SuiGeneris- Interno

IType

PLAN DE GESTIÓN DE CONFIGURACIONES

Autores:

- Cancinos, José
- Giraudo, Juan Pablo
- Oliva, Nahuel

Versión del documento: 1.0.0

HISTORIAL DE REVISIONES

Versión	Fecha	Resumen de cambios	Autores
1.0.0	2/5/2019	Documento inicial	Cancinos, José María Giraudo, Juan Pablo Oliva, Nahuel

REGISTRO DE APROBACIÓN

Lista de miembros del equipo cuya aprobación debe ser explicitada para todo cambio significativo en este plan:

Aclaración:

- Aprobaciones no son necesarias en caso de no ser significativas.
- Aprobaciones sobre el plan son solamente necesarias si el cambio es realizado por otra persona distinta al administrador de configuraciones

Gerente de configuración (CM)	Fecha	Firma
Giraudo, Juan Pablo	2/5/2019	

Subgerentes de manejo de la configuración	Fecha	Firma
Cancinos, José María	2/5/2019	
Oliva, Nahuel	2/5/2019	

ÍNDICE

1. INTRODUCCIÓN

1.1) PROPÓSITO Y ALCANCE DEL PLAN

El presente documento expone el Plan de Gestión de Configuraciones realizado para el proyecto “IType”. Mediante este plan se busca poner en conocimiento las políticas, estrategias, herramientas y métodos empleados para el manejo del software producido.

1.2) PROPÓSITO DEL PLAN DE GESTIÓN DE CONFIGURACIONES

- Definir políticas y procesos de la administración de configuración; y explicitar los fundamentos de las mismas.
- Establecer cuáles son los roles en el CCB.
- Definir herramientas de gestión de versiones y entrega.
- Exponer los principios para la construcción del sistema que garanticen la calidad requerida y pretendida.

1.3) HERRAMIENTAS PARA LA ADMINISTRACIÓN DE CONFIGURACIONES

Herramienta/Proceso	Propósito
Android Studio	Herramienta de desarrollo de aplicaciones para Android. IDE.
GitLab	Control de versiones. Integración continua
VisualParadigm	Herramienta gráfica para diagramas UML
GitLab CI/CD	Herramienta de integración continua y monitoreo de cada rama
JUnit	Herramienta de pruebas automatizadas
Gradle	Herramienta de construcción automática
Microsoft Word	Editor de texto para la realización de los informes
Issues de GitLab	Reporte de problemas

1.3.1) DIRECCIÓN Y FORMA DE ACCESO A LA HERRAMIENTA DE CONTROL DE VERSIONES

Se emplea la herramienta GitLab.

La dirección del mismo es: https://gitlab.com/Nadaol/typing_app

Se accede mediante petición de acceso al propietario del repositorio del proyecto ya que el mismo se ha declarado como “privado”. Es requerido poseer una cuenta en GitLab.

1.3.2) DIRECCIÓN Y FORMA DE ACCESO A LA HERRAMIENTA DE INTEGRACIÓN CONTINUA

Se emplea la herramienta integrada a GitLab de integración continua y entrega continua – GitLab CI/CD.

La dirección del mismo es el propio directorio que utilizaremos. Para mayor información, el enlace de GitLab CI/CD es: https://docs.gitlab.com/ee/ci/README.html

Se accede con una cuenta GitLab. Si bien es una funcionalidad de pago, ésta es brindada de forma gratuita para proyectos open source y estudiantiles.

1.3.3) DIRECCIÓN Y FORMA DE ACCESO A LA HERRAMIENTA DE GESTIÓN DE DEFECTOS

JUnit.

Se integrará junto a GitLab para el testing del código y la realización de pruebas automáticas.

Acceso mediante GitLab.

Mayor información en documentación de pruebas JUnit:
https://docs.gitlab.com/ee/ci/junit_test_reports.html

1.3.4) DIRECCIÓN Y FORMA DE ACCESO A LA HERRAMIENTA DE SCRIPT DE CONSTRUCCIÓN AUTOMATIZADA

Gradle.

Permite colocar un modelo del proyecto de software que se construirá. Incluir todos los módulos y componentes externos. Definir el orden de construcción de los elementos del código.

Cabe aclarar que Android Studio tiene integrado el funcionamiento de Gradle.

https://developer.android.com/studio/releases/gradle-plugin

1.3.5) DIRECCIÓN Y FORMA DE ACCESO A LA HERRAMIENTA DE REPORTE DE PROBLEMAS

Issues de GitLab.

Permite la creación y manejo de mensajes acerca de problemas o solicitudes por parte de un desarrollador hacia el resto del equipo. O tareas a asignar. Es un modo transparente de establecer comunicación entre el equipo.

https://gitlab.com/Nadaol/typing_app/issues

2) Esquema de directorios y propósito de cada uno.

2.1) ESQUEMA DE DIRECTORIOS

README	./readme
Contiene el archivo “README.md”. Documento que contiene la información esencial del proyecto (nombre, descripción, autores, licencias, etc)	

LICENCIAS	./licencias
Contiene las declaraciones completas sobre licencias y derechos de copyright sobre el proyecto.	

VERSION	./version
Contiene cada paquete final de las versiones obtenidas del software	

LANZAMIENTOS (RELEASE)	./release
Contiene paquetes, documentos de lanzamiento y archivos binarios	

REQUERIMIENTOS	./requerimientos
Archivos con declaraciones de requerimientos especificados para el proyecto.	

DOCUMENTACIÓN	./docs
Documentación propia del proyecto. Informes internos. Recursos externos.	

PRUEBAS	./pruebas
Suite o conjunto de pruebas unitarias	

CÓDIGO FUENTE	./src
Contiene el código fuente del proyecto. Se divide en subdirectorios según paquetes	

ARCHIVOS	./archivos

Archivos varios del proyecto, como archivos de configuración de herramientas, comandos, archivos de texto, etc.

DISEÑO	./diseno
Archivos referidos al diseño del sistema	

HERRAMIENTAS	./herramientas
Archivos varios	

BASE DE DATOS	./basedatos
Archivos sobre las bases de datos del sistema y documentación asociada a ellas.	

3) Normas de etiquetado y de nombramiento de los archivos.

3.1) NORMAS DE NOMBRAMIENTO

- Uso de guiones (-) para separar etiquetas del nombre.
- Uso de puntos (.) para separar palabras o números.
- Nunca usar espacios o guiones bajos.
- No usar caracteres especiales.
- Usar letras minúsculas.
- Uso de "versionado de software" en la etiqueta(SemVer - semantic versioning, en inglés) para indicar el nivel de versión del archivo[1]

3.2) SECCIONES DEL NOMBRE:

1. Identificador del proyecto: "itype"
2. Nombre del archivo
3. Estado del archivo:
   - Final: final
   - Estable: stable
   - Inestable: un
4. Extensión del archivo

Ejemplo: itype-plan.gestion.configuraciones-final.pdf

3.3) NORMAS DE ETIQUETADO:

Se utilizarán las etiquetas (tags, en inglés) para identificar versiones de un archivo dentro del repositorio.

- Numero de versión: "vX.Y.Z"
  - Posición 1: indica cambios de gran importancia. Cambios en la API.
  - Posición 2: indica cambios menores. Nuevas funcionalidades compatibles con la presente versión mayor.
  - Posición 3: parches al software. Solución a bugs.

  Ejemplo: v3.1.0

- Fecha de creación
  - Formato: AAAA-MM-DD

  Ejemplo: 2019-05-02

3.4) EJEMPLO FINAL DE FORMATO

Un ejemplo con tanto nombre y etiqueta según las políticas adoptadas será:

Nombre: plan.gestion.configuraciones-final.pdf

Etiquetas: v1.0.0; 2019-05-02

[1] Versionado de Software – Wikipedia. https://es.wikipedia.org/wiki/Versionado_de_software

4) Plan del esquema de ramas a usar.

4.1) ESQUEMA

Existe una rama principal (master), una rama de desarrollo (develop) y luego, cada característica a desarrollar tiene su propia rama y finalmente, ramas para el desarrollo de cada función asignada al programador.

Se realiza para no introducir errores durante el desarrollo y continuar sin afectar a las versiones de rama principal.

Cada nueva característica o funcionalidad (feature) implicará una nueva rama.

Por lo tanto, esta organización permite trabajar al mismo tiempo en funcionalidades distintas.

- Rama principal - *master branch*: aquí estarán las versiones entregables de nuestra aplicación o releases.

    - El acceso es restricto.
    - Los cambios sólo pueden ser insertados con peticiones “pull”. No commits directos son permitidos.
    - Siempre tiene que tener la última versión de release.

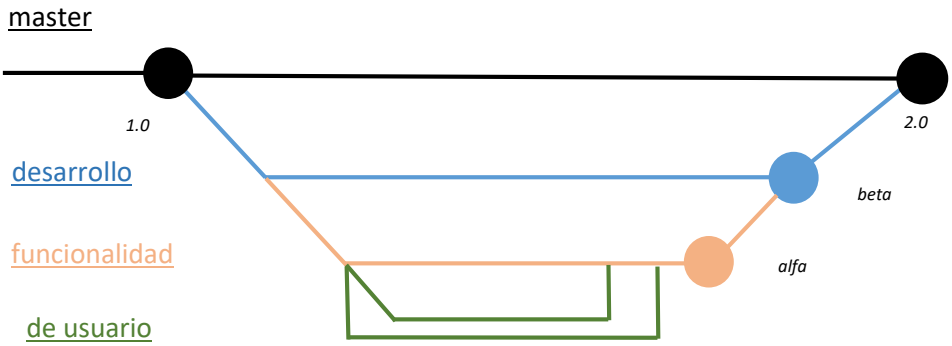
- Rama de desarrollo – *develop branch*: rama en donde se alojan las releases alfa o beta. Toma los “commits” de la rama principal.

    - El acceso es restricto.
    - Cambios puede ser insertados por peticiones “pull” o commits directos (en casos determinados).
    - Tiene siempre las últimas versiones alfa o beta.
    - No puede contener código no release.

- Rama de funcionalidad – *feature branch*: es la rama principal para una función en desarrollo. Cuando se trabaja en una nueva funcionalidad, se crea una “rama de desarrollo de función” en base a la rama de desarrollo.
    - El acceso es irrestricto.
    - “Commits directos” se permiten para cambios menores o si una única persona está trabajando en dicha rama.
    - También uso de “pull”
    - Al completar el desarrollo de una funcionalidad, debe hacer merge con la rama de desarrollo. No nuevos commits después de finalizar la función.

- Ramas privadas o de usuario: ramas creadas para cada desarrollador.

    - Etiquetado: “desarrollo_(funcionalidad)_(nombredesarrollador)”
    - Commits directos hacia una rama ajena no se permiten.
    - “Pull” a ramas ajenas deben ser aprobadas por el dueño de la rama.
    - Merge a “rama de funcionalidad” al terminar el trabajo.



4.2) POLÍTICAS DE NOMBRADO DE LAS RAMAS

Rama principal: "master"

Rama de desarrollo: "desarrollo"

Rama de funcionalidad:

Formato: *funcion-NombreFuncion-vX.Y*

- Palabra "funcion": identifica que es una rama de funcionalidad
- NombreFuncion: nombre que indica cuál es la función en desarrollo
- VX.Y: indica la versión:
  - X: según el índice del full release en desarrollo (rama principal)
  - Y: según el índice de la beta en desarrollo (rama desarrollo)

Ejemplo: funcion-sensorco2-v1.4

Rama de usuario:

Formato: *usuario-NombreDesarrollador-NombreFuncion-vX.Y*

- Palabra "usario": identifica que es una rama de usuario
- NombreDesarrollador: indica el sujeto que realiza el trabajo sobre esa rama.
- Idem anteriores

Ejemplo: usuario-juanrem-funcion-sensorco2-v1.4

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

5.1) POLÍTICA DE FUSIÓN DE ARCHIVOS

1. Sólo pull desde la rama de desarrollo hacia la rama principal con la versión beta final.
2. Sólo pull desde la rama de funcionalidad hacia la rama de desarrollo con la versión alfa final.
3. Commit directos o pull desde ramas de usuario de desarrollo hacia rama de funcionalidad.

5.2) ETIQUETADO DE ENTREGAS

Se respeta las políticas definidas anteriormente para el etiquetado de archivos.

El formato de la etiqueta es en base a SemVer (semantic versioning, en inglés) sumado con la palabra “release”.

Numero de versión: “vX.Y.Z”

- Posición 1: indica cambios de gran importancia. Cambios en la API.
- Posición 2: indica cambios menores. Nuevas funcionalidades compatibles con la presente versión mayor.
- Posición 3: parches al software. Solución a bugs.

Identificador de entrega:

- release
- beta
- alfa

Ejemplo:

#v1.0-release

#v1.4-beta

#v1.3.8-alfa

6) Forma de entrega de los “releases”, instrucciones mínimas de instalación y formato de entrega.

Los “releases” o lanzamientos estarán presente en la rama principal (master branch).

6.1) FORMATO DE ENTREGA DE RELEASES:

Cada release se entrega en un archivo .zip exportado del proyecto de Android Studio.

Este archivo debe seguir las políticas de nombre y etiquetado expuestas anteriormente.

Ejemplo:

Nombre: itype-release-final.zip

Etiquetas: v3.0.0; 2019-5-5; release

6.2) FORMATO DE ENTREGA DEL INSTALADOR

El archivo de instalación será un archivo de tipo. apk (Android Application Package). Es un formato empleado por Android para la distribución e instalación de aplicaciones.

El APK se obtiene una vez compilado del programa y sus módulos ensamblados en un solo archivo.

Un archivo APK contiene:

- META-INF: certificados de aplicación, licencia, etc.
- Lib: código compilado para cada uno de los procesadores destinados.
- Res: recursos no compilados. Layouts, imágenes, strings, etc.
- Assets: estructuras de archivos, etc.
- AndroidManifest.xlm: contiene información de la aplicación que debe conocer el sistema Android para poder ejecutar el código de la app. Determina permisos, bibliotecas, nombre de paquetes, describe componentes, etc.
- Classes.dex: clases compiladas en el formato .dex usada por la máquina virtual de Android y Android Runtime.
- Resources: recursos precompilados como XML

6.3) INSTRUCCIONES MÍNIMAS DE INSTALACIÓN:

1) Obtener el archivo final .apk.
2) En un dispositivo Android, copiar el archivo.
3) Permitir “instalaciones de fuentes desconocidas”.
   Dirigirse a Ajustes/Seguridad.
4) Abrir el archivo. apk y proseguir con la instalación.
5) Abrir el programa desde el menú de aplicaciones.

7) Change Control Board. Se debe incluir el propósito, la lista y forma de los integrantes del equipo y su rol en la CCB, la periodicidad de las reuniones, etcétera.

7.1) INTRODUCCIÓN Y OBJETIVOS

El Change Control Board (Comité de Control de Cambios, en español) es el grupo encargado de evaluar, aprobar o rechazar los pedidos de cambios realizados por diversos agentes internos o externos (clientes).

El comité busca que todo cambio sea considerado por todas las partes y lograr su total autorización antes de su implementación en los planes, documentos y el código de la aplicación.

Se monitorea y controla las peticiones de cambio para establecer luego las bases de los requisitos de configuración.

Las decisiones deben realizarse en base de actividades que permitan asegurar la calidad de producto y el buen avance del objetivo en cada ciclo de prueba.

El CCB debe cumplir con reuniones de trabajo de forma periódica en dónde se definen las políticas de trabajo.

7.2) MIEMBROS[2]:

Posición dentro del CCB	Titular	Suplente	Función
Director del CCB	Oliva, Nahuel	Cancinos, José María	• Aprobación o modificación de cambios. • Organizador de reuniones. • Manejo del CCB. • Moderador.
Gerente de manejo de configuraciones (GMC)	Giraudo, Juan Pablo	Cancinos, José María	• Responsable en general de tareas de planeamiento, diseño de las políticas de control de seguimiento y su actualización. • Veedor de cumplimiento de lo anterior
Gerente de ingeniería o de release (GI)	Cancinos, José María	Oliva, Nahuel	• Evaluación del costo en cuanto a cambios de infraestructura del sistema, fechas de entrega, etc.

[2] "Software Configuration Management Implementation Roadmap" – Mario E. Moreira. 2004

Gerente de coordinación (GI)	Giraudo, Juan Pablo	Oliva, Nahuel.	• Se encarga de las peticiones de cambio mediante change request forms y logs. • Documenta y traduce a modo formal los cambios.
Gerente de pruebas	Oliva, Nahuel	Cancinos, José María	• Aplica el plan de requerimientos para contrastar con lo obtenido. • Organizar las pruebas • Definir las actividades a implementar como estrategia de prueba (test) • Evalúa impactos del cambio en la calidad de la entrega y el calendario de testing

7.3) FRECUENCIA DE REUNIÓN DE TRABAJO

MOTIVO	FRECUENCIA
Reunión periódica del comité	Lunes y Jueves (en forma presencial)
Contacto y comunicación	Lunes a Sábado mediante contacto por medios personales

7.4) PROCESO DE CONTROL DE CAMBIOS

Este proceso consta de las siguientes etapas:

1) Inicio de la solicitud de cambios: una planilla de solicitud de cambios es formalmente presentada y registrada.
2) Análisis de la solicitud de cambio: se determina si tiene razón de ser o no. Luego, se determina las implementaciones, los impactos en el proyecto, tiempo, dinero, etc.
3) 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
   c) Postergada
   d) Aprobada
   e) **Removida**: el solicitante desestima la idea de cambios.
4) Implementación de la solicitud de cambio: se lleva a cabo la organización temporal, de recursos y de implementación.
5) Verificación y cierre de la solicitud: la implementación de los cambios es corroborada y se cierra la petición.

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

8.1) ESPECIFICACIÓN DE HERRAMIENTA DE SEGUIMIENTO DE DEFECTOS

Utilizaremos como herramienta de seguimiento y reporte de defectos a la propia funcionalidad de GitLab definida como “Issues”.

Se puede acceder desde el siguiente enlace: https://gitlab.com/Nadaol/typing_app/issue

Es una funcionalidad de seguimiento de problemas del proyecto. Acá se reporta la existencia de errores en el código/archivo.

Pueden ser confidenciales para ciertos integrantes o visibles para todos los miembros del grupo de trabajo.

Es personalizable en sus campos y etiquetas.

- Contenido:
  - Título
  - Descripción del problema y tareas a realizar
  - Comentarios
- Personas:
  - Autor del reporte de problema
  - Miembros asignados para resolver el problema
  - Miembros que pueden ver este reporte
- Estado del problema:
  - Estado: abierto/cerrado
  - Estado de las tareas.
- Planeamiento y seguimiento
  - Objetivo a cumplir
  - Fecha a cumplir
  - Peso del problema
  - Seguimiento del tiempo
  - Etiquetas

8.2) ESTADOS DEL PROBLEMA

GitLab permite modificar el campo “Estado” en sólo dos opciones: “abierto” y “cerrado”.

Por ello, utilizaremos etiquetas en los reportes de problemas para establecer mayor especificación del mismo.

- Pregunta: implica una pregunta sobre determinada parte del código o funcionalidad
- Petición: indica un pedido de nuevas características o funcionalidades.
- Bug: problemas de tamaño pequeño con rápida solución.
- Ayuda: el autor del reporte solicita mayor información sobre un problema.
- Duplicado: expone problemas parecidos a otros realizados anteriormente.

Importante: se reporta un problema que “rompe” el funcionamiento del sistema.

- Solicitud: se informa sobre la necesidad de un archivo para poder continuar con el trabajo.

Cabe aclarar que los reportes de problemas no se eliminarán. Solamente se cerrarán y almacenarán en la pestaña “Closed”. De esta forma, cualquier integrante podrá conocer el historial de reportes y obtener información útil para resolver un problema nuevo.

9) Cualquier otra información relevante.

Se adjunta una plantilla para toda solicitud de cambios