Universidad Nacional de Córdoba

Facultad de Ciencias Exactas, Física y Naturales

Plan de manejo de las Configuraciones

Docentes:
Miceli, Martín.
Bustos, Martin

Grupo:"Drink Team"

Integrantes:

- Depetris, Stefano
- Gauna, Macarena del Sol
- Nieto, Marcos

Plan de manejo de las Configuraciones

Versión	Fecha	Resumen de cambio	Autor
1.0	30/04/2020		• Depetris, Stefano • Gauna, Macarena • Nieto, Marcos
1.1	10/06/2020	Correcciones.	• Depetris, Stefano • Gauna, Macarena • Nieto, Marcos

Introducción:

Este documento contiene el plan de manejo de las configuraciones para el proyecto de un sistema control de turnos de un consultorio oftalmológico. En el mismo, se detalla los documentos, programas y herramientas utilizadas para la realización de nuestro proyecto. Como se dijo anteriormente consistirá en sistema que se podría implementar en una clínica de ojos. El mismo estará basado en una agenda en la cual se gestionarán las reuniones entre pacientes y profesional verificando horarios y días disponibles. Se podrán agregar cambiar y quitar turnos.

1) Control de Versiones:

Un control de versiones es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante.
Para nuestro software,se usa GitHub como herramienta y las direcciones de los repositorios son:

Descripción	**Link**
Repositorio remoto	https://github.com/StefanoDepetris/DrinkTeam
Herramienta de seguimiento de defectos	https://github.com/StefanoDepetris/DrinkTeam/issues
Repositorio de archivos de tutorial	https://github.com/StefanoDepetris/DrinkTeam/wiki

2) Herramienta de Integración Continua:

La integración continua es una práctica del desarrollo del software que realiza integraciones automáticas del código desarrollado por cada programador del equipo. Esto permite detectar errores de integración lo antes posible.

Para nuestro software se utilizará Travis CI.
Travis-CI es un sistema de Integración Continua, gratuita para proyectos Open Source y de pago para proyectos privados. Se integra sin problemas con GitHub y automáticamente ejecuta el pipeline definido en cada push o pull requests.

https://travis-ci.com/github/StefanoDepetris/Dgn2020

3) Herramienta de gestión de defectos y gestión de tareas:

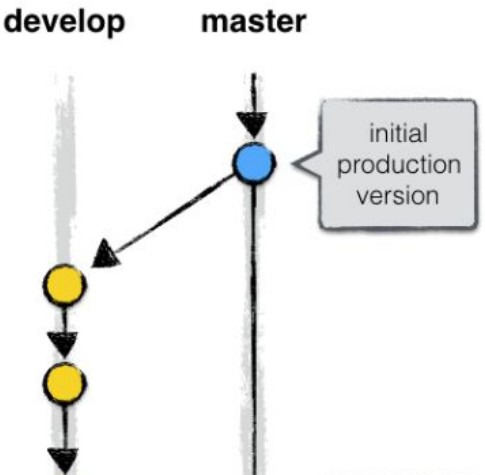
Se utilizará la sección “Issue” que nos ofrece Github para que tanto los desarrolladores como los usuarios colaboradores que lo deseen, nos informen si detectan algún error, falla o mala implementación en alguna parte del código o documentación. La información en esta sección será detallada en forma de lista.
Además se utilizará la misma para realizar la gestión de tareas, estas tareas son actividades que deben ser completadas lo antes posible y estarán bajo la responsabilidad de un encargado en particular. Es decir, cada integrante del grupo tendrá asignadas algunas tareas a resolver a lo largo del proyecto.
A medida que las tareas se van realizando, se irán cerrando las mismas pero quedarán en el historial que nos brinda la herramienta.
También, los integrantes del grupo, tendrán acceso y permisos de edición a la documentación del proyecto en cualquier momento utilizando la herramienta de Google Drive llamada “Docs”. De esta forma se pueden observar los cambios efectuados en tiempo real y se hace mucho más práctica y llevadera la gestión de tareas relacionada con los documentos del proyecto. Luego de que se resuelvan algunas de las mencionadas anteriormente, se realiza la actualización utilizando el control de versiones.

https://github.com/StefanoDepetris/DrinkTeam/issues

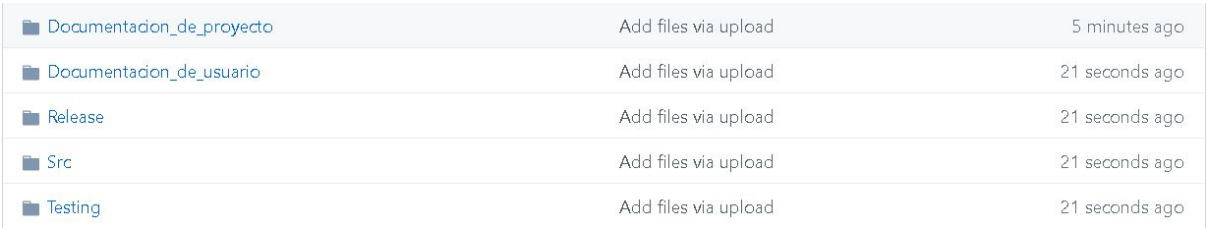
4) Esquema de directorios:

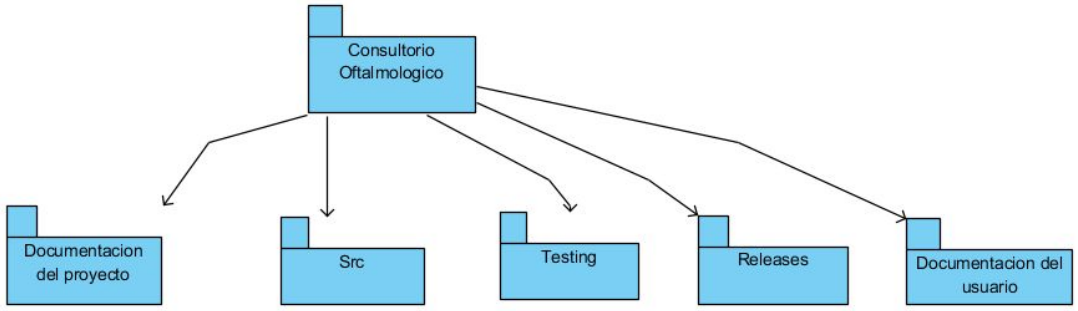
Para que cada programador puede desarrollar su trabajo sin poner en peligro la version estable del programa se procede a ramificar por cada tarea, luego unificar la misma en caso de que pase todos los test correspondientes y sea aprobado por los demás.
En caso de que se encuentra un problema crítico para resolver entre todos se pueden seguir los siguientes pasos:

1. El programador vuelve desde dónde esté a la rama original.
2. Entra a su rama personal para poder solucionar dicho problema o crea una nueva con el nombre de la funcionalidad a corregir.
3. Tras las pertinentes pruebas, fusionará esa rama y la enviará a la rama original.



Directorios en github:





Documentación_de_proyecto	https://github.com/StefanoDepetris/DrinkTeam/tree/master/Cosultorio_Ofmaltologico/Documentacion_de_proyecto
Documentación_de_usuario	https://github.com/StefanoDepetris/DrinkTeam/tree/master/Cosultorio_Ofmaltologico/Documentacion_de_usuario
Releases	https://github.com/StefanoDepetris/DrinkTeam/tree/master/Cosultorio_Ofmaltologico/Release
Src-Codigo fuente	https://github.com/StefanoDepetris/DrinkTeam/tree/master/Cosultorio_Ofmaltologico/Src
Testing	https://github.com/StefanoDepetris/DrinkTeam/tree/master/Cosultorio_Ofmaltologico/Testing

En la primera carpeta se tendrán todos los documentos asociados al diseño y planeamiento del proyecto, como por ejemplo, el Plan de Gestión de Configuraciones, el Documento de Requerimientos, el Documento de Casos de Prueba y el Documento de Arquitectura.

En la segunda, se tendrá el o los archivos ejecutables,datos y documentación del software, por ejemplo información o instrucciones de uso, imágenes y modelos explicativos.

En la tercera se tendrán los distintos releases en el caso de que fueran varios y la información de sus arreglos en caso de que hubiera algo para solucionar.

En la carpeta de Src se tendrá el codigo fuente del software, es decir, las distintas clases del sistema con su correspondiente implementación de código e información complementaria si se requiere.

Y en la última carpeta se encontrarán los distintos unit tests que se han creado para verificar el correcto funcionamiento del software y toda información pertinente a las mismas.

5) Normas de Etiquetado y Nomenclatura de Archivos:

Las distintas instancias del programa o versiones, tendrán distintas etiquetas para poder diferenciar unas de otras. Estas etiquetas serán numeradas utilizando la notación de tres números separados por puntos y precedidos por la letra “V”. Ejemplo: v2.8.2

Su significado será:

- **El tercer dígito**: Representa la corrección de errores y bugs.
- **El segundo dígito:** Representa modificaciones funcionales, es decir se han añadido, eliminado o modificado funcionalidades al código.
- **El primer dígito:** Representa cambios mayores en el diseño del código.Por ejemplo algún cambio en los patrones de diseño o en el enfoque del software. Este primer dígito estará en 0 mientra que el programa esté en desarrollo.

Las distintas instancias de la documentación del proyecto también tendrán sus correspondientes etiquetas para poder diferenciar unas de otras. Estas etiquetas serán numeradas utilizando la notación de dos números separados por puntos y precedidos por la letra “V”. Ejemplo: V2.8.

Su significado será:

- **El segundo dígito:** Representa los cambios menores que hemos hecho en los documentos. Por ejemplo falta de información en alguna sección, errores de redacción, información desactualizada, entre otros.
- **El primer dígito:** Representa cambios mayores en la estructura del documento. Por ejemplo la creación de nuevas secciones, la eliminación de una determinada sección y nuevos enfoques de la información.

6) Plan del esquema de ramas a usar:

Rama de integración:

La denominada master o head se define como la rama de integración principal donde se guardarán todas las funciones que hayan sido integradas y estén funcionando según nuestro criterios. Esta rama viene por default en el sistema de control de versiones cuando se agrega un archivo al control de origen.

Rama de desarrollo privado:

Son aquellas ramas creadas por los desarrolladores/programadores para el desarrollo de nuevas características y funcionalidades, así como el tratamiento de bugs. Esta rama representa o simula un espacio individual de trabajo y de esta forma pueden trabajar varios programadores a la vez sin entorpecer el trabajo de los demás. Cada una de las ramas tiene además un control de versiones y cada programador se asegura de mantener su rama libre de problemas para luego llevarla a la rama master.

Ramas experimentales:

Serán aquellas ramas que se crearán a partir de la rama principal cuando se requiera un hotfix. Ésta cuando sea creada será la de mayor prioridad para los desarrolladores/programadores para que sea solucionado lo antes posible.

7) Políticas de fusión de archivos y de etiquetado de acuerdo al progreso de calidad en los entregables:

Para fusionar nuevas ramas que pudieran surgir del código principal, el administrador realizará la fusión o merge haciendo un rebase del código junto a los desarrolladores. De esta forma rápidamente se podrán identificar errores, los desarrolladores podrán trabajar en resolverlos y el administrador estará al tanto de la situación.

Para realizar un Merge, se debe especificar ciertas etiquetas. Los tipos de etiquetas son los siguientes:

- **Etiquetas raíz:** Cada vez que un programador crea un nueva rama, una etiqueta con formato Raiz _<nombre_rama> será creada en la versión fuente desde donde se crea dicha rama.
- **Etiquetas de Merge**: Por lo general se aplican cada vez que se realice un Merge en las versiones originales.

  El formato es merge-from-<source>-<yyyymmdd>-<nn>.

- **Etiqueta Pre-Merge**: Este tipo de etiquetas es opcional. Se sugiere que se usen cuando se va a realizar un Merge muy completo. La idea es aplicar esta etiqueta solo en las versiones que van a recibir algún tipo de cambio antes que se realice la

función, entonces podremos saber cual es la versión estable (si por algún motivo dicha versión no funciona mas).

El formato del nombre es pmerge-from-<source>-<yyyymmdd>-<nn>.

Estas etiquetas se utilizan para identificar las versiones que se fusionaron, por lo que en caso de que se necesite otra combinación en esa misma rama, podemos usar la última etiqueta de fusión aplicada en la combinación anterior como la versión base común (etiqueta de inicio) y así evitar tratar con los mismos conflictos que resolvimos en la combinación anterior. También, opcionalmente, se puede aplicar una etiqueta de Pre-merge en caso de que se tenga que lidiar con una combinación muy compleja para poder identificar las versiones estables antes de realizar la fusión.

8) Forma de entrega de los “releases”

Una vez que tengamos terminado el código y hayamos finalizado los diferentes test, se procederá a la compilación donde el compilador generará un archivo ejecutable Java.
Este archivo puede ser ejecutado por un usuario siempre y cuando tenga instalado en el equipo la última versión del sistema de Java Runtime Environment(JRE)
En el caso de no cumplir con el paso anterior, el mismo puede descargarse de la página oficial: https://www.java.com/es/ . Esto será especificado a los clientes en los pasos a seguir para la instalación.

El archivo ejecutable se encargará de:

- Creación de los directorios requeridos.
- Creación de la interfaz de usuario.
- Copia, desempaque y descompresión de los archivos desde el paquete de software.
    - Archivos principales.
    - Archivos de datos y documentación, por ejemplo información de uso, imágenes y modelos explicativos para hacer menos tedioso el aprendizaje del software.
    - Archivos de configuración
    - Bibliotecas
- Compilación y enlace con la bibliotecas requeridas.
- Configuración.

9) Change Control Board:

El CCB es una Junta de Control de Cambios compuesta por expertos en la materia y jefes técnicos, que tomarán decisiones sobre si los cambios propuestos o no en el proyecto de software debe ser implementado.
Cualquier cambio en los requisitos de línea de base acordados con el cliente debe ser asumido por el equipo del proyecto con la aprobación de este comité. Aquí se consideran

los efectos de los cambios desde un punto de vista organizacional y estratégico. Son los que deciden si dicho cambio está aprobado o no antes de implementarlo. Al momento de la solicitud de un cambio, el CCB decidirá si es factible el mismo con la ayuda de un análisis preliminar evaluando la importancia del cambio, usuarios afectados, impacto de su aprobación o desaprobación, costos y tiempos estimados. El siguiente esquema muestra el tipo de organización y metodología que usa el CCB cuando se solicita un cambio.

- Tiempo estimado para realizar el cambio.
- Fecha de solicitud del cambio, de la aprobación y de la implementación.
- Pruebas y reporte de pruebas.
- Costos estimados del cambio.
- Componentes afectados.
- Prioridad del cambio.
- Cambio requerido.
- Solicitante del cambio.

Integrantes	Roles en CCB	Forma de contacto
Depetris, Stefano	Colaborador de proyecto general: ● Programador ● Diseñador ● Analista	● Redes sociales ● Reuniones vía videollamada ● Correo electrónico ● Contacto Telefónico
Gauna,Macarena	Colaborador de proyecto general: ● Master ● Programador ● Diseñador ● Analista	● Redes sociales ● Reuniones vía videollamada ● Correo electrónico ● Contacto Telefónico
Nieto, Marcos	Colaborador de proyecto general: ● Programador ● Diseñador ● Analista	● Redes sociales ● Reuniones vía videollamada ● Correo electrónico ● Contacto Telefónico

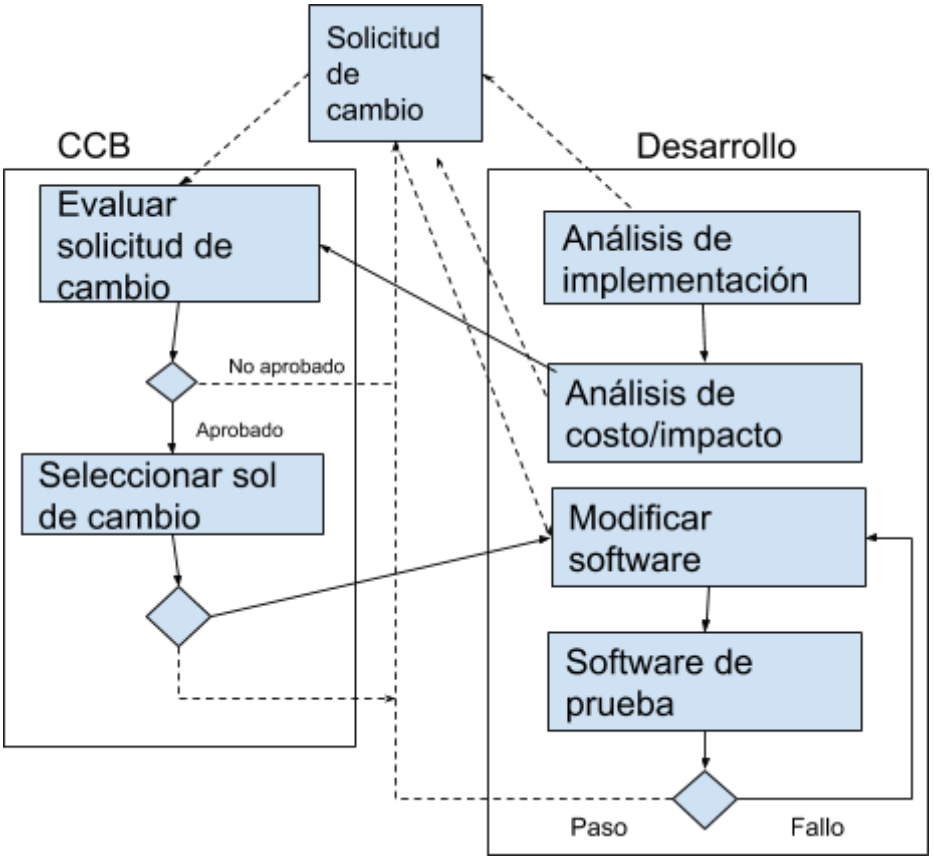
Colaborador de proyecto general: Por ser un grupo de 3 personas y al tratarse de un proyecto relativamente pequeño, todos los miembros del grupo podrán participar de la totalidad del proyecto y todas las etapas del desarrollo de software.

- ***Programador***: Aquellos que se encargan del código propiamente dicho.

- ***Master***: Vigila el cumplimiento de la metodología. Define las prioridades, objetivos y roles de cada uno.
- ***Analista***: El analista es el encargado de entender las necesidades del cliente y asegurarse que el proyecto va encaminado para cumplir dichas necesidades.
- ***Diseñador***: Encargado de la disposición general de una aplicación. Ya sea diseñando completamente la interfaz de usuario de la agenda de turnos hasta alguna directriz de interfaz de usuario.

Las reuniones en este momento son solo vía teleconferencia debido a la pandemia que es de público conocimiento y se realizan varias veces por semana, en principio, con todos los miembros del equipo.

Mecanismo de seguimiento de los cambios solicitados por los clientes es el siguiente:



Con el fin de evaluar el impacto y el costo de hacer o no hacer un cambio tomando como decisión final si vale la pena hacerlo o no se lleva a cabo este proceso que se ven el diagrama.Dicho proceso es debidamente documentado y aprobado.

Bibliografía:

https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Acerca-del-Control-de-Versiones

https://git-scm.com/book/es/v2/Ramificaciones-en-Git-Procedimientos-B%C3%A1sicos-para-Ramificar-y-Fusionar

https://en.wikipedia.org/wiki/Change_control_board