Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática

Configuration Management Methodology

Grado en Ingeniería Informática – Ingeniería del Software

Proceso Software y Gestión II

Curso 2022 – 2023

Fecha	Repositorio
11/2/2023	https://github.com/gii-is-psg2/psg2-2223-g5-52

Grupo de prácticas: G5-52	
Autores por orden alfabético	**Emails**
Daniel Arriaza Arriaza	daniel.arriaza.arriaza@gmail.com
Guillermo Galeano De Paz	guillermo.galeanodepaz@gmail.com
Juan Luis Ruano Muriedas	juanluis.ruano.muriedas@gmail.com
Pedro Jesús Ruiz Aguilar	cocode2002@gmail.com
José Manuel Ruiz Pérez	josruiper4@alum.us.es
Jesús Solís Ortega	jesussolisortega@gmail.com

Índice

1. Control de versiones

Número de versión	Fecha	Descripción
0.1	21/02/2023	Añadido las políticas de mensajes commit
0.2	21/02/2023	Añadido los estándares de código
0.3	21/02/2023	Añadida la estructura de repositorios
0.4	21/02/2023	Añadido la estrategia de ramas
0.5	21/02/2023	Añadido la política de versionado
	22/02/2023	Revisión de apartados

2. Introducción

La Gestión de Configuración es un proceso fundamental en el desarrollo de software que permite rastrear y controlar los cambios realizados en el código fuente y otros artefactos durante todo el ciclo de vida del proyecto. En este documento, se describe la metodología de Gestión de Configuración seguida por nuestro grupo, que puede ser aplicada a diferentes proyectos.

3. Contenido

a) Estándares de código

Los estándares de código son una serie de reglas a seguir por el equipo de trabajo a la hora de crear código. Estas reglas se componen de buenas prácticas a seguir para facilitar la comprensión del código escrito y para mantener la uniformidad del código escrito por diferentes desarrolladores.

Los definidos por el grupo de trabajo G5-52 son los siguientes:

-Uso de camelCase:

A la hora de escribir nombres de métodos, clases, interfaces y variables se usará camelCase al ser el método al que están habituados los miembros del grupo además de ser la sintaxis usada por java.

-Llaves de sentencias:

Al definir sentencias y métodos se abrirán las llaves en la misma línea en la que se defina, además al cerrar se escribirán en una línea separada exceptuando los else/ else if en los que se cerrará la sentencia del if en la misma línea que en la que se cree.

Ejemplo:

```
if(...){

…

}else{

…

}
```

-Nombramiento de variables y métodos:

Durante la creación de nuevas variables o métodos, el nombramiento de estos deberá ser autoexplicativo, es decir, el nombre debe ser tal que al leerlo de suficiente información para comprender su propósito o significado.

-Idioma:

Todo el código deberá ser desarrollado en inglés ya que en la industria del desarrollo software, el inglés es el idioma vehicular. Por tanto, todo comentario o nombre en el código deberá estar en inglés.

-Comentarios:

Evitar el uso de comentarios para mantener el código limpio. Estos se emplearán para aportar la información que se considere necesaria para el entendimiento del funcionamiento de métodos de una complejidad alta. Además al nombrar variables dentro de un comentario se usarán mayúsculas para facilitar la identificación de estas.

b) Políticas de Mensajes de Commit

Un commit es una instantánea del código fuente y otros artefactos que se realizan para guardar los cambios realizados en el repositorio. Es importante que los mensajes de commit sean claros y concisos para que cualquier persona que lea el historial de cambios pueda entender lo que se hizo en ese commit. A continuación, se describen las políticas de mensajes de commit que seguimos en nuestro grupo:

Estructura de los commits:

- La referencia en el Product Backlog de la tarea: esto permite un mejor seguimiento de las tareas realizadas en función de los requerimientos del cliente.
- Usar "Fixes #x", "Refs #x" para autocerrar, referenciar: Si un commit soluciona un problema o referencia una tarea específica, se debe utilizar la palabra "Fixes" o "Refs" seguida del número de la tarea correspondiente. Esto permite que la tarea se cierre automáticamente cuando se fusiona el commit en la rama principal.
- Una descripción concisa de lo que se ha hecho. Esto nos permite asegurarnos de que estamos realizando los cambios correctos y que estamos alineados con los objetivos del proyecto.

Esto nos da la siguiente estructura del commit, que a modo de ejemplo quedaría así:

A2.7.b Fixes #10 Commit message policies

Reglas de redacción:

- Intentar que sean títulos breves, en caso necesario añadir descripción: Los títulos de los mensajes de commit deben ser breves y concisos para que sean fáciles de leer y entender. En caso de ser necesario, se puede añadir una descripción detallada de los cambios realizados en el cuerpo del commit.
- Commits en inglés: Los mensajes de commit deben estar escritos en inglés para que sean accesibles a cualquier miembro del equipo o persona interesada en el proyecto.

c) Estructura de repositorios y ramas por defecto

En cuanto a la estructura que sigue nuestro repositorio, que en general es similar a la mayoría de proyectos, lo explicaremos a continuación.

En la raíz del proyecto podemos encontrar documentos varios de configuración como pueden ser el readme, el pom, configuración referente a git o a maven e información del proyecto. Encontramos también una carpeta con lo referente a maven, otra con configuración de Visual Studio Code, y la carpeta src donde encontraremos todo el contenido del proyecto.

Dentro de src tenemos la carpeta de los test, donde lógicamente irán todos los test de nuestra aplicación, y por otro lado, tenemos main donde se modelará la aplicación siguiendo todos los requisitos que tengamos.

Dentro de main podemos encontrar casi lo más importante del proyecto. Encontramos algunas carpetas de configuración del proyecto como por ejemplo less, appengine o wro. También vemos la carpeta resources, que sería todo lo referente a recursos como imágenes, configuración de base de datos. Otra carpeta es la de webapp donde irán todas las vistas de la aplicación y por último la carpeta samples/petclinic donde modelamos todas las entradas a la base de datos, así como la lógica de la aplicación, controladores, servicios, entidades….

Un repositorio Git es un espacio virtual donde se pueden almacenar proyectos. De esta forma, se puede conservar el código en el repositorio que se irá actualizando con las nuevas versiones que se suban. No solo ofrece la seguridad de una posible pérdida del código, sino que también fomenta el trabajo en equipo sincronizado, de forma que todo el mundo puede disponer de la última versión en cualquier momento y trabajar al mismo tiempo.

Una de las herramientas más importantes que nos ofrece el uso de un repositorio, son el uso de las ramas. Una rama es una bifurcación en el código de forma que el código original se mantiene en una versión intacta. La idea es trabajar siempre en la rama correcta para evitar conflictos en el producto final, así que cuando la nueva versión funciona correctamente se une mediante un merge a su rama original para llevar allí el cambio.

En cuanto a nuestro modus-operandi, contaremos con unas ramas por defecto que serán por así decirlo las ramas troncales más importantes. Hay que tener en cuenta que a parte de estas ramas que mencionaremos a continuación existirán también muchas otras, que será donde implementaremos todas las features, se solucionarán errores, se modificarán documentos, etc.

- Main: Esta es la rama principal del proyecto, de la que parten todas las demás, y la única existente cuando el proyecto se crea. En esta rama no se implementa nada, la idea es que se trabaje en otras ramas y ésta sólo sirva para almacenar el producto final que será lo que se entregue al finalizar un sprint. Entonces la última versión del proyecto llegará a

Main con la seguridad de que no se modificará nada del producto, pues éste funcionará correctamente.

- Develop: Es una rama que se usa para recopilar todas las versiones del proyecto que se vayan implementando en otras ramas específicas para cada tarea. De esta forma se van solucionando los conflictos que se pueden generar al unir todas las versiones, con la idea de tener el producto final funcionando correctamente en esta rama.
- Release: Esta rama funciona como una especie de cortafuegos entre Main y Develop. Cuando tenemos la última versión del producto final funcionando sin problemas en Develop, lo pasamos a la rama Release, a modo de detectar posibles errores o bugs que se hayan podido escapar. En el momento en que todo se haya testeado y funciona al 100%, entonces actualizaremos Main con ésta nueva versión libre de fallos, para tener en Main el producto final.

d) Estrategia de ramas

En nuestro proyecto vamos a aplicar la estrategia basada en Git Flow para ello vamos a detallar algunas ramas adicionales a Main, Release y Develop que complementan nuestro repositorio:

-Feature: en esta rama se añade alguna utilidad funcional que describen los requisitos, el nombre de la rama vendrá dado por “Feature/X – AY” siendo X el número de la issue e Y el requisito del sprint backlog al que hace referencia la funcionalidad, y si fuera necesario, a que subtarea del apartado hace referencia.

-Hotfix: esta rama sirve para arreglar problemas de código urgentes en la rama main, el nombre de la rama vendrá dado por “Hotfix/X – AY” siendo X el número de la issue e Y la funcionalidad del sprint backlog donde se ha encontrado el problema.

-Docs: similar a la rama Feature, esta rama se utiliza exclusivamente para trabajar sobre los documentos del repositorio, el nombre de la rama vendrá dado por “Docs/X – AY” siendo X el número de la issue e Y el apartado del sprint backlog que precise la realización del documento, y si fuera necesario, a que subtarea del apartado hace referencia.

En cuanto al flujo de trabajo, el proyecto inicial partirá de la rama Main como proyecto base y seguidamente se creará la rama Develop de donde saldrán las ramificaciones de tipo Feature o Docs en las que trabajará el equipo para crear las funcionalidades o documentación y posteriormente revisarlas. La metodología de revisión del trabajo que aplicaremos será en cadena, es decir, en una lista ordenada alfabéticamente de los miembros del equipo el trabajo realizado por cierto miembro será revisado por quien esté situado directamente debajo de dicha lista, si todo cumple con los requisitos pasará a añadirse a la rama Develop. Una vez añadidas todas las funciones y documentos de una versión sobre la rama Develop se añadirán a una rama Release donde se prepara la versión final y se hacen los arreglos pertinentes previos a la adición en la rama Main. El nombre de la rama Release vendrá dado por “Release X.Y” siendo X e Y el número de la versión que cierra la rama usando la política de

versionado que se define más adelante. Por último, cuando sea necesario a lo largo del ciclo del proyecto se podrá crear una ramificación sobre la rama Main de tipo Hotfix para arreglar problemas urgentes de la versión final.

e) Política de versionado

Respecto a las políticas de versionado del proyecto se usará el número de versión mayor 0 para referirse al contenido anterior a la entrega del sprint S2, cada tarea del product backlog añadirá un 1 al número de la versión menor, por ejemplo si se añade una funcionalidad a la versión 0.0 pasaría a ser la 0.1, así como, si se completan todos las tareas del product backlog o de cualquier product backlog pasará a sumarse 1 en el número de la versión mayor siendo el ejemplo de que al acabar el S2 pasará de ser la versión 0.X a la 1.0. Se seguirán estas dos pautas para versionar en futuras entregas, no se hace uso del tercer número el número de parches para una mayor simpleza y transparencia, también se usarán tags pero solo para referirse a una versión mayor nueva y se llamará como el entregable al que haga referencia en este caso al alcanzar la versión 1.0 se le llamará S2.

4. Conclusiones

En este documento, hemos descrito la metodología de Gestión de Configuración seguida por nuestro grupo. Hemos discutido los estándares de codificación, las políticas de mensajes de commit, la estructura de repositorios y ramas predeterminadas, la estrategia de branching basada en Git Flow y la política de versionado. Al seguir estas prácticas, esperamos ser capaces de controlar los cambios realizados en el código fuente y otros artefactos, lo que nos permitirá desarrollar software de alta calidad de manera eficiente y efectiva. Esperamos que esta metodología pueda ser aplicada en diferentes proyectos para mejorar su Gestión de Configuración y obtener mejores resultados.

5. Bibliografía

- Proceso Software y Gestión II: S2 – Configuration Management. 2.3 Methodologies and Best Practices. Autor: Departamento de Lenguajes y Sistemas Informáticos. Ed: Universidad de Sevilla.
- Proceso Software y Gestión II: S2 – Configuration Management. 2.4 Applying a Branching Strategy. Autor: Departamento de Lenguajes y Sistemas Informáticos. Ed: Universidad de Sevilla.