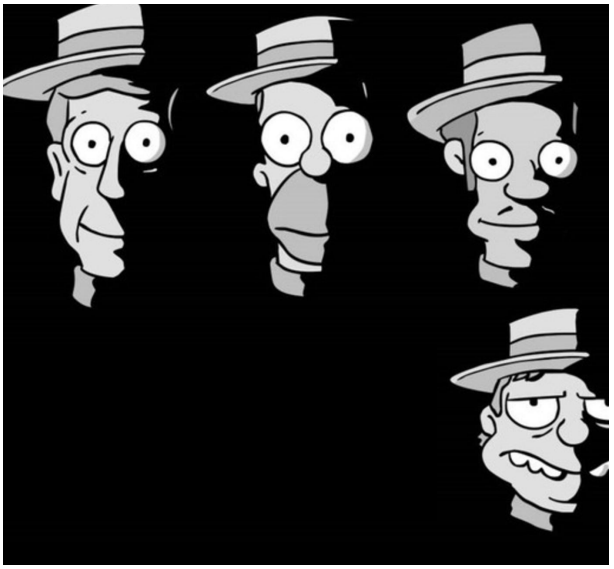
Los Borbotones

Llavero Digital

Configuration Management Plan



Autor(es):

Lorenzo, Facundo

Ojeda, Gastón

Riba, Franco

Roig, Patricio

Versión actual del documento: 1.0

Fecha:23/04/2022

Índice

Historial de Cambios

Versión	Fecha	Detalle	Autor
0.0	04/23/2022	Tomamos la decisión de hacer un llavero digital.	Los Borbotones - Full Team

Glosario de Acrónimos

Acrónimo	Descripción
CI	Continuous Integration
IDE	Integrated Development Environment
CR	Change Request

Referencias

Referencia	Descripción	Enlace
Organización	Perfil del grupo de desarrolladores	https://github.com/LosBorbotonesISOFT
Repository	Repositorio principal del proyecto	https://github.com/LosBorbotonesISOFT/TP_Final_ISOFT

1. Introducción

1.1. Propósito y Alcance

Este documento presenta el plan de Gestión de las Configuraciones (Configuration Management Plan) del proyecto Llavero Digital. Mediante el plan de CM se pretende organizar las responsabilidades del equipo de trabajo, controlar la configuración de los requerimientos, documentos, software y herramientas utilizadas en este proyecto.
El proyecto busca en principio generar un software que corra sobre la plataforma Java VM de manera local. El sistema en resumen podrá gestionar de forma segura y cómoda el uso de contraseñas e información sensible de diferentes usuarios del producto.
Luego a futuro debería correr de forma remota en un servidor web y ser versátil en diferentes dispositivos para el usuario final.

1.2. Propósito del Plan de Gestión de las Configuraciones

El plan de configuración busca establecer una jerarquía de responsabilidades dentro del equipo de trabajo, dichas responsabilidades están distribuidas en diferentes acciones necesarias para llevar a cabo el proyecto de forma ordenada. El plan de gestión de la configuración detalla cómo vamos a registrar, supervisar, controlar y auditar la configuración.

Todos los requisitos de configuración de un proyecto, incluidos los requisitos de funcionalidad, los requisitos de diseño y cualquier otra especificación, deben identificarse y registrarse. A medida que se modifica el alcance del proyecto, se debe evaluar, aprobar y documentar su impacto en la configuración. Realizar un seguimiento de la configuración de nuestro proyecto en todo momento, saber en qué versión se encuentra tu configuración y tener un historial de las versiones anteriores.

1.3. Herramientas de Administración de Configuraciones

Herramienta	Propósito
IntelliJ IDEA	IDE

GitHub	Control de versiones
Visual Paradigm	Diagramación UML
Jenkins	Integración Continua
selenium	Pruebas Automatizadas
Documentos de Google	Editor de texto
GitHub Desktop	Gestión de defectos

1.4. Roles y Responsabilidades

Administrador de configuraciones *:* Gaston Ojeda
Encargado de los test(tester): Franco Riba
Encargado de programar el código: Facundo Lorenzo
Administrador de proyecto: Patricio Roig
Diseñador: Facundo Lorenzo
Asegurador de calidad del programa: Franco Riba
Documentador: Gaston Ojeda
Encargado del release: Patricio Roig
Administrador de versiones: Facundo Lorenzo

2. Esquema de Directorios

2.1. Estructura de Directorios y Propósito de cada uno

La estructura de directorios del proyecto está totalmente contenida en un repositorio remoto de github.

- **/BrainStorm** diagramamos con imágenes los posibles proyectos a tomar.
- **/Documentación** se encuentra el documento del plan de gestión de las configuraciones, el documento de requerimientos, la matriz de trazabilidad y un documento que identifica los casos de uso y pruebas del sistema.
- **/mvnSistemaLlavero** es el directorio raíz de trabajo del proyecto
    - **/.idea** es un directorio que guarda las configuraciones del IDE Intellij IDEA
    - **/META-INF** es un directorio reconocido por la plataforma de Java para configurar aplicaciones.
    - **/src** contiene el source code del proyecto más el archivo de configuración pom.xml.

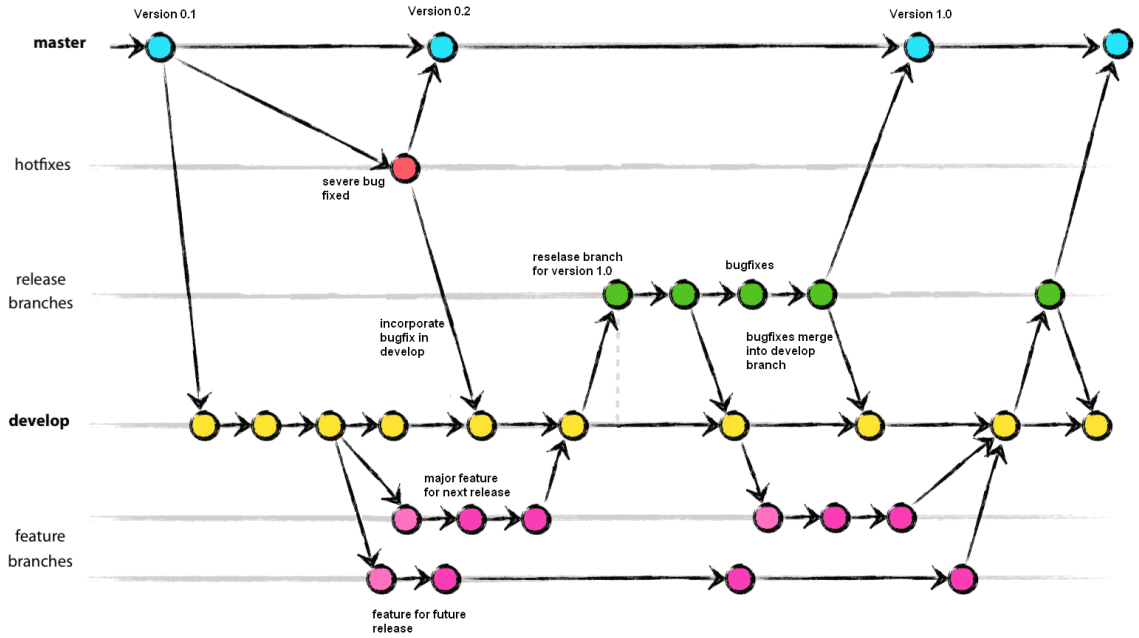
- **/main**
    - **/java** contiene el código escrito en lenguaje Java.
        - **/Entidad**: en este directorio se distinguen las entidades mediante sus atributos.
        - **/Interfaz**: en este directorio se guardan los archivos .form y .java que tiene como propósito mostrar interfaces gráficas (ventanas).
        - **/Lógica**: en este directorio se escriben clases con el código correspondiente al back end del proyecto.
        - **/Repository**: en este directorio se escriben clases con el código responsable de persistir la información del sistema.
    - **/resources:** en este directorio se encuentran ciertos archivos de configuración.
        - **/Images**: directorio para guardar imágenes a utilizar dentro del sistema.
- **/test**
    - **/java**
        - **/PruebasUnitarias:** en este directorio encontramos clases que implementan la librería de JUnit para hacer test unitarios referido al código fuente.
        - **/PruebasComplemento:** En este directorio encontramos clases destinadas a correr tests para chequear el compartimiento en conjunto de ciertas funcionalidades.
        - **/BigBang:** En este directorio encontramos código de test que busca correr el sistema en su totalidad.
- **/target:** En este directorio encontramos archivos de compilación y package relacionados con maven y también es donde generamos nuestro ejecutable .jar con sus respectivas dependencias.

2.2. Normas de Etiquetado y Nombramiento de Archivos

El nombramiento de archivos dentro del código fuente se define por el nombre de la clase más el tipo de clase que representa, por ejemplo dentro del directorio Entidades encontraríamos a la clase PersonaEntidad.java.
El nombramiento de directores se divide por responsabilidades a la hora de escribir código.

3. Gestión de la Configuración del Código

3.1. Esquema de Ramas



<u>El esquema de ramas cuenta con:</u>

- Una rama principal o master en la cual se encuentran las versiones estables del proyecto.
- A continuación de ésta se encuentra la rama de hotfixes la cual se emplea para solucionar errores muy urgentes.
- Una rama de releases en la cual se encuentran las versiones prontas a salir pero que requieren algún control de bugs final.
- La rama de desarrollo, es la rama clave donde el proyecto evoluciona entre versiones
- Finalmente se encuentran las ramas de features donde se implementan las nuevas características del proyecto, priorizando las características que se implementaran inmediatamente en el próximo release por sobre las que serán implementadas en releases futuros.

3.2. Política de Etiquetado de las Ramas

En nuestro esquema de ramas, las versiones finales, es decir las que pasan por la rama de release son etiquetadas como una versión nueva, por ejemplo 1.0, 2.0, 3.0 respectivamente. Por su parte las versiones que solo requirieron la solución de un bug, es decir que pasaron por la rama de hotfixes se etiquetan como versiones preliminares, por ejemplo 1.1, 1.2, 2.3.

3.3. Política de Fusión de Archivos

En éste esquema de ramas la fusión de archivos funciona de la siguiente manera:

- En primer lugar la rama master se fusiona con archivos provenientes tanto de la rama hotfixes (versiones preliminares) como de la rama de release (versiones finales).
- Si una versión presenta alguna modificación en la rama de hotfixes, sus archivos modificados deben fusionarse tanto con los archivos de la rama master como con los archivos de la rama develop, para que el desarrollo continúe con los archivos modificados.
- De igual manera los que se encuentran en la rama de release deben fusionarse tanto con los de la rama master como los de la rama develop para continuar el desarrollo de futuras versiones.
- Por su parte las modificaciones en la rama develop no se fusionan con ninguna rama sino que se envía la versión próxima a lanzar a la rama de release que se encargará de enviarla a la rama master. La rama develop también se encarga de crear las ramas de features para implementar nuevas características.
- En última instancia las características implementadas en la rama de features deben fusionarse con los archivos de la rama develop para poder continuar con el desarrollo.

4. Gestión de Cambios

4.1. Change Control Board (CCB)

4.1.1. Introducción y Objetivos

El grupo que conforma la CCB (Comité de Control de Cambios) tomará las decisiones relacionadas a la aprobación o el rechazo de los CR (Pedidos de

Cambio) realizados ya sea por clientes o por desarrolladores, se evaluarán para ello los potenciales impactos generados por dichos cambios respetando prioridades que se les ha asignado previamente a cada uno. El objetivo de la CCB es la toma de decisiones en tiempo y forma, siguiendo una agenda clara respecto a cambios en la documentación, código fuente, fechas de releases, mutación en los requerimientos, y cualquier otro elemento sobre el que se evalúa un posible cambio.

4.1.2. Miembros

Roll dentro de la CCB	**Titular/es**	**Suplente/es**	**Función**
General Coordinator (GC)	Roig, Patricio	Ojeda, Gastón	● Organización de Reuniones ● Supervisión de actividades ● Mediador ante discrepancias entre miembros de la CCB
Change Request Analyst (CRA)	Lorenzo, Facundo Riba, Franco	Ojeda, Gastón Roig, Patricio	● Análisis de factibilidad de los CR que le fueron asignados de forma previa a las sesiones ● Exposición ante la junta de los posibles riesgos, ventajas, consecuencias negativas y positivas de los CR discutidos.
Change Request Receiver (CRR)	Ojeda, Gastón	Riba, Franco	● Recepción de las planillas con los CR de clientes y desarrolladores. ● Clasificación del tipo de pedido de cambio. ● Asignación a los analistas según la clasificación asignada. ● Planificación de CR a discutir en cada sesión..

4.1.3. Frecuencia de Reunión de Trabajo

Motivo	**Frecuencia**	**Duración**

Reunión de reporte periódico y coordinación	Hasta 3 veces por semana, de forma virtual o presencial a convenir por los miembros.	Hasta 1 hora
Sesión de Agenda	2 veces por semana, de forma virtual o presencial a convenir por los miembros.	Hasta 2 horas

4.2. Proceso de Control de Cambios

Etapas del proceso:

1. Inicio de la solicitud de cambios: el cliente o desarrollador debe presentar ante el Change Request Receiver la planilla de solicitud, respetando el formato establecido y proporcionando todos los detalles solicitados en la misma.

2. El CRR revisa las planillas y las clasifica en función del área (desarrollo, marketing, control de versiones, etc) del proyecto a la cual se refiere la solicitud de cambio, luego informa y releva el CR a los analistas correspondientes a cargo de esa área.

3. Análisis de la solicitud de cambio: se determina por parte del analista asignado si el CR tiene razón de ser o no. Luego, se determinan las implementaciones, los impactos en el proyecto, tiempo, dinero, etc., que deberán ser expuestos antes los miembros de la CCB en las Sesiones de Agenda.
4. Resolución del análisis: se define el estado de la solicitud por parte del CCB:

a) Revisar: petición de mayor información sobre los cambios.
b) Rechazada: la petición no resulta factible a nivel técnico o por el contexto actual y el contexto previsto del proyecto.
c) Postergada: la petición es factible, el contexto actual del proyecto no permite su aceptación, pero en el futuro será aceptada potencialmente.
d) Aprobada: la petición resulta factible.
e) Removida: el solicitante desestima la idea de cambios.

5. Implementación de la solicitud de cambio: se lleva a cabo la organización temporal, de recursos y de implementación.

6. Verificación y cierre de la solicitud: la implementación de los cambios es corroborada y se cierra la petición.

7.

7.1. Herramienta de Control de Cambios

La herramienta de control de cambios a utilizar es GitHub Desktop.

8. Releases

8.1. Formato de Entrega de Releases
8.2. Formato de Entrega del Instalador
8.3. Instrucciones Mínimas de Instalación