Versión 1.0

PSG2

PROCESO SOFTWARE Y GESTIÓN II

Informe técnico de la gestión de la configuración

Tutor: José Antonio Parejo Maestre

Carmen Mª Muñoz Pérez (Scrum Master)

Enrique Salazar Márquez

Javier Martínez Fernández

José Carlos Morales Borreguero

Rafael Ángel Jiménez Fernández

Contenido

[Estructura de los repositorios y ramas por defectos 2](#_Toc67167954)

[Estrategias de ramas basadas en Git flow y revisión por parejas 2](#_Toc67167955)

[Desarrollo de las ramas features 2](#_Toc67167956)

[Preparación de la release 3](#_Toc67167957)

[Cómo solucionar errores en Produción 3](#_Toc67167958)

[política de versiones 3](#_Toc67167959)

### Estructura de los repositorios y ramas por defectos

La estrategia de ramas seguida es Git Flow, que constará de dos ramas por defecto: *“master”* y *”develop”.*

La rama “master” solo será actualizada cuando se tenga una versión lista para producción tras un Sprint. Las ramas que podrán actualizar *“master”* serán únicamente *“release”* y *“hotfix”* (explicadas a continuación).

La rama *“develop”* será actualizada desde la familia de ramas *“feature”* una vez se tenga una funcionalidad totalmente implementada, siendo esta actualización un nuevo commit. Además, al producirse una modificación en las ramas *“release”* o *“hotfix”*; *“develop”* será actualizada obligatoriamente por el Scrum Master. Cabe destacar, que en caso de encontrar algún fallo o modificación en esta rama, el cambio será realizado desde la rama *“feature”* correspondiente a esa funcionalidad, recalcando que ese cambio no se realiza directamente en la rama *“develop”.*

En cuanto a los documentos, se seguirá una estrategia determinada. La ruta en la cual se añadirán todos los documentos desarrollados será “src/main/resources/doc”. Aquellos documentos que sean actas de reuniones (Daily Scrum, Sprint Planning …), serán añadidos directamente a la rama *“develop”.* Por otro lado, aquellos que sean informes técnicos serán realizados en una rama de tipo *“feature”* correspondiente y se procederá como se explicará más adelante.

Finalmente, se utilizarán ostras ramas para completar el flujo de Git Flow: una rama *“release”,* una rama *“hotfix”* y varias ramas *“feature”* (se explicarán en la siguiente sección en detalle).

Una estandarización de nuestro proyecto será que cada vez que se realiza un *“merge”* entre ramas, este deberá ser forzado a no realizar fast-forward. El parámetro por añadir será: “--no-ff ”.

### Estrategias de ramas basadas en Git flow y revisión por parejas

### Desarrollo de las ramas features

Se ha acordado la creación de una rama “feature” por cada tarea/funcionalidad que se implementará en el proyecto, siendo la persona encargada de esa funcionalidad la que la cree. La nomenclatura de esta rama tendrá la siguiente estructura siempre: “feature/X-{descripción breve}”, donde X es el código que identifica a la tarea creada en GitHub.

Para que una feature se considere terminada deberá ser revisada por al menos una persona diferente a la asignada a la tarea, teniendo en cuenta que la decisión del número de revisores de esta tarea se llevará a cabo en el Sprint Planning (sujeto a cambios por complejidad de la tarea). Así, cuando una tarea se encuentre en revisión, la primera persona que esté libre será la encargada de realizar esa revisión, indicando su nombre en la descripción de la tarea.

Una vez que la tarea esté revisada y cerrada, esta se considerará terminada y se podrá realizar una Pull Request a la rama *“develop”.* Este lo realizará la persona que tenía asignada esa tarea.

Finalmente hay una excepción con lo explicado anteriormente. Si la rama *“feature”* corresponde a una tarea de documentación se pueden dar tres escenarios. Si la tarea está asignada a todos los miembros del grupo, la persona que revisará esa tarea será el Scrum Master, y además será el encargado de hacer el merge con *“develop”.* Si la tarea está asignada a un miembro del grupo diferente al Scrum Master, sucederá lo mismo que el caso anterior. Finalmente, si la tarea está asignada al Scrum Master, otro miembro del equipo tendrá que revisar el documento. Como se puede ver, en cualquier caso, una tarea de tipo documento siempre será cerrada por el Scrum Master.

### Preparación de la release

Siguiendo la estrategia de ramas de Git Flow, se creará una única rama en el repositorio llamada *“release”.*

No se creará la rama *“release”* hasta que estén las suficientes funcionalidades que se estimen para obtener una probable primera versión del producto.

Durante todo el ciclo de vida del desarrollo del producto solo existirá una única rama de este tipo. Cada nueva release se creará a raíz de la rama *“develop”*. Una vez se consideré que en la rama *“release”* está el producto listo para producción, está rama actualizará *“master”.* Además, en release se podrán solucionar pequeños errores y actualizar metadatos.

Normalmente, se realizará una release por cada Sprint; ya que en cada uno se pretende realizar las funcionalidades suficientes como para tener un producto potencialmente bueno para producción. En caso de que no se consigan los objetivos del Sprint, es posible que esta no se realice. Hay que añadir, que cada release tendrá una tag asociada indicando el nombre de esta.

Finalmente, es importante recalcar que solo el Scrum Master será el encargado de hacer el merge de esta rama con otras.

### Cómo solucionar errores en Produción

Siguiendo la estrategia de ramas de Git Flow, se creará una única rama en el repositorio llamada *“hotfix”.* Esta se dedicará exclusivamente a corregir aquellos errores que se produzcan una vez el sistema está en producción, es decir, una vez la rama *“master”* está actualizada y se produce un fallo en el producto.

El error en cuestión deberá ser corregido por aquella persona del equipo de desarrollo cuya funcionalidad ha fallado. Una vez esté solucionado, el Scrum Master será el encargado de hacer el merge con las ramas *“master”* y *“develop”.* Es importante que las ramas anteriormente mencionadas se actualicen, ya que si no podría haber conflictos.

### política de versiones

Para esta política se seguirá la siguiente semántica de versionado: X.Y.Z, siendo X, Y, Z números enteros positivos. A continuación, se irán especificando cuál es el criterio para modificar esos números.

La X representa cambios mayores, es decir, este número se incrementará en una unidad cada vez que haya un cambio muy significativo en la aplicación. Estos cambios serían cambio de funcionamiento de alguna funcionalidad, eliminar alguna funcionalidad o añadir un número aceptable de funcionalidades o cambios muy significativos.

La Y representan cambios menores, es decir, este número se incrementará una unidad cuando se haya hecho algún cambio significativo en el sistema pero que realmente no cambie en sí la API de la aplicación. Estos cambios serán añadir pequeñas funcionalidades o cambiar algunas funcionalidades sin alterar en gran medida cómo estaban realizadas. Finalmente, también en estos cambios se incluyen la solución de errores no muy esenciales que se hayan producido en el sistema, siempre y cuando este error no haya surgido cuando el sistema está ya en producción.

La Z representa solución de errores, es decir, este número se incrementará en una unidad cuando se haya solucionado un error significativo en el sistema. Estos errores incluyen bugs solucionados una vez el sistema está en producción o la solución de errores significativos cuando ya una funcionalidad estaba completada y terminada (aunque aún no forme parte de la versión en producción.

La primera release tendrá la siguiente forma: 1.0.0. Esta se irá modificando según lo ya explicado anteriormente.