MIND GAME

PLAN DE GESTIÓN DE LAS CONFIGURACIONES

Autores: Sosa Ludueña Diego
Sleiman Mohamad
Choquevilca Gustavo

Versión del Documento: 1.0.0

Revisión Histórica

Version	Fecha	Resumen del Cambio	Autores
1.0.0	30/04/2017	Versión base	Sosa Ludueña Diego, Sleiman Mohamad, Choquevilca Gustavo

Tabla de Contenidos

Introducción

Cuando se quiere realizar un producto de software, es necesario previo al diseño y a la implementación realizar un plan de gestión de las configuraciones. Nuestro proyecto es realizar una aplicación de escritorio, más precisamente un juego llamado **MindGame**, por lo que empezaremos por el manejo de las configuraciones. Para realizar el manejo de las configuraciones va hacer necesario la creación de este documento, el cual se usará para nuestro proyecto. El mismo incluye el seguimiento del control de versiones, la integración continua, la gestión de defectos, el esquema de directorios, norma de etiquetado de archivos, esquemas de ramas a usar, configuración de la CCB, entre otros puntos.

Control de Versiones

Antes de empezar a hablar sobre el sistema de control de versiones, es necesario dejar en claro que nuestro proyecto se debe almacenar en un sitio web llamado repositorio. El mismo guardará todos los directorios y archivos de nuestro proyecto. En este caso optamos por utilizar *Github* debido a que es un repositorio gratuito. Para acceder al mismo debemos ingresar al enlace siguiente Github. En dicho sitio se debe ingresar con la cuenta de usuario y su respectiva contraseña (el sitio proporciona los pasos para generar una cuenta en caso de no tener). Por lo tanto, ahora sabiendo en donde se almacenará nuestro proyecto debemos utilizar alguna herramienta que controle y realice el seguimiento de las versiones de todos los componentes de nuestro proyecto. Esta herramienta que utilizaremos en este caso es llamada *Git.* Para acceder a la misma debemos descargar e instalar la herramienta *Git.* Para descargar *Git* debemos entrar al sitio web oficial DescargarGit y la descarga comenzará automáticamente. Una vez descargado se procede a instalar *Git* siguiendo una serie de pasos los cuales son muy intuitivos, para poder instalarlo (de esta forma se obviara describir dichos pasos). Ahora que ya tenemos instalado la herramienta debemos ejecutarla (la herramienta se instala con el nombre de *Git bash*), donde se abrirá una consola. Mediante el uso de la consola podremos interaccionar con nuestro proyecto, ya sea subir una version nueva al repositorio o bajar una versión de otro integrante del grupo y muchas más funcionalidades. Debemos proporcionar el link de nuestro proyecto el cual es GithubProyecto, el mismo se redireccionará al repositorio de nuestro proyecto.

Integración Continua

Es necesario que además de tener un sistema que controle las versiones que haga cada integrantes del grupo de trabajo, también haya un servicio que garantice que dichas versiones del código fuente no generen retrasos ni anomalías en el desarrollo de nuestro proyecto. Por lo tanto este servicio es llamado integración continua, el cual cada vez que un integrante realice una integración (commit), dicho servicio se asegurará de compilar y testear automáticamente cada integración que se haga. Nosotros usaremos la herramienta *TravisCI* ya que es sencilla de usar y también en este caso es un servicio gratuito para proyectos Open Source. Este servicio se asocia con nuestra cuenta de *Github* donde debemos ingresar a TravisCI y luego autorizar a Travis a acceder a nuestros repositorios. Por lo tanto ahora activamos la integración continua de nuestro proyecto para poder utilizar el servicio, paso siguiente mostramos el enlace del mismo TravisCIProyecto. Luego antes de empezar a desarrollar nuestro proyecto será importante cargar nuestro archivo .travis.yml en el directorio raíz de nuestro repositorio. Cada vez que se

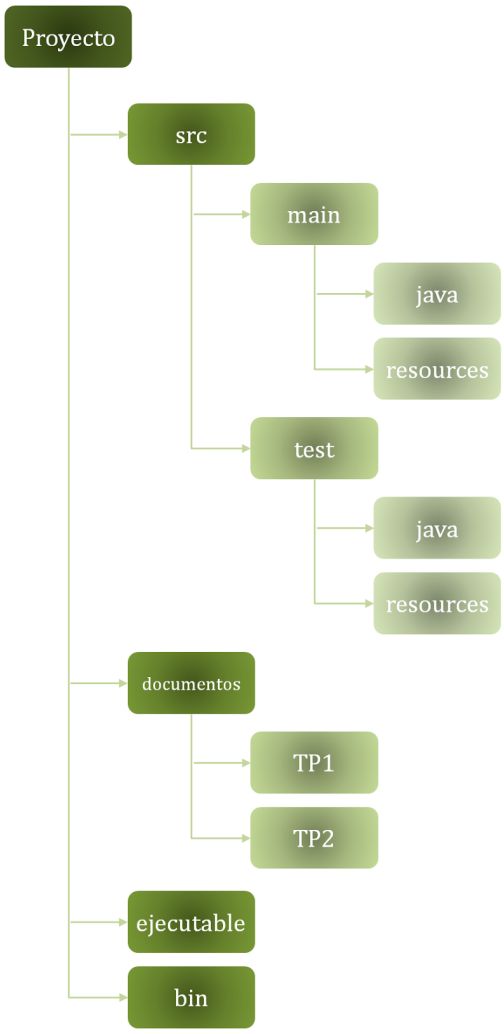
haga un push desde *Git* se iniciara una verificación de nuestro proyecto según las configuraciones estipuladas.

Gestión de Defecto

La gestión de defectos es la herramienta que facilita el registro y seguimiento del estado de los defectos. A menudo incorporan facilidades orientadas al flujo de trabajo para el seguimiento y control de la asignación, corrección y repetición de pruebas de los defectos y proveen facilidades para la realización de informes. En nuestro caso utilizaremos *Issue* ya que lo provee *GitHub*. Un issue es la unidad de trabajo designada para realizar una mejora en un Sistema informático (en nuestro caso a un proyecto). Puede ser el arreglo de un fallo, una característica pedida, una tarea, una solicitud de documentación en específico y todo tipo de solicitud al equipo de desarrollo. Con las issues se puede asignar una tarea a un colaborador del repositorio o proyecto. Estos issues tienen etiquetas, las cuales sirven para poder filtrar la búsqueda de estas y sirven precisamente para indicar la idea principal que esta tarea implica. Por lo tanto para acceder a un issue se debe ingresar al siguiente enlace IssueProyecto.

Esquemas de Directorios y Propósito de Cada Uno

Para el esquema de directorios se planteó realizarlo de la siguiente manera:



Como vemos la carpeta *Proyecto* es el directorio raíz, es el primer directorio o carpeta en una jerarquía. Contiene todos los subdirectorios de la jerarquía. La carpeta *src* es un directorio donde se encuentra toda la lógica del programa. Esta carpeta contiene dos directorios el *main* y *test* cada una de ellas tendrán a su vez el directorio *java* que contendrá las clases generadas de la aplicación y el directorio *resources* que contendrá todos los archivos con los recursos que usa la aplicación. Luego se tendrá el directorio *documentos* que tiene las carpetas *TP1* y *TP2* donde se guardaran toda la documentación para el caso de *TP1* contendrá el documento "plan de gestión de las configuraciones" y el "documento de requerimientos" y para el caso del *TP2* contendrá el documento "diseño, implementación y pruebas". Luego tenemos el directorio *ejecutable* que contendrá el archivo de ejecución. Y por último tenemos el directorio *bin* donde se encuentran todos los archivos generados por la propia aplicación.

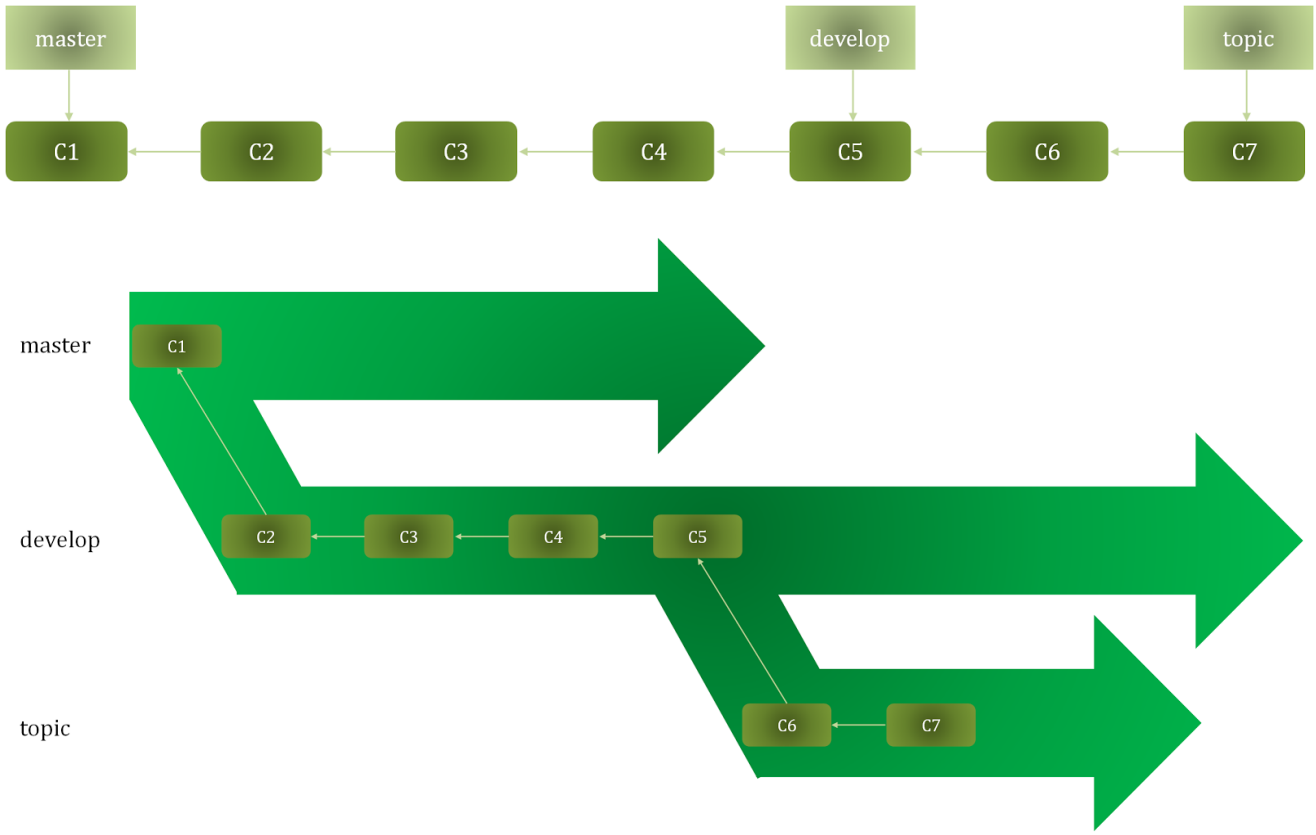
Norma de etiquetado y nombramiento de los archivos

Para este proyecto se seguirán unas normas de versionado (también llamada versionamiento semántico), las cuales se utilizan por lo general en proyectos *Open Source.* Por lo tanto siguiendo estas normas, las versiones las numeramos con la siguiente nomenclatura *X.Y.Z* (también nombradas como *MAYOR.MENOR.PATCH*). Tanto *X,Y* y *Z* son números enteros no negativos. Cada elemento debe incrementarse numéricamente en incrementos de 1. Por lo tanto se especificará cada uno de estos elementos:
X (MAYOR): debe ser incrementada cuando cualquier cambio no es compatible con la versión anterior. Puede incluir cambios de nivel menor y/o patch. Las versiones patch y menor deben ser reseteadas a 0 cuando se incrementa la versión mayor.
Y (MENOR): debe ser incrementada si se introduce nueva funcionalidad compatible con la versión anterior. Se debe incrementar si cualquier funcionalidad es marcada como deprecada. Puede ser incrementada si se agrega funcionalidad o arreglos considerables al código privado. Puede incluir cambios de nivel patch. La versión patch debe ser reseteada a 0 cuando la versión menor es incrementada.
Z (PATCH): Debe incrementarse cuando se introducen solo arreglos compatibles con la versión anterior. Un arreglo de bug se define como un cambio interno que corrige un comportamiento erróneo.
La numeración de las versiones de los entregables (release) software permiten expresar de forma normalizada las modificaciones existentes entre las distintas entregas, indicando los cambios que se han producido entre una versión y la siguiente.

Plan del esquema de ramas a usar

El esquema de ramas a utilizar es el de ramas de largo recorrido, el fusionar de una rama a otra multitud de veces a lo largo del tiempo es fácil de hacer. Esto nos posibilita tener varias ramas siempre abiertas, e irlas usando en diferentes etapas del ciclo de desarrollo; realizando frecuentes fusiones entre ellas. En este esquema se identifica una rama "master" que es la rama en la cual se mantiene únicamente el código que es totalmente estable, es decir, el código que se puede liberar o ha sido liberado. También tenemos otra rama paralela denominada rama de "develop" en las que se realizaran pruebas y modificaciones, no necesariamente el código de esta rama será totalmente estable, pero cuando lo estén estables se fusionaran con la rama master. Por último se tendrá una rama denominada "topic", habitualmente es una rama que tiene

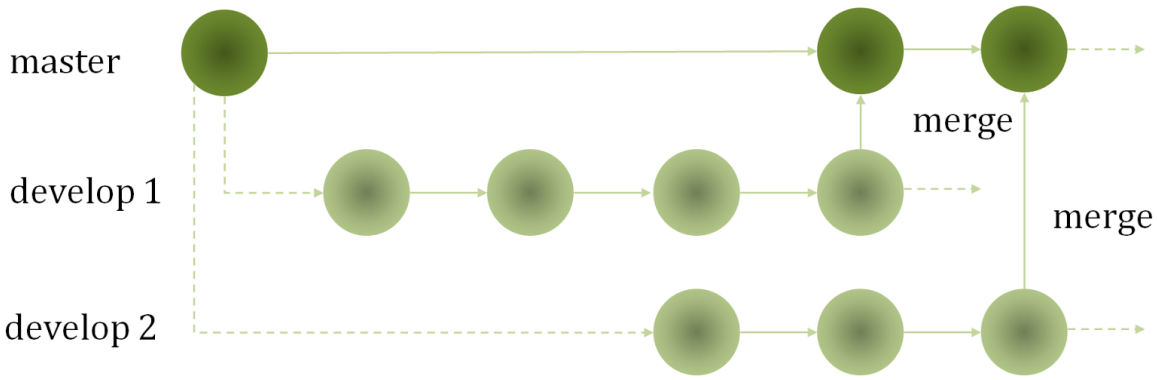
una vida muy corta y se cierran después del merge. Es utilizada cuando se necesitan realizar cambios internos como lo son por lo general la solución de bugs de una cierta versión.
Las siguientes imágenes muestran un ejemplo que es aplicado en *Git*.



Notamos que en todo momento hablamos de apuntadores moviéndose por la línea temporal de confirmaciones de cambio (commit history). Las ramas estables apuntan a las posiciones más antiguas del registro de confirmaciones. Mientras que las ramas avanzadas apuntan hacia las posiciones más recientes.

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

La idea es mantener siempre diversas ramas en diversos grados de estabilidad; pero cuando alguna alcanza un estado más estable, la fusionamos con la rama inmediatamente superior a ella. Esta estrategia es llamada estrategia *Mainline Branch*, es simple y bastante efectiva. Solo una rama principal, cada desarrollador crea una rama para trabajar en una funcionalidad y al terminar *mergean* con la rama master. La rama master siempre está lista para subir. En otras palabras, la rama master solo contiene código de calidad, testeado y nunca estará rota. Esto se podría mostrar en un ejemplo de lo que queremos realizar:



Formato de entrega de los "releases", instrucciones mínimas de instalación y formato de entrega

Es importante saber que nuestro producto de software, es un producto genérico, es decir, que es un sistema independiente que podrá ser vendido a cualquier cliente que desee comprarlo. Dicho esto, es esencial preparar nuestro software para la entrega externa y hacer un seguimiento de las versiones de nuestro sistema que se entregaron para uso del cliente. Por lo tanto a medida que nuestro software evolucione en el transcurso, tendremos versiones del software lista para ser entregada a los clientes. Estas versiones de software listas para entregar al cliente son llamadas *releases* y en nuestro caso se entregarán cada vez que se avance en una versión mayor (estas son las versiones potencialmente entregables). Estas entregas serán distribuidas a los clientes en formato comprimido ejecutable *.jar.* Debido a que nuestro proyecto será un producto de software el mismo respetara los requisitos mínimos de entrega el cual además de contar con el archivo *.jar* también tendrá la documentación asociada la cual estará contenida en un documento de texto *.txt.* Al utilizar un formato ejecutable no es necesario una instalación, solo se debe ejecutar el archivo *.jar* haciendo doble clic sobre el mismo, también es necesario contar con la última versión de java la cual se puede descargar del siguiente enlace DescargarJava.

Change Control Board. Propósito, lista y forma de los integrantes del equipo y su rol en la CCB, periodicidad de las reuniones, etc

Muchas veces tanto los desarrolladores como los clientes realizan peticiones ya sea para cambiar alguna funcionalidad del sistema como también para reportar algún error o falla que descubrieron del sistema. Por lo tanto cualquiera de ellos podrá enviar un documento describiendo y detallando está peticiones, el cual es llamado CRF. El encargado de recibir y analizar estos CRF es la CCB. Como ya dijimos tiene el propósito de analizar si el cambio al software de nuestro producto es rentable. Los integrantes del CCB que en este caso seremos nosotros tendremos que revisar y aprobar todas las peticiones de cambio, a menos que los cambios impliquen simplemente corregir errores menores en pantalla de despliegue o documentos. Si se detectan estas peticiones menores tendremos que transmitirle al equipo de desarrollo sin un análisis detallado, pues esto podría ser más costoso que implementar el cambio. Una vez que nuestra junta decida aceptar los cambios esto se transmitirán de regreso al grupo de desarrollo, las peticiones de cambio rechazadas se cierran y ya no se emprenderán más acciones.
La siguiente tabla muestra los miembros del equipo que asisten a las reuniones técnicas del CCB. Todos los miembros deben asistir obligatoriamente a las reuniones.

Integrantes	CCB rol	Mail
Sosa Ludueña Diego	Developer/Admin	diegosl1294@gmail.com
Sleiman Mohamad	Developer/Tester	hamu738@gmail.com
Choquevilca Gustavo	Developer/Tester	g.choqeuvilca@gmail.com

La periodicidad de las reuniones será:

CCB Reunión	Frecuencia
nombre del proyecto CCB	Las reuniones se realizarán una vez a la semana o en base a la demanda por si los usuario o los desarrolladores sugieren cambios se llamará a reunión lo mas antes posible

Herramienta de seguimiento de defectos usada para reportar los defectos descubiertos y su estado. Forma de acceso y dirección

La herramienta de seguimiento de defectos (BUG Tracking System) que usaremos será *issues* de *Github*, esta herramienta nos permitirá realizar un seguimiento de las tareas y errores para nuestro proyecto. Su interfaz es parecida a un correo electrónico salvo que puedan ser compartidas y discutidas con el resto del equipo.

En términos generales esta herramienta cuenta con las siguientes características:

- Seguimiento de problemas de *Github* issues es especial debido a nuestro enfoque en la colaboración, referencias, y un excelente formato de texto.
- Un título y descripción describen de qué trata el tema.
- Existen etiquetas que son una gran manera de organizar diferentes tipos de problemas.
- Codifica etiquetas por colores que ayudan a clasificar y filtrar los temas.
- Un miembro del proyecto es asignado y responsable de trabajar sobre un tema en un momento dado.
- Existen grupos de temas que corresponden a un proyecto, característica o periodo de tiempo esto se llaman milestones
- Permite comentarios de cualquier miembro del repositorio para proporcionar retroalimentación.
- Los milestones las etiquetas y los asignados son grandes características para filtrar y clasificar los problemas.
- El administrador puede cambiar o añadir un milestones un asignado y una etiqueta haciendo click en sus engranajes correspondientes en la barra lateral a la derecha.
- Existen notificaciones que es la forma de github para mantenerse al día con los problemas, cada miembro puede utilizarlas para obtener información sobre nuevos temas en los repositorios o simplemente para saber cuando alguien necesita su opinión para avanzar en un problema. Estas notificaciones pueden recibirse por correo electrónico.

- La forma de acceso será en el repositorio de *Github* donde estará contenido nuestro proyecto, para acceder al Issues tendremos que acceder a la pestaña *issues* y allí realizar los seguimientos de defectos.