número de caminos que el código puede tener durante su ejecución.

tipos de métricas. Su compatibilidad con Maven y Ant permite generar un informe complementario dentro del ciclo de construcción.

Por sus características y por su posible integración tanto en Eclipse como en Jenkins, Sonar resulta una herramienta de interés para trabajos como el nuestro. Sonar es integrable con Jenkins mediante el plugin 'SonarQube'. Una vez instalado el plugin en el servidor es necesario dirigirse a las configuraciones generales del sistema (comentadas en el apartado 2.3) donde existirá una nueva sección para Sonar. En esta sección se debe preparar el servidor de Sonar para poder utilizarlo en los job. En la *Figura 44* se puede observar los datos a incluir del servidor.

Figura 44 – Configuración del servidor Sonar en Jenkins.



Posteriormente, para cada job en el que se quiera añadir la funcionalidad de Sonar, en el apartado 'Acciones para ejecutar después' > 'Añadir una acción' se puede seleccionar 'Sonar'. Una vez situados en este punto se deben añadir datos como: la instalación de Sonar a la que se hace referencia (a través del nombre que se le ha dado en la configuración del servidor), la versión de Maven que se ha utilizado en el proceso de ejecución y la versión JDK del job.

4. Ejecución de tareas.

La ejecución de un job puede realizarse en cualquier momento, ya este programado o no. En cada job existe la opción denominada 'Construir ahora', lo que permite realizar pruebas en el momento deseado. Una vez lanzado el job, este es ejecutado en el nodo correspondiente y su progreso se puede visualizar en el panel de ejecución de tareas o en la salida de consola del propio job.

Tras finalizar la ejecución existe un icono indicativo de su resultado. Si este es de color azul, indica que el proceso ha terminado correctamente, pero si es de color rojo indica que ha surgido algún error en el proceso. En el caso de no haber ejecutado nunca un job, este icono será de color gris. También existe otro icono, representado con estados meteorológicos, que indican su evolución tras varias ejecuciones del mismo job.

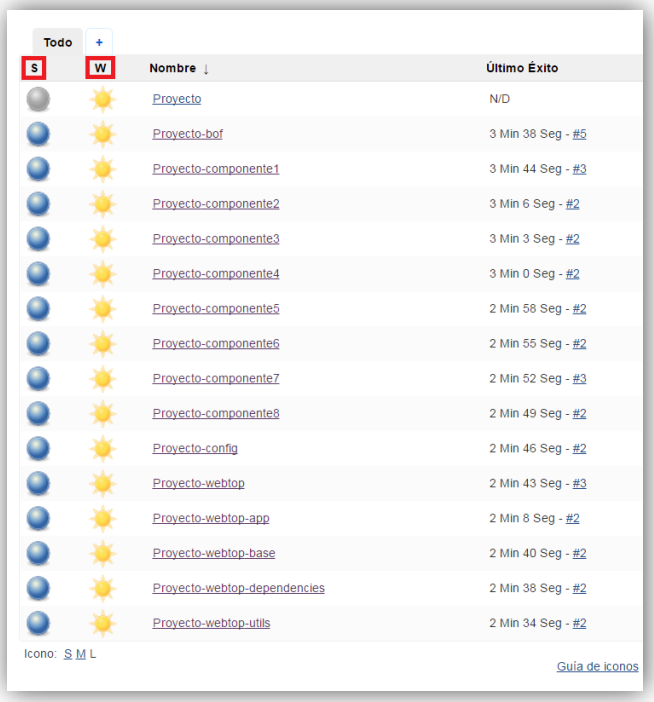
El icono soleado  indica que las últimas ejecuciones del job han progresado correctamente.

Los nubes y claros  indican que el job ha fallado en algún momento.

La tormenta  indica que las últimas ejecuciones del job han sido fallidas.

En el panel principal se pueden observar todos los proyectos con sus correspondientes iconos de estado. Las columnas **S** y **W** los representan.

Figura 45 – Panel de control Jenkins con las columnas de estados de ejecución.



5. Optimización del código.

Sonar presenta una interfaz web que permite visualizar gráficos, diagramas y estadísticas de los resultados que definen la calidad de nuestro código. Con su integración en Jenkins podemos acceder a ella con un solo 'click'. Está disponible en el panel de control de cada proyecto en el que se haya realizado su previa configuración, en nuestro caso, sobre los proyectos desarrollados en Eclipse.

Figura 46 – Panel de control de un proyecto Jenkins con Sonar incluído.

Para cada componente se ha realizado la optimización del código en base a los resultados obtenido en Sonar. Los incidencias o issues se dividen en cinco niveles: bloqueantes, críticos, mayores, menores e informativos. La corrección de los bloqueantes y los críticos es prioritaria puesto que son los que más perjudican a la calidad del código. En la *Tabla 5* se muestran algunos ejemplos de ellos.

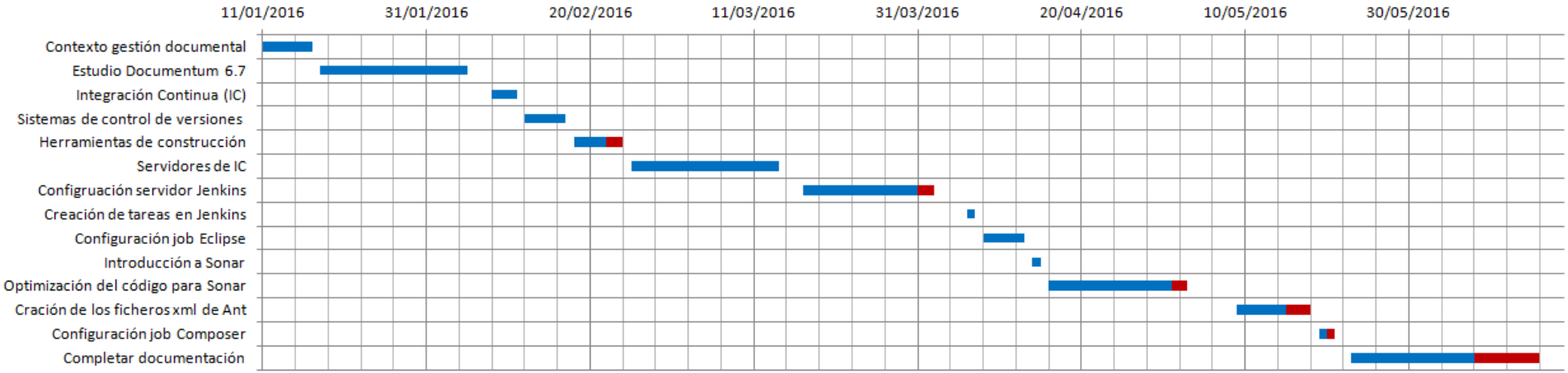
Tabla 5 – Ejemplos de issues bloqueantes y críticos en Sonar.

Nivel	Issue	Descripción
Bloqueantes	Unused import.	Importación de un paquete no utilizado.
Críticos	Private method is never called.	Método privado que nunca se utiliza.
	Nullcheck of value previously dereferenced.	Se comprueba si un valor es nulo pero este valor no puede serlo porque eliminan la referencia con anterioridad, si fuera nula se hubiera producido una excepción de la referencia con anterioridad.
	Unchecked/unconfirmed cast.	No todas las instancias del tipo se pueden convertir. Hay que comprobarlo, es decir, ver si la instancia pertenece a la clase (instanceof).
	Method uses the same code for two branches.	Código repetido en un mismo método.
	Method concatenates strings using + in a loop.	Construcción de una cadena mediante la concatenación en bucle. String-StringBuffer-String esto produce alto coste en iteraciones.
	Dead store to local variable.	Variable local no utilizada en ninguna instrucción del método.

6. Cambios en la planificación.

A lo largo del desarrollo del trabajo se han dado circunstancias que han afectado a la planificación y ha sido necesario realizar modificaciones. La planificación resultante es como la que se muestra en la *Gráfica 2.*

- Con motivo de enfermedad, la actividad 'Herramientas de construcción' se ha incrementado en 2 días siendo su duración total de 6 días.
- Con motivo de ausencia por asuntos personales, la actividad 'Configuración del servidor Jenkins' se ha incrementado en 2 días siendo su duración total de 16 días.
- Con motivo de un nuevo contrato con aumento de horas laborales, las cuatro últimas actividades se han visto afectadas.



	Completar documentación	Configuración job Composer	Cración de los ficheros xml de Ant	Optimización del código para Sonar	Introducción a Sonar	Configuración job Eclipse	Creación de tareas en Jenkins	Configruación servidor Jenkins	Servidores de IC	Herramientas de construcción	Sistemas de control de versiones	Integración Continua (IC)	Estudio Documentum 6.7	Contexto gestión documental
Fecha Inicio	23/05/2016	19/05/2016	09/05/2016	16/04/2016	14/04/2016	08/04/2016	06/04/2016	17/03/2016	25/02/2016	18/02/2016	12/02/2016	08/02/2016	18/01/2016	11/01/2016
Duración (días)	15	1	6	15	1	5	1	14	18	4	5	3	18	6
Incremento (días)	8	1	3	2				2		2				

Gráfica 2 – Diagrama de Gantt con las modificaciones tras el desarrollo del trabajo.

Conclusiones.

Todo tipo de proyecto debe tener una organización entre sus participantes y un proceso de evolución creciente y óptima para satisfacer tanto al cliente como al propio empleado. El método de integración continua fundamenta esta visión, consigue establecer el orden con prácticas basadas en el sentido común y reducir el tiempo de desarrollo evitando complicaciones innecesarias. Cierto es que implica tener conocimientos adicionales para emplear las herramientas que facilitan la integración continua, pero dedicar tiempo a formarse en esas herramientas es verdaderamente productivo puesto que se van a utilizar en cualquier tipo de proyecto software y aporta muchas ventajas a varios niveles como personales, temporales y económicos.

A lo largo de los últimos años se han desarrollado múltiples servidores para la integración continua. Los más utilizados presentan diferencias respecto a la interfaz que emplean, los entornos para los que están desarrollados y los métodos de configuración, pero todos realizan tareas similares y cumple con el objetivo final, es decir, construir y desplegar proyectos de forma automática. Si que cabe destacar la evolución de los servidores mencionados puesto que, sólo parte de ellos implementan nuevas funcionalidades y mejoras en el servidor con el paso del tiempo, siendo Jenkins uno de los más completos y actualizados.

Emplear un servidor como Jenkins en el que cualquier tipo de configuración se puede realizar desde la interfaz web y a su vez, esta resulta intuitiva y fácil de usar, agiliza considerablemente el proceso de desarrollo software.

Actualmente la gestión de documentos a través de software es un servicio muy demandado por las empresas ya que ofrece un tratamiento de documentos rápido, completo y seguro. Con la evolución de las nuevas tecnologías y el alcance de Internet la digitalización y el mantenimiento de documentos se han vuelto una necesidad que ofrece grandes ventajas.

Bibliografía

Referencias bibliográficas.

Manfred Moser, Tim O'Brien (2011) Oracle, Inc. *The Hudson Book*

Pawan Kumar (2010) Packt Publishing. *Documentum 6.5 Content Management Foundations*

Referencias web.

Gestión documental.

- EMC Corporation (2011). *Documentum Content Server Version 6.7.* Disponible en http://www.jouvinio.net/wiki/images/c/c2/Documentum_Content_Server_6.7_Fundamentals.pdf - [Consultada el 15/02/2016]

- EMC Corporation (2008). *EMC Documentum Composer Version 6.5.* Disponible en https://community.emc.com/servlet/JiveServlet/previewBody/3230-102-2-8563/DFC%20Development%20Guide.pdf - [Consultada el 28/05/2016]

- EMC Corporation (2007). *EMC Documentum Foundation Classes Version 6 Development Guide*. Disponible en http://www.jouvinio.net/wiki/images/c/c2/Documentum_Content_Server_6.7_Fundamentals.pdf – [Consultada el 07/05/2016]

- Parag Doshi's ECM notes (2013). Disponible en http://ecmnotes.com/blog/2013/05/28/documentum-history/ - [Consultada el 05/02/2016]

- Sociedad de la información (2007). *Gestión documental.* Disponible en http://www.sociedadelainformacion.com/12/Gestion%20Documental.pdf – [Consultada el 10/02/2016]

Sistemas de control de versiones (CVS).

- Producir software de código abierto (n.d). Disponible en http://producingoss.com/es/vc.html - [Consultada el 29/02/2016]

- Scott Chacon (n.d). *Pro Git, el libro oficial de Git.* Disponible en http://librosweb.es/libro/pro_git/capitulo_1/acerca_del_control_de_versiones.html - [Consultada el 26/02/2016]

- TortoiseSVN (2015). *Un cliente de Subversion para Windows.* Disponible en https://tortoisesvn.net/docs/nightly/TortoiseSVN_es/index.html - [Consultada el 25/04/2016]

- TortoiseSVN (n.d). *The coolest interface to Subversion control.* Disponible en https://tortoisesvn.net/about.html - [Consultada el 15/02/2016]

Herramientas de construcción.

- Apache (2016). *Apache Ant™ 1.9.7 Manual.* Disponible en http://ant.apache.org/manual/index.html - [Consultada el 05/05/2016].

- Apache (2016). *Apache Maven Project.* Disponible en https://maven.apache.org – [Consultada el 20/04/2016].

- JavaTPoint (n.d). *Maven Tutorial*. Disponible en http://www.javatpoint.com/maven-tutorial - [Consultada el 20/02/2016]

Servidores de integración continua.

- Apache Contiunnum (2015). Disponible en https://continuum.apache.org/ - [Consultada el 20/03/2016]

- Atlassian Documentation (2016). Disponible en https://confluence.atlassian.com/bamboo - [Consultada el 28/03/2016]

- Casaris Asociados (2015).*Gestión de Ambientes con Bamboo.* Disponible en http://casari-asoc.com/blog/gestion-de-ambientes-con-bamboo-conceptos/ - [Consultada el 28/03/2016]

- Cruise Control (n.d). Disponible en http://cruisecontrol.sourceforge.net/ - [Consultada en 01/04/2016]

- Cruisecontrol Configuration (n.d). Disponible en http://cc-config.sourceforge.net/ - [Consultada en 02/04/2016]

- Jenkins (2016). Disponible en https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins – [Consultada el 15/05/2016]

- Novagenia Information Technologies, S.L. (n.d). *Atlassian Bamboo* . Disponible en http://www.novagenia.com/bamboo.php - [Consultada el 28/03/2016]

- SolanoLabs (n.d). Disponible en http://docs.solanolabs.com/ - [Consultada el 31/03/2016]

Calidad de software.

- SonarQube (2016). Disponible en http://www.sonarqube.org/ - [Consultada en 24/04/2016].
- SonarQube (2016). *Documentation.* Disponible en http://docs.sonarqube.org/display/SONAR/Documentation - [Consultada en 01/05/2016].