Gestión de entregas

Dirección General de Sistemas de Información y Comunicaciones

Áreas de Gobierno Tecnológico de SSII y del Dato

Servicio Andaluz de Salud
Consejería de Sanidad, Presidencia y Emergencias

Contenido

Gestión de entregas

Versión: v03r04

Estado: ACTIVO

Entrada en vigor desde: 5 nov 2025

Obligado cumplimiento desde: 5 nov 2025

Enlaces relacionados: Gestión de entregas | Archivos de las aplicaciones. Consideraciones generales.

Cumplimiento normativo

Las normas expuestas son de obligado cumplimiento. La STIC podrá atender casos excepcionales que serán gestionados a través de los responsables del proyecto correspondiente y autorizados por el Área de Gobernanza de la STIC. Asimismo, cualquier aspecto no recogido en estas normas deberá regirse en primera instancia por las guías técnicas correspondientes al esquema nacional de seguridad y esquema nacional de interoperabilidad, según corresponda, y en su defecto a los marcos normativos y de desarrollo software establecidos por la Junta de Andalucía, debiendo ser puesto de manifiesto ante la STIC.

La STIC se reserva el derecho a la modificación de la norma sin previo aviso, tras lo cual, notificará del cambio a los actores implicados para su adopción inmediata según la planificación de cada proyecto.

En el caso de que algún actor considere conveniente y/o necesario el incumplimiento de alguna de las normas y /o recomendaciones, deberá aportar previamente la correspondiente justificación fehaciente documentada de la solución alternativa propuesta, así como toda aquella documentación que le sea requerida por la STIC para proceder a su validación técnica.

Contacto Dpto: **Oficina de Calidad**

Histórico de cambios

Los cambios en la normativa vendrán acompañados de un registro de las modificaciones. De este modo se podrá realizar un seguimiento y consultar su evolución, ordenándose de más recientes a menos recientes.

Versión	Pre-release	Publicación Release	Entrada en Vigor	Retiro	Alcance
v03r04	-	5 nov 2025	5 nov 2025		• El ANEXO I, correspondiente a la e*structura de la carpeta scrips,* se actualiza en el marco de las acciones de mejora del proceso de integración continua.
v03r03	-	19 jul 2024	19 jul 2024	5 nov 2025	• Se incluye en el artículo Archivos de las aplicaciones. Consideraciones generales el apartado con las directrices que se han de aplicar en aplicaciones dockerizadas. • Se mejora el formato del artículo.

v03r02	-	23 abr 2024	23 abr 2024	19 jul 2024	• Se modifica el ANEXO II: *Estructura del repositorio testproject*.
					  ◦ Se elimina configuración de los pom.xml que se han trasladado a la normativa "Automatización de pruebas funcionales"
					  ◦ Se incluyen aclaraciones del repositorio testproject cuando éste incluye sólo pruebas y cuando incluye pruebas y código

v03r01	15 feb 2023	28 abr 2023	28 abr 2023	23 abr 2024	• Se incluye el apartado ***Entregables***.
					• Se incluye el apartado ***Versionado***.
					• Se modifica el apartado ***Estructura del repositorio***.
					  ◦ Se incluye subapartado ***Organización de los proyectos***.
					  ◦ Se incluye organización para el modelo aplicación-componente.
					  ◦ Se actualizan las tablas de los directorios de carpetas, eliminándose las que han quedado obsoletas.
					• Se elimina el apartado ***Características de los ficheros de configuración***. Esta información se traslada al artículo *Archivos de las aplicaciones. Consideraciones generales.*
					• Se elimina el apartado ***Entrega de código fuente*** se sustituye por ***Procedimiento de entrega***. Se suprime cualquier referencia a Faro.
					• Se incluye el apartado ***Servicios de verificación previos a la petición de lanzamiento (PL).***
					• Se incluye el apartado ***Consideraciones sobre la PL.***
					• Anexos:
					  ◦ Se sustituyen *ANEXO I: Catálogo de etiquetas* y *ANEXO II: Estructura de la carpeta scripts* por *ANEXO I: Estructura de la carpeta scripts* y *ANEXO II: Estructura del repositorio testproject*.

v02r01	-	10/03/2021	10/03/2021	27/04 /2023	Gestión de entregas • Estructura del repositorio. En la estructura de proyectos se añade la carpeta *"pruebasdecarga".* • Características de los ficheros de configuración. Se rehace el apartado para explicar mejor los campos a cumplimentar tanto en el pom.xml como en el build.xml. Verificación de calidad del código fuente • Se eliminan las "Notas de interés". • Se actualiza el apartado "El perfil de Sonar". • Se añade información acerca de las características técnicas de la plataforma. • Se actualiza la tabla con los plugins instalados. • Se actualizan los umbrales de calidad. Se elimina el Quality Gate "Warning", ya que no está contemplado en el nuevo SonarQube.

v02r00	-	26/10/2018	01/01/2019	01/03/2021	• Se actualizan todas las referencias al repositorio de código SVN incluyendo GIT como nuevo repositorio. • Dentro del entorno tecnológico se incluye un enlace para la descarga de una instancia de Sonarqube 6.7.2 parametrizada con los datos de la STIC • Características del fichero de configuración: se ha incluido una tabla con los códigos a incluir en la propiedad groupId. Esta tabla contiene una codificación de las diferentes suite de aplicaciones existentes en la CMS • Se ha incluido cual es objeto de las PLs de "Correctivo de datos" y "Carga de datos" y cual es el contenido de los scripts que se adjuntan • Verificación de calidad del código fuente: se indican cuales son la nuevas métricas Sonarqube a revisar, umbrales mínimo y de confianza y los criterios para la revisión de PLs en FARO • Calidad de test unitarios: para aquellos proyectos JAVA se incluye el plugin PITEST como parte de la ejecución en JENKINS con el objeto de medir la calidad de los test unitarios realizados. • Normativa CNO: será de obligatorio cumplimiento todas las reglas descritas en la herramienta CNO • Ficha de Entrega en PLs: Se elimina la obligatoriedad de adjuntar la ficha de entrega en las PLs de FARO. Esta información es mantenida por el proveedor de desarrollo en GIT mediante el fichero readme.md

Introducción

Todos los proyectos software ejecutados para la STIC deben construirse y desarrollarse en base a los requisitos técnicos y tecnológicos que la infraestructura del SAS soporta. En caso contrario, será necesaria la autorización por parte del Área de Gobernanza y Calidad que evaluará la propuesta realizada durante la reunión de lanzamiento del proyecto.

Uno de los objetivos del Área de Gobernanza y Calidad es asegurar que se dispone de productos desplegables en un entorno productivo y que se adecúen a las necesidades de los usuarios.

El presente texto recoge la normativa aplicable para la gestión de entregas del software desarrollado por y para la STIC. Con la aplicación de estas normas y recomendaciones, la STIC pretende:

- Definir los entregables.
- Definir la política de numeración de versiones.
- Implantar y normalizar el procedimiento de entrega de código fuente.

La gestión de las entregas del software desarrollado para la Subdirección de Tecnologías de la Información y Comunicaciones del Servicio Andaluz de Salud se realizará usando las herramientas, los procedimientos y las directrices establecidas en esta norma.

La STIC se está evolucionando a un nuevo modelo de gestión del ciclo de vida del software basado en la dupla APLICACION - COMPONENTES, con el objeto de simplificar y mejorar la gestión de las aplicaciones. Todo lo relacionado con un producto se va a gestionar en un único proyecto en JIRA. Cada una de las particiones independientes de una aplicación se denominarán componentes, que podrán ser del tipo módulo o librería. Actualmente este modelo se está pilotando en varios proyectos. En las distintas secciones de esta norma que lo requieran, se puntualizará la información a tener en cuenta en aquellos proyectos que vayan adoptando este modelo de gestión.

Entregables

Se parte de la premisa de que el código fuente de una aplicación, sus scripts de base de datos, y su correspondiente código compilado deben quedar almacenados en la infraestructura de la STIC (herramientas ALM), independientemente de la tecnología utilizada. Para materializar este objetivo, la suite de herramientas que cubren el ciclo de integración y entrega continua son:

- Sistema de control de versiones: GitLab (http://git.sas.junta-andalucia.es/).
- Servidor para la integración continua: Jenkins (http://ci.alm.sas.junta-andalucia.es/).
- Artefactos compilados, archivos de configuración y cualquier fichero necesario para el despliegue: DML (Nexus) (https://dml-si.sas.junta-andalucia.es/).

Para que el software objeto de la entrega sea correctamente gestionado, es necesario solicitar vía CGES tanto la creación del repositorio de código de la aplicación, como el job de Jenkins que realizará las operaciones que sean necesarias en función de la naturaleza del proyecto. Estas peticiones deben realizarse una vez registrada la aplicación y /o componente en CMS, lo más pronto posible (al inicio del proyecto), y siempre antes de la primera entrega. Si necesita conocer cómo darlas de alta, puede consultar el artículo Catálogo peticiones a la Oficina de Calidad vía CGES.

En la solicitud de alta de un nuevo componente en CMS, es necesario tener en cuenta que la nomenclatura del atributo "Código ALM", siempre debe basarse en la convención de nombres de la tecnología del proyecto y coincidirá con el nombre del repositorio en Git.

- Java, Node: Todo minúsculas y guiones medios
- .Net, Drupal, Delphi: CamelCase sin guiones

Para más información con respecto a CMS y la plantilla para el registro de una aplicación en el mismo, puede consultar el artículo Servicios de Gestión interna de la DGTIC#Plantillas

Otro aspecto a tener en cuenta es que, para la configuración del job de Jenkins, es necesario disponer del archivo **read me.md** correctamente cumplimentado. La plantilla a utilizar está disponible en el repositorio http://git.sas.junta-andalucia.es/gobernanza/ArchivosDeDesarrollo/recursos_sas. **Es obligación del proveedor de desarrollo mantener constantemente actualizada la información incluida dentro de este fichero**.

Al inicio de un proyecto **NUEVO es *necesario realizar una entrega "ficticia" con el pom.xml y el archivo readme.md y etiquetarla como 0.0.0.1*** debido a que Sonarqube sólo analiza el incremento de código incluido desde la última entrega de la versión anterior. Así, de cara a la primera entrega para el despliegue de la aplicación, Sonarqube tendrá con qué comparar y los resultados del análisis aparecerán de forma correcta.

Modelo aplicación-componente

Los proyectos que comiencen a adoptar el nuevo modelo de gestión aplicación-componentes, deberán contactar con el jefe del proyecto del área de Coordinación y Estrategia, Olimpia Torres Barranco <olimpia.torres@steria.com> , quien orquestará el inicio de todo el proceso y la gestión de las peticiones.

Versionado

El versionado de las entregas, independientemente de la tecnología de desarrollo, deberá ser identificado con 4 dígitos, según la notación X.Y.Z.T (**mayor.menor.revisión.entrega)** donde *X, Y, Z y T* son números enteros no negativos, ***y NO DEBEN ser precedidos de ceros****.*

- **mayor (X):** indica la versión principal del software, consistiendo en un conjunto de funcionalidades concretas que son recogidas y cubiertas en dicha versión. Debe modificarse cuando la versión que se entregue presente nuevas funcionalidades claves de la aplicación respecto de la versión anterior. La modificación de esté dígito se realiza por la inclusión de nuevos requerimientos al sistema, tal y como la inclusión de nuevos módulos o una revisión completa de los existentes.
- **menor (Y):** indica la funcionalidad menor cubierta en la versión de software entregada. Debe ser modificado cuando hay cambios significativos en la forma en la que se ofrece la funcionalidad existente, cuando se corrigen grandes fallos del sistema, o cuando hay nuevas versiones evolutivas que modifican significativamente la funcionalidad ofrecida.
- **revisión (Z):** se modifica cuando hay revisiones de código ante fallos de la aplicación.
- **entrega (T):** este dígito indica el número de veces que un paquete ha sido entregado. La re entrega del producto podrá producirse para subsanar un defecto en el software detectado durante las etapas de revisión previas a su instalación, durante la propia instalación, o bien durante el periodo de pruebas anterior a su puesta en producción.

La primera versión estable de cualquier producto NUEVO debe comenzar en la 1.0.0.1. En el caso de que el producto ya exista y tenga una versión diferente, se entenderá que la primera versión dada de alta en el repositorio de código será congruente con la existente. Además:

- El campo versión en Jira debe informarse 3 dígitos según la notación **mayor.menor.revisión**.
- El campo entrega en Jira debe añadir un cuarto dígito a la versión identificada de la que depende, según la notación **mayor.menor.revisión.entrega** y, siempre, en consonancia con el versionado de las entregas de software realizadas en el repositorio de código de corporativo de la STIC.

Modelo aplicación-componente

Los proyectos que adopten el modelo de gestión aplicación-componentes, deberán aplicar las reglas anteriores al versionado de cada componente individual.

Para el versionado de la aplicación completa, se establece como buena práctica la adopción de los mismos criterios, que estará condicionada por la versión de sus componentes.

Criterios de versionado para la aplicación:

- Si algún **componente** se entrega con un **cambio en el primer dígito** (por cambio tecnológico o funcional de gran envergadura), **la versión de la aplicación también deberá reflejar ese cambio en el primer dígito.**
- Si los componentes entregan cambios en los **dígitos segundo (Y)** y **tercero (Z)**, la versión de la aplicación deberá **incrementar el segundo dígito**.
- Si los cambios en componentes afectan **solo al segundo dígito (Y)**, la aplicación también incrementará ese dígito.
- Si solo se realizan entregas de **correctivos (tercer dígito, Z)**, la aplicación deberá reflejar ese cambio en su propio tercer dígito.
- Si únicamente se realizan **reentregas (cuarto dígito, T)** de los componentes, la aplicación **incrementar á también su cuarto dígito en una unidad.** Este valor indica el número de veces que se ha reentregado la aplicación dentro de la misma versión. **Cabe señalar que el valor del cuarto dígito de la aplicación no tiene por qué coincidir con el mayor valor del cuarto dígito entre los componentes individuales.**

Estructura del repositorio

Según la tecnología de desarrollo utilizada será necesario incluir carpetas específicas en el directorio que se definirán al inicio del proyecto por el Área de Gobernanza y Calidad. La estructura acordada se ha de documentar en el espacio Confluence propio del proyecto.

La normativa aplicable para la operativa con el repositorio GIT puede encontrarse en el siguiente enlace.

Los aspectos comunes a las diferentes tecnologías, relacionados con la configuración de las aplicaciones, están descritos en el artículo *"Archivos de las aplicaciones. Consideraciones generales"*.

Organización de los proyectos

La organización del proyecto en Git seguirá las siguientes directrices:

- Debe existir un repositorio para el código. Se ha de solicitar el alta con el nombre del código ALM de la aplicación en CMS.
- Debe existir un repositorio para las pruebas. Almacenará todos los ficheros .features de las aplicaciones que utilizan metodología agile y/o automatización de pruebas funcionales. Se solicitará el alta con el nombre: <código ALM de la aplicación en CMS>-testproject. Puede consultar todos los detalles de la estructura interna en el *ANEXO II: Estructura del repositorio testproject*.
- Ambos estarán englobados en un grupo. Se solicitará su alta con el nombre del código ALM de la aplicación en CMS.

Estructura del repositorio de código para proyectos JAVA

El objeto del repositorio de proyectos para el control del versiones es el **almacenamiento de manera exclusiva del código fuente y los procedimientos que permiten obtener el código binario asociado** a ese código fuente del aplicativo. Quedan excluidos del repositorio los ficheros relacionados con la gestión o documentación del proyecto (ficheros del tipo .eap .doc, etc). **El incumplimiento de esta directriz conllevará el rechazo de la PL asociada a la entrega de código fuente** .

Partiendo de la directriz marcada en el párrafo anterior y de la estructura que proporciona la tecnología MAVEN para los proyectos JAVA, todos los repositorios incluidos en la herramienta de control de versiones (GIT) tienen que estar organizados según la estructura común que a continuación se indica:

Directorio	Contenido
pom.xml (para proyectos basados en Maven) o build.xml (para proyectos basados en Ant)	En la carpeta raíz es donde se almacenará el fichero.
scripts	Contiene todos los ficheros necesarios para construir el modelo de datos. La estructura interna de esta carpeta se define en el siguiente enlace.

pruebasdecarga	Contiene todos los ficheros relacionados con las pruebas de carga del sistema. La estructura interna de esta carpeta se define en el artículo Pruebas de carga del sistema.

Estructura del repositorio de código para otras tecnologías NO JAVA

Desarrollo PHP

Para los desarrollos con tecnología PHP, el código fuente deberá estar estructurado según las pautas comunes:

Estructura del repositorio para tecnología PHP

Directorio	Contenido
scripts	Contiene todos los ficheros necesarios para construir el modelo de datos. La estructura interna de esta carpeta se define en el siguiente enlace.
pruebasdecarga	Contiene todos los ficheros relacionados con las pruebas de carga del sistema. La estructura interna de esta carpeta se define en el artículo Pruebas de carga del sistema.

Desarrollos VB6 y .NET

Para los desarrollos .NET, el código fuente estará estructurado según las pautas definidas en el documento de definición del Framework Base .NET desarrollado en base a la Arquitectura de Referencia .NET y referenciado en el documento de Normas de Desarrollo .NET publicado en este espacio.

Los scripts de BBDD serán incluidos en un directorio (cuyo nombre debe ser scripts) identificado a tal efecto dentro del mismo paquete de software.

Estructura del repositorio de código para tecnología VB6 y .NET

Directorio	Contenido
scripts	Contiene todos los ficheros necesarios para construir el modelo de datos. La estructura interna de esta carpeta se define en el siguiente enlace.
pruebasdecarga	Contiene todos los ficheros relacionados con las pruebas de carga del sistema. La estructura interna de esta carpeta se define en el artículo Pruebas de carga del sistema.

Desarrollos ENSEMBLE

Para los desarrollos realizados con tecnología ENSEMBLE e incorporados al repositorio de código no existe una estructura de directorios específica. Estos desarrollos son incorporados al repositorio como un fichero con formato XML plano.

Otros desarrollos: Cobol, Delphi, PowerBuilder, business intelligence

El resto de proyectos existentes en la STIC, con tecnología Cobol, Delphi y PowerBuilder, o aquellos proyectos enfocados dentro del área de business intelligence, mantendrán la estructura de directorios existente actualmente debiéndose incluir, los scripts de BBDD en un directorio (cuyo nombre debe ser scripts) identificado a tal efecto dentro del mismo paquete de software.

Estructura del repositorio para tecnología Cobol, Delphi y PowerBuilder

Directorio	Contenido
scripts	Contiene todos los ficheros necesarios para construir el modelo de datos. La estructura interna de esta carpeta se define en el siguiente enlace.

Modelo aplicación-componente

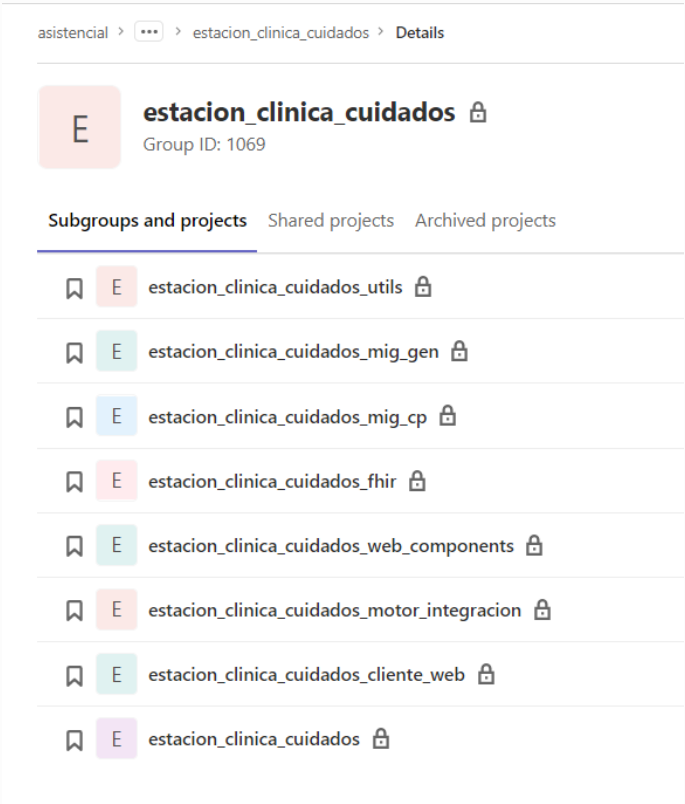
En el modelo de gestión aplicación-componentes se deberá tener en cuenta que la organización en GitLab pretende replicar la de JIRA. Habrá un grupo equivalente a la aplicación JIRA en el que estarán incluidos los repositorios correspondientes a los distintos componentes.

Así, por ejemplo, existirá una aplicación en JIRA llamado *ESTACION CLINICA DE CUIDADOS VERSION WEB*, en el que se gestionará todo lo relacionado con esta aplicación y con los componentes que la forman:

- estacion_clinica_cuidados_cliente_web
- estacion_clinica_cuidados_fhir
- estacion_clinica_cuidados_mig_cp
- estacion_clinica_cuidados_mig_gen
- estacion_clinica_cuidados_motor_integracion
- estacion_clinica_cuidados_utils
- estacion_clinica_cuidados_web_components
- estacion_clinica_cuidados-testproject

En GitLab existirá un grupo llamado estacion_clinica_cuidados (su nombre corresponderá con el código ALM de la aplicación en CMS), que englobará a todos los componentes de la aplicación. Cada componente tendrá su propio repositorio.

http://git.sas.junta-andalucia.es/asistencial/diraya/estacion_clinica_cuidados



En CMS, la aplicación aparece con su número de entrega y agrupando a sus componentes que también tienen informada su correspondiente entrega.

Un aspecto importante a tener en cuenta es ***la entrega de las pruebas***.

El código de las pruebas automatizadas y sus correspondientes archivos .features se entregarán dentro de un componente denominado <CALM>-testproject, donde CALM es el código ALM de la aplicación. El testproject se tratará como un componente más, y tendrá su propio repositorio, aunque su estructura interna será un poco distinta. Puede consultar todos los detalles en el *ANEXO II: Estructura del repositorio testproject.*

Procedimiento de entrega

Para la entrega de código fuente en GIT, se almacenará en la rama principal (master) el código de la aplicación y cada entrega de código supondrá la creación de una nueva versión (tag) según lo establecido en la normativa GIT .

La entrega de código fuente se realizará accediendo a través de VPN al repositorio GIT correspondiente publicado en la dirección http://git.sas.junta-andalucia.es/ siguiendo el procedimiento:

1. Preparar el paquete para la entrega, incluyendo código fuente y scripts de base de datos según la estructura de directorios definida en el apartado Anexo I: Estructura de la carpeta scripts.
2. Incorporar el código fuente (que incluye los scripts de BBDD) a GIT según lo establecido en la normativa GIT. Como parte de la entrega de código fuente deberá existir en el directorio raíz un fichero **readme.md** cuya información se ha de mantener actualizada.
3. Etiquetar la entrega según la normativa definida para el versionado. **Una vez generada una versión de un producto, almacenada en el repositorio correspondiente y etiquetado, ésta no podrá ser modificada bajo ningún concepto**.

4. Desde la STIC se está impulsando la NO aportación de software empaquetado y/o scripts de BBDD dentro de las PLs independientemente del tipo de tecnología usada, habilitándose la **Digital Media Library** (DML).

Actualmente existe un número bastante grande de aplicaciones gestionadas por la STIC. La tecnología en la que están desarrolladas condiciona su forma de entrega.

> Al inicio del proyecto se definirá el escenario que aplica sobre el proyecto teniendo en cuenta la tecnología y características del mismo con el área de Gobernanza de la STIC.

Nos podemos encontrar con diversos escenarios:

- Aplicaciones compiladas por las herramientas CI/CD de la STIC.

  Por ejemplo, aplicaciones desarrolladas con Java:

  En este caso, una vez subido el código y los scripts de base de datos a git, Jenkins se encargará de orquestar las tareas propias de CI/CD (compilación, test, deploy, code quality). Se creará un paquete (formato .zip) con el binario y los scripts de base de datos. El conjunto se subirá de forma automática a la DML, donde estará disponible para que CTI lo despliegue.

- Aplicaciones empaquetadas por las herramientas CI/CD de la STIC.

  Por ejemplo aplicaciones desarrolladas con otras tecnología: desarrollo PHP, VB6 , ENSEMBLE, Cobol, Delphi.

  En este caso, una vez subido el código y los scripts de base de datos a git, Jenkins se encargará de empaquetarlos de forma conjunta (formato .zip) y lo subirá a la DML donde quedará disponible para que CTI lo despliegue.

- Aplicaciones ***NO*** compiladas ni empaquetadas por las herramientas CI/CD de la STIC.

  Por ejemplo, aplicaciones desarrolladas con PowerBuilder, MicroStrategy. En este caso, se realizarán una o dos acciones:

  Si existe código fuente, se debe versionar en git, según la normativa *"Flujos de trabajo en los repositorios STIC"*.

  Es necesaria la creación de un tique JIRA del tipo ***"Entrega de binarios"*** en el que se adjuntará el código compilado y los scripts de base de datos (en formato zip), necesarios para el despliegue.

5. Crear el registro del servicio de verificación en Jira que puede ser de dos tipos:

Servicio de Verificación de Entrega (SVE), si se trata de un evolutivo o correctivo de aplicación/componente donde la información de la versión y entrega debe estar alineada con la misma en Git.

Servicio de Verificación de Datos (SVD), si se trata de un correctivo o carga de datos. Se ha de tener en cuenta la siguiente información:

- Para peticiones de " ***Carga de Datos*** ", no se permitirá el uso de cualquier sentencia que pueda modificar el esquema de BBDD. Solo se permitirán sentencias SELECT, INSERT, UPDATE o DELETE.
- Para peticiones de " ***Correctivo de Datos*** " se podrá modificar el esquema de BBDD durante la ejecución de los scripts con todo tipo de sentencias (siempre teniendo en cuenta la normativa de BBDD aplicable), **si bien al finalizar la ejecución de la PL el esquema de BBDD deberá permanecer exactamente igual que antes de la ejecución de la PL.** Es decir, si suponemos que se incluye una sentencia CREATE TABLE en los scripts (porque sea necesario crear una tabla de apoyo), dentro de los mismos scripts debe venir la correspondiente sentencia DROP TABLE. Y así con todos los elementos del esquema de BBDD que se creen o modifiquen.

El Servicio de Verificación se ha de asignar a la Oficina de Calidad una vez realizada la entrega con el objetivo de garantizar el cumplimiento normativo de la STIC, así como la calidad del código del producto y una vez resuelta se podrá proceder con el siguiente paso.

El proceso de subida automática a DML aún no está implementado para PLs de datos. Mientras no se encuentre disponible dicha adaptación de las herramientas para las PLs de tipo "Carga de Datos de Aplicativo" o "Correctivo de Datos de Aplicativo" independientemente de la tecnología utilizada, se admite la subida de script de base de datos en la SVD (obligatoria de forma previa a la petición de lanzamiento).

6. Crear Petición de Lanzamiento (PL) en Jira. Solicitud de servicio de transición de instalaciones software que llegará al adjudicatario para el despliegue de la entrega.

- Peticiones de lanzamiento de incremento de versión (PL versión) para el despliegue de evolutivo o correctivo de aplicación.

  Ha de cumplir las pautas definidas para el versionado y alineada con la entrega.

  Requiere disponer previamente de una SVE resuelta y que no haya sido rechazada por error en la compilación.

  Sólo podrá crearse una PL por entrega a menos que la primera PL se cancele. En ese caso, se podrá crear otra PL usando la misma SVE.

- Peticiones de lanzamiento para modificaciones de datos (PL datos) cuando se solicita carga o corrección de datos que requieren implantación.

  Sólo se podrá avanzar la PL cuando la resolución de la SVD es Favorable.

  La PL se CANCELA de forma automática si la SVD relacionada con la misma ha sido rechazada por modificación de la línea base.

  La PL se mantiene bloqueada si la SVD relacionada con la misma ha sido rechazada por incumplimiento CNO. En este caso se permite generar una nueva SVD con las correcciones y una vez resuelta favorablemente se podría avanzar con la PL.

- Peticiones de lanzamiento para cambios de configuración (PL configuración). Aplican sobre todas las aplicaciones centralizadas sobre las que se requieren peticiones de configuración.
- Peticiones de lanzamiento para modificación/configuración de software base (PL sistemas). Aplican sobre todas las plataformas centralizadas sobre las que se quieren hacer cambios menores, es decir, sin que sea necesaria una petición de plataforma.

Para el caso de las **PLUs** de incremento de versión y de datos, ***NO será necesario que previamente haya una SVE, ni una SVD resuelta por parte de la OCa***. Los servicios de verificación correspondientes se generarán de forma automática y la Oficina de Calidad informará del resultado en los mismos aún estando la entrega ya implementada.

Para más información puede consultar el artículo Gestión del Lanzamiento.

7. Cumplimentar el campo observaciones de la PL correspondiente con los comentarios relativos a las excepciones o peculiaridades a tener en cuenta durante el proceso de instalación o desinstalación de dicha PL, como pueden ser las peticiones CGES no resueltas y con las que exista alguna dependencia.

Servicios de verificación previos a la petición de lanzamiento (PL)

Servicio de Verificación de la Entrega (SVE)

La Oficina de Calidad realiza una revisión de diversos aspectos relacionados con la entrega de la versión.

De forma general, las comprobaciones que se realizan son las siguientes:

- Verificación del repositorio de código.
  - Uso correcto de Gitflow.
  - Etiquetado de la versión.
  - Organización del proyecto en el repositorio.
- Verificación y validación de la calidad estática de código.
  - Revisión tarea Jenkins.
  - Sonar: Métricas, umbrales, excepciones.

    Artículo de referencia: "*Verificación de la calidad del código fuente*".

    Con respecto a Sonarqube, hay que tener en cuenta una cuestión importante. Cuando se realiza una entrega, Sonarqube sólo analiza el incremento de código incluido desde la última entrega de la versión anterior. No se analiza el código completo de la aplicación, sólo el correspondiente a la versión que se pretende implantar.

    Para que la primera comparativa se realice correctamente, es necesario realizar una entrega "ficticia" con el pom.xml y el archivo readme.md y etiquetarla como 0.0.0.1. Así, Sonarqube tendrá con qué comparar y los resultados del análisis aparecerán de forma correcta.
  - Revisión script: CNO.

    Artículo de referencia: *"Normativa BBDD Oracle"*.
- Comprobación del correcto empaquetado y ubicación en la DML.

  En el momento de ejecutar la PL, CTI cogerá los binarios de Nexus. Por lo tanto, es imprescindible comprobar que se han subido correctamente. La ruta relativa de la ubicación de los binarios en la DML será la misma que su análoga para el código fuente en Git. En el caso de que no haya entrega de código fuente, la ruta será la indicada en el registro de entrega de binarios.

  Por ejemplo:

  Componente: estacion_clinica_cuidados_cliente_web

Entrega: 3.6.6.2

Url de Git: http://git.sas.junta-andalucia.es/asistencial/diraya/estacion_clinica_cuidados/estacion_clinica_cuidados_cliente_web/-/tree/3.6.6.2

Url de Nexus: https://dml-si.sas.junta-andalucia.es/repository/static-binaries/asistencial/diraya/estacion_clinica_cuidados/estacion_clinica_cuidados_cliente_web/estacion_clinica_cuidados_cliente_web.3.6.6.2.zip

Por lo tanto en el PID correspondiente, habría que indicar que el binario se encuentra en Nexus, en la ubicación:

Ruta del componente: /asistencial/diraya/estacion_clinica_cuidados/estacion_clinica_cuidados_cliente_web/

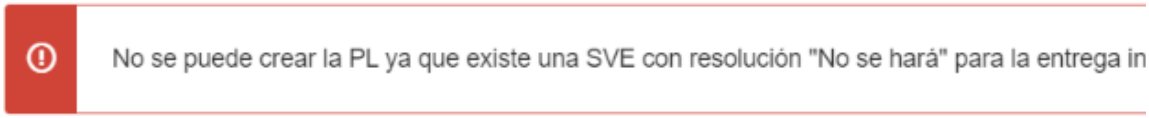
Entrega: estacion_clinica_cuidados_cliente_web.3.6.6.2.zip

Motivos de rechazo

A continuación se muestra el listado de los motivos por los que una SVE se puede rechazar. De todos ellos, sólo la compilación fallida en Jenkins será bloqueante para la creación de la PL. El resto no la impedirá, quedando la decisión de seguir adelante, bajo responsabilidad del jefe de proyecto.

- Estructura repositorio incorrecta (faltan carpetas/readme, etc ...)
- Plantilla Readme.md mal informada
- Incumplimiento CNO
- GitFlow incorrecto (commits hechos en master o develop, merges mal hechos, etc ...)
- Estructura y/o contenido incorrecto del Pom/Build
- No cumple los umbrales de Sonarqube.
- Compilación fallida en Jenkins

Este motivo de rechazo impedirá la creación de la PL en JIRA.



Modelo aplicación-componente

¿Cómo influye este nuevo modelo en el procedimiento de entrega?

En cada entrega, tendremos ***una SVE de aplicación, vinculada a n SVE´s de componentes***. Se procederá de la siguiente manera:

- La SVE de aplicación se creará de forma manual. Irá versionada y con la entrega también informada.
- Se vinculará con cada una de las SVE de componente de todos los módulos o librerías incluidos en la entrega de la aplicación.
- No será necesaria la creación de las SVE´s de componentes. Será Jenkins el encargado de hacerlo de forma automática tras la ejecución exitosa del job.

Motivos de rechazo

Los motivos de rechazo de una SVE de aplicación son los mismos que los de cualquier otra SVE. Si cualquiera de las SVE´s de componente es rechazada, su correspondiente SVE de aplicación también lo será y en el caso que se rechacen varios componentes de la misma, se seleccionará como “Motivo” del rechazo el más restrictivo.

Servicio de Verificación de Datos (SVD)

Como comentábamos en el apartado *Procedimiento de entrega*, existe un Servicio de de verificación (SVD) asociado a las PL de datos.

Tanto en el caso de carga como en el de correctivo de datos, se revisará que los scripts que se entregan cumple la ***Normativa BBDD Oracle*** publicada en el área de Gobernanza y Calidad.

Motivos de rechazo

Básicamente son dos:

- Incumplimiento de CNO.

  Los scripts incumplen la normativa.

- Modificación de la línea base del aplicativo.

  Se utiliza una PL de datos de forma incorrecta ya que se modifica, de alguna manera , el esquema de base de datos.

Modelo aplicación-componente

¿Cómo influye este nuevo modelo en el procedimiento de entrega?

En cada entrega, se generará ***la SVD a nivel de aplicación.***

Consideraciones sobre la PL

A la hora de dar de alta una PL, se debe tener en cuenta que:

- Es necesario tener una SVE que poder asociar a la PL. En el caso de una PLU no es necesario que esté resuelta.
- Ya no se adjunta el PID. Ahora es un documento Confluence asociado a la PL. De hecho, ***no se adjunta ningún archivos a la PL***.
- Como hemos comentado varias veces a lo largo de este artículo, CTI sólo va a tener acceso a la DML. Cualquier paquete necesario para un despliegue debe estar ubicado en Nexus.

Anexos

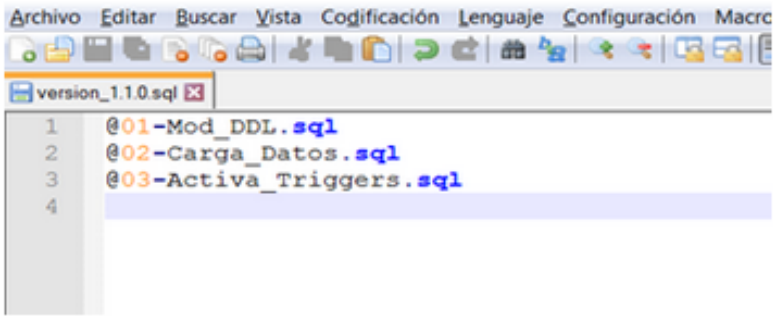
ANEXO I: Estructura de la carpeta scripts

La carpeta scripts dentro del repositorio del código fuente contendrá una carpeta por cada versión del producto realizada siguiendo la nomenclatura de 3 dígitos (en este caso, se obvian las diferentes reentregas de una misma versión).

Cada una de estas carpetas contendrá la siguiente información:

- Todos los scripts de base de datos necesarios para realizar la actualización del producto.
- Los scripts con el juego de datos necesario para poder actualizar la base de datos disponiendo así de los datos necesaros para ejecutar las pruebas correspondientes a la versión entregada.

Salvo que la actualización del producto necesite instrucciones especiales, y de cara a una próxima automatización de tareas, se estandarizará la nomenclatura de los ficheros existentes dentro de estas carpetas, debiendo existir siempre dos ficheros llamados version_X.X.X.sql y undo_version_X.X.X.sql el cual se encargará de realizar la actualización completa del producto (Y la marcha atrás en el caso del undo) ya sea porque contenga todos los scripts necesarios o bien porque se encargue de invocar a otros sql dentro de la misma carpeta.



Ejemplo de fichero version_X.X.X.sql

De esta forma, la estructura de carpetas y scripts quedará de la siguiente forma:

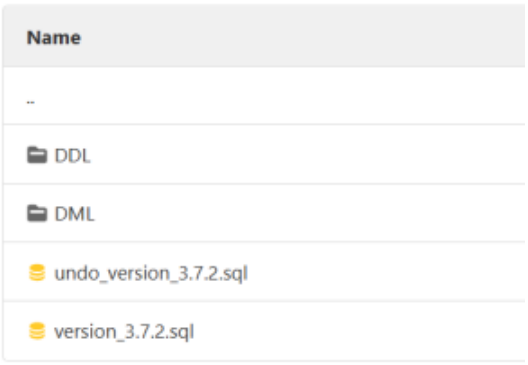
- Una carpeta DDL (Lenguaje de Definición de Datos).

Scripts con sentencias CREATE, ALTER, DROP.

- Una carpeta DML (Lenguaje de manipulación de datos)

Scripts con sentencias SELECT, INSERT, UPDATE, DELETE.

- version_X.Y.Z.sql. Script que ejecuta la actualización completa del producto.
- undo_version_X.Y.Z.sql. Script que ejecuta la marcha atrás.



Con carácter obligatorio

Es muy importante que los scripts queden agrupados en sus respectivas carpetas (DDL - DML). ***Es esencial prestar especial cuidado a la idempotencia de los scripts*** ya que es la única manera de evitar errores durante su ejecución automática.

ANEXO II: Estructura del repositorio testproject

T*estproject* es un repositorio en el que se almacenan las pruebas y, caso de que lo hubiera, el código correspondiente a su automatización.

El proyecto sólo contiene pruebas

Algunas consideraciones:

- Si el proyecto almacena sólo el enunciado de las pruebas, es decir, si no se sube código, no es necesario que versione.
- Las pruebas se enlazarán, como es habitual, con su HU correspondiente.
- Se cumplirán las buenas prácticas en el manejo de Git (no commits en develop, creación de feature para las subidas, ...).

La organización del repositorio se consesuará con la STIC. Por efecto, se escogerá entre:

- Si existen subsistemas funcionales en el proyecto:
  - *Modelo aplicación-componente: root/ código ALM de la aplicación-testproject/features/$Subsistemafuncional$/*

    - *Modelo tradicional: root/código ALM de la aplicación-testproject/features/$Subsistemafuncional$/*
- Si NO existen subsistemas funcionales definidos en el proyecto:
    - *root/código ALM de la aplicación-testproject/features/*

El proyecto contiene pruebas y código

Si el repositoriro *Testproject* almacena código de automatización de pruebas:

- Actuará como un componente más del proyecto.
- Versionará de forma independiente, como cualquier otro repositorio.
- La gestión del código cumplirá con la normativa establecida.

Toda la información con los detalles sobre la estructura interna del repositorio *Testproject* se encuentra en : *Automatización de pruebas funcionales*

La carpeta *"feature"*

Aunque todos los detalles relacionados con la estructura del Testproject así como las principales directrices de codificación se encuentran en el artículo *Automatización de pruebas funcionales,* hay que hacer mención especial a la carpeta en la que se entregan los archivos .feature.

Desde el punto de vista de la organización, es importante señalar que las pruebas se agruparán, siempre que sea posible, en áreas funcionales que llamaremos Subsistemas. Cada uno de ellos tendrá asociado un código.

- El código del subsistema será numérico.
- La relación entre el subsistema y su código quedará reflejada en una tabla que se publicará en el espacio de Confluence del proyecto correspondiente.

ID	SUBSISTEMA
001	$Subsistemafuncional$
002	$Subsistemafuncional2$
003	$Subsistemafuncional3$
004	$Subsistemafuncional4$

Modelo aplicación-componente

En el caso de que se adopte un modelo de aplicación-componente, la tabla tiene la siguiente estructura:

ID	SUBSISTEMA	COMPONENTE (código ALM)
001	$Subsistemafuncional1$	Módulo 1
002	$Subsistemafuncional2$	
003	$Subsistemafuncional3$	Módulo 2
004	$Subsistemafuncional4$	