Blue Screen- interno

Garçon

PLAN DE GESTIÓN DE CONFIGURACIONES

Autores:

* Sebastian Emmanuel Mazzeo
* Fabrizio Andres Recchini
* Giraudo Juan Pablo
* Ortega, Cristian
* Oller, Carlos

Fecha:

2/4/2021

Versión del documento: 1.0.0

**Historial de Cambios**

|  |  |  |  |
| --- | --- | --- | --- |
| Versión | Fecha | Resumen de cambios | Autores |
| 1.0.0 | 2/4/2021 | Documento inicial | Mazzeo, Sebastian Emmanuel  Recchini, Fabrizio Andres    Giraudo, Juan Pablo  Ortega, Cristian  Oyola Oller, Carlos Daniel |

**Acrónimos/Glosario**

1. **Introducción**

1.1) **Propósito y alcance del plan**

1.2) **Propósito del plan de gestión de configuraciones**

1.3) **Herramientas para la administración de configuraciones**

|  |  |
| --- | --- |
| Herramienta/Proceso | Propósito |
|  | Herramienta de desarrollo.  IDE. |
| GitHub | Control de versiones.  Integración continua |
| VisualParadigm  StarUML | Herramienta gráfica para diagramas UML |
| JUnit | Herramienta de pruebas automatizadas |
| Gradle | Herramienta de construcción automática |
| Google Docs | Editor de texto para la realización de los informes |
| Issues de Github | Reporte de problema |

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

Se emplea la herramienta Github.

La dirección del mismo es://https

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 Github.

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

1.3.3) **DIRECCIÓN Y FORMA DE ACCESO A LA HERRAMIENTA DE PRUEBAS AUTOMÁTICAS**

Java -> JUnit.

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

**java ->** Gradle.

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.

# **2) Esquema de directorios y propósito de cada uno.**

2.1) **ESQUEMA DE DIRECTORIOS**

|  |  |
| --- | --- |
|  |  |
|  | |

# **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]](#footnote-0)
  1. **SECCIONES DEL NOMBRE:**

1. Identificador del proyecto: “Garço”
2. Nombre del archivo
3. Estado del archivo:

* Final: final
* Estable: stable
* Inestable: un

1. Extensión del archivo

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

* **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

* 1. **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

# **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.



* 1. **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 “usuario”: 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:**

# **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]](#footnote-1)**:**

|  |  |  |  |
| --- | --- | --- | --- |
| Posición dentro del CCB | Titular | Suplente | Función |
| Director del CCB | Giraudo,  Juan Pablo | Recchini, Fabrizio Andrés | * Aprobación o modificación de cambios. * Organizador de reuniones. * Manejo del CCB. * Moderador. |
| Gerente de manejo de configuraciones  (GMC) | Cristian,  Ortega |  | * 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) |  |  | * Evaluación del costo en cuanto a cambios de infraestructura del sistema, fechas de entrega, etc. |
| Gerente de coordinación  (GI) |  |  | * Se encarga de las peticiones de cambio mediante change request forms y logs. * Documenta y traduce a modo formal los cambios. |
| Gerente de pruebas | Mazzeo,  Sebastian | Giraudo,  Juan Pablo | * 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 Viernes a las 17:00 hs |
| Contacto y comunicación | De Lunes a Viernes por el grupo de whatsapp |

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:
4. **Revisar**: petición de mayor información sobre los cambios.
5. **Rechazada**
6. **Postergada**
7. **Aprobada**
8. **Removida**: el solicitante desestima la idea de cambios.
9. Implementación de la solicitud de cambio: se lleva a cabo la organización temporal, de recursos y de implementación.
10. 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 GitHub definida como “Issues”.

Se puede acceder desde el siguiente enlace: <https://github.com/gar/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.**

1. Versionado de Software – Wikipedia. https://es.wikipedia.org/wiki/Versionado\_de\_software [↑](#footnote-ref-0)
2. “Software Configuration Management Implementation Roadmap” – Mario E. Moreira. 2004 [↑](#footnote-ref-1)