-nameNotFound- Team

"JavaChess"

PLAN DE GESTIÓN DE LAS CONFIGURACIONES

Autor(es)
Cáceres, Juan Manuel
Hidalgo, Fabian Nicolás
Mamaní, Franco Ivan
Mendez, Jorge Nicolas

Versión de documento
1.0.0

Tabla de contenidos

1 Historial de Revisiones

Versión	Fecha	Observación	Autor(es)
1.0.0	20/04/2022	Versión base del documento	Cáceres, Juan Manuel Hidalgo, Fabian Nicolás Mamani, Franco Ivan Mendez Jorge Nicolas
1.0.1	18/06/2022	Correcciones en base a Feedback	Cáceres, Juan Manuel Hidalgo, Fabian Nicolás

2 Página de Aprobaciones

A continuación se encuentra la información de las personas cuya aprobación es necesaria para realizar cambios significativos al siguiente plan.

* Aprobación sólo será necesaria en caso de cambios mayores, no será necesaria para cambios menores.
* Aprobación sólo será necesaria en caso de que el cambio sea realizado por personas que no son Change Managers.

Configuration Manager	Fecha	Firma
Hidalgo, Fabian Nicolás	20/04/2022	

Configuration Manager BackUp	Fecha	Firma
Cáceres, Juan Manuel	20/04/2022	

Build Manager	Fecha	Firma
Méndez, Jorge Nicolás	20/04/2022	

Release Manager - Issue Coordinator	Fecha	Firma
Mamaní, Franco Iván	20/04/2022	

3 Introducción

3.1 Propósito y Alcance

Este documento cubre el plan de gestión de configuración para “JavaChess”. La intención de este plan es controlar la configuración de los requerimientos, documentos, hardware, software, y herramientas usadas para este proyecto.
Administración de las Configuraciones de Software es el proceso por el cual se identifican los métodos y herramientas para controlar el software a lo largo de su desarrollo y uso.

3.2 Propósito de las Prácticas de Administración de las Configuraciones de Software

- ➢ Asegurar consistencia a la hora de practicar actividades de administración de las configuraciones de software.
- ➢ Definir los organismos autorizados para soportar las prácticas de SCM.
- ➢ Mantener la integridad del producto durante el ciclo de vida del mismo.
- ➢ Informar a grupos e individuos pertinentes del estado del proyecto.
- ➢ Crear un historial verificable de estados actuales y pasados de productos de trabajo.
- ➢ Mejoramiento del proceso.

3.3 Referencias / Acronimos / Glosario

Referencia	Descripción	Link/Dirección
JavaChess_Kanban	JavaChess Kanban Board Link	https://trello.com/b/dCF3rzqK/javachess
JavaChess_GitHub	JavaChess Repositorio Remoto Link	https://github.com/Nick07242000/JavaChess
JavaChess_GitHub_Actions	JavaChess CI y CD Link	https://github.com/Nick07242000/JavaChess/actions/
JavaChess_GitHub_Release	JavaChess Release History	https://github.com/Nick07242000/JavaChess/releases/
JavaChess_GitHub_Issues	JavaChess Issue Tracking List	https://github.com/Nick07242000/JavaChess/issues

Tabla 1 - Referencias

Acrónimo	Descripción
SCM	Software Configuration Management - Administración de las Configuraciones de Software
PCM/CM	Project Configuration Manager - Administrador de Configuración de Proyecto
CI	Continuous Integration - Integración Continua
CD	Continuous Deployment - Implementación Continua
Tag/Etiqueta	Identificadores para un ítem de configuración
CR	Change Request - Solicitud de Cambio

Tabla 2 - Acrónimos/Glosario

3.4 Herramientas de Administración de las Configuraciones

Herramienta	Propósito	Controles de Herramienta
Github	Herramienta de administración de las configuraciones utilizada para el versionado y control de código fuente …	Capacidades de branching de código fuente y merging. Capacidades de desarrollo paralelo.
Github Actions	Herramienta de CI y CD utilizada para automatizar la generación de builds, testeo y releases del proyecto …	Builds generadas y testeadas automáticamente en cambios a las ramas.
Github Issues	Herramienta de administración de cambios y requerimientos …	Control y administración de todos los requerimientos de cambios/mejoras
Maven	Herramienta de administración de software de proyecto y herramienta de comprensión. Permite la administración de builds, documentación, testeo …	Manejo de dependencias, build, testeo …
diagrams.net	Diagramas de diseño y arquitectura …	N/A
Trello	Administración de flujo de trabajo para equipos ágiles …	Asignación de tareas, seguimiento de progreso, estado de tareas.

Tabla 3 - Herramientas de Administración de las Configuraciones

4 Roles de Administración de las Configuraciones

4.1 Administradores de Configuración del Proyecto

Las actividades de administración de configuración serán coordinadas dentro del proyecto JavaChess por el Project Configuration Manager (PCM) rol el cual será asignado a una persona. Adicionalmente se asignará un PCM de respaldo.
El PCM será responsable de actividades tales como realizar el seguimiento de las ramas main, release, y feature, y determinar cuando las ramas son creadas, que actividades se realizarán en las mismas, etc
Las actividades, procesos, procedimientos, y políticas de administración de las configuraciones deben ser seguidas por todos los miembros del equipo. Es la responsabilidad de cada persona seguir y aplicar el proceso de administración de la configuración apropiada de acuerdo a sus roles asignados.

Rol	Principal	Respaldo
PCM	Hidalgo, Fabian Nicolás	Cáceres, Juan Manuel

Tabla 4 - Administradores de Configuración

4.2 Responsabilidades de Administración de las Configuraciones

Rol	Responsabilidades
PCM	Posee la responsabilidad de todos los ítems de configuración. Es responsable de la creación de todas las ramas y la administración de sus políticas. Es responsable de la aplicación de labels en ramas main y release. Asegura la integridad del producto y la trazabilidad de los ítems de configuración para todo el proyecto. Coordina actividades de CM dentro del proyecto. Asegura la correcta ejecución del esquema de CM. Asiste en las actividades de merge en las ramas main y release. Actividades de build en las ramas main y release. Analiza todos los hallazgos relacionados a CM.
Team	Ayuda a resolver conflictos durante las actividades de merge. Asegurar que los criterios de calidad de los entregables a la rama main sean cumplidos. Sigue todos los procesos, políticas y prácticas asociadas definidas para sus roles asignados.

Tabla 5 - Responsabilidades CM

5 Administración del Cambio

5.1 Alcance

La administración del cambio es el proceso que ocurre después de identificar y aprobar la baseline de la documentación, código fuente o productos de hardware. Cambios incluyen cambios internos al enfoque original documentado debido a resultados de tests o simulaciones, o pedidos externos de cambios a características o funcionalidades.

5.2 Comité de Control de Cambios

El propósito de este comité es asegurar que todos los cambios sean analizados y autorizados antes de su implementación. Es responsabilidad del comité aprobar o rechazar los pedidos de cambio realizados en la herramienta de gestión de cambios (JavaChess_GitHub_Issues).
El alcance de su trabajo será el de aprobar o rechazar cambios con respecto a los planes, documentos o código del proyecto.

5.3 Miembros Comité

A continuación se encuentran los miembros del comité que tienen poder de decisión sobre los cambios a realizar.

Rol	Nombre
Configuration Manager	Hidalgo, Fabian Nicolás
Configuration Manager BackUp	Cáceres, Juan Manuel
Build Manager	Méndez, Jorge Nicolás
Release Manager - Issue Coordinator	Mamaní, Franco Iván

Tabla 6 - Miembros Comité

5.4 Frecuencia

Reunión	Frecuencia
JavaChess Comité Control de Cambios	Basada en demanda

Tabla 7 - Frecuencia Reuniones Comité

5.5 Herramientas

La herramienta a utilizar será la sección Issues (JavaChess_GitHub_Issues) del repositorio remoto. Además, todo cambio aprobado se agregara al board kanban del proyecto.

5.6 Mapa del Proceso de Control de Cambio.

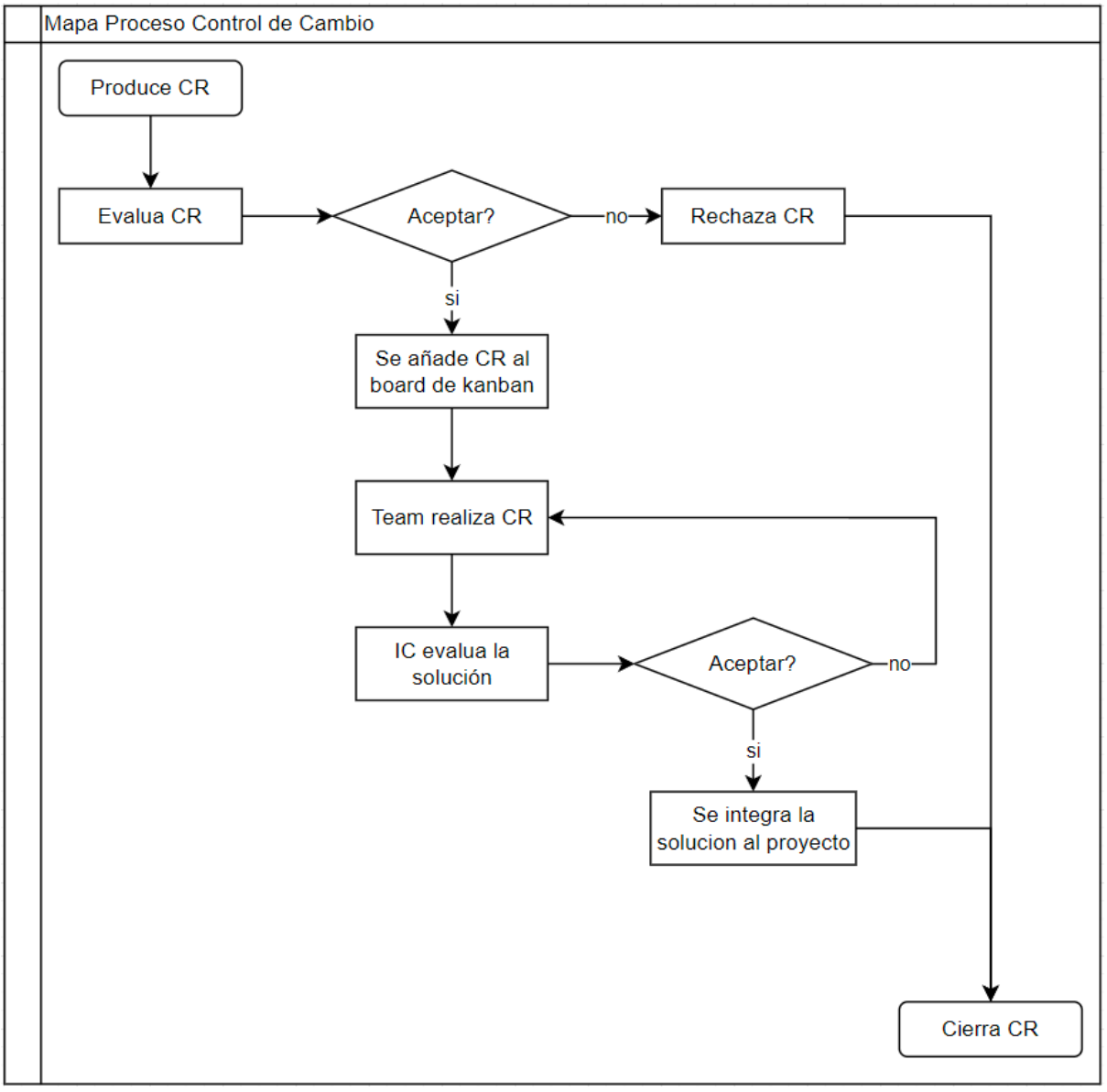

Gráfico 1 - Mapa Proceso Control del Cambio

El IC es el encargado de convocar la reunión del comité y presentar los CR, y agregar aquellos que hayan sido aprobados al board de Kanban.
Si el cambio es menor, el IC puede asignarlo al Kanban sin la aprobación del comité.

6 Identificación de Configuración

GitHub será usado para almacenar y compartir los componentes principales de distintos releases (JavaChess_GitHub).

Documentación general será almacenada dentro del archivo README dentro del proyecto, el cual también puede ser visualizado desde la página principal del proyecto en GitHub.

Documentación específica con respecto al release será almacenada en el cuerpo del mismo, el cual se declara a la hora de generar el release.

Por cada release generado se creará una nueva release en la sección de releases del repositorio de GitHub, donde se incluirá la documentación general, la documentación específica del release, el ejecutable de la versión, y el código fuente de la versión.

Las releases serán identificadas con el formato de versionado semántico 2.0.0, usando un número de versión MAYOR.MENOR.PARCHE, donde mayor representa cambios incompatibles en el proyecto, menor representa la adición de una funcionalidad compatible con versiones anteriores, y parche representa reparar errores compatibles con versiones anteriores.

La sección releases del repositorio funcionará de historial de versiones donde será accesible el código fuente de la aplicación al momento del release.

7 Equipos

El alcance de este documento cubre el código fuente administrado por el equipo de desarrollo de JavaChess.

Debido al número limitado de miembros del equipo, en lugar de conformarse diversos equipos que se encarguen de distintas áreas, utilizamos un enfoque DevOps donde las distintas áreas trabajan conjuntamente. En nuestro caso nuestro equipo se encarga de todas las áreas.

El equipo está a cargo de desarrollar nuevas funcionalidades utilizando metodologías ágiles. Deberá registrar qué tarea está realizando en el kanban (JavaChess_Kanban) e ir declarando su progreso en la misma. Realizarán commits del código fuente sobre ramas de la funcionalidad que están realizando. Una vez completada la funcionalidad, se realizará un merge con la rama principal.

Cada push realizado al repositorio remoto será acompañada por tests automáticos que deberán ser diseñados por el equipo. En caso de fallos de los mismos o del descubrimiento de bugs el equipo será responsable de realizar las correcciones apropiadas al código fuente.

La corrección de bugs sobre un release se realizará sobre la rama de release, realizando una posterior comprobación de la existencia de los bugs en el estado actual de la rama main y corrigiendo de ser necesario.

El equipo será responsable de acompañar cada commit y push al repositorio remoto con la información adecuada para su fácil identificación.

8 Administración de la Configuración del Código Fuente

En esta sección se describen distintos ítems de la administración del código fuente, como el esquema de ramas, tags a utilizar, estrategia de merge, y niveles de calidad esperados para el producto.

8.1 Esquema de Branches

El siguiente gráfico muestra el esquema de branches a utilizar, el cual parte del esquema GitHub Flow:

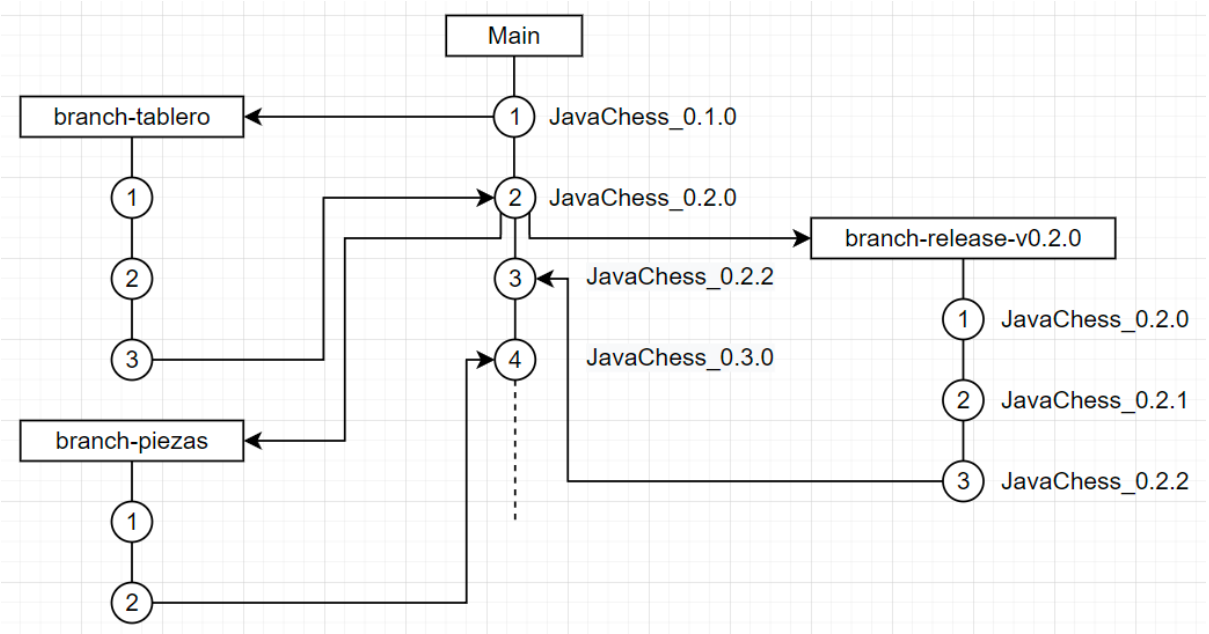

Gráfico 2 - Esquema de Ramas

8.1.1 Definición de Ramas

Las ramas a utilizar durante el proyecto son las siguientes:

- Rama de Integración: La rama Main es definida como la rama principal de integración donde todas las funcionalidades terminadas serán almacenadas. Es la rama creada por defecto por GitHub.

- Ramas de Desarrollo/Funcionalidad: Estas son las ramas donde se desarrollarán nuevas funcionalidades para posteriormente ser integradas a la rama de integración mediante un merge.

Se creará una rama por funcionalidad/componente de la aplicación a desarrollar. Esto es para asegurar estabilidad a la hora de desarrollar una funcionalidad y prevenir la creación de conflictos.
El desarrollo de una funcionalidad hasta su completitud en una rama separada de la rama de integración asegura la estabilidad de la rama principal …
El formato del nombre para estas ramas será branch-<funcionalidad/componente>.

- Ramas de Release: Estas ramas serán creadas por el CM con el propósito de establecer la versión del código fuente a utilizar para un release específico, las cuales siempre partirán de Main.
  El formato del nombre para estas ramas será branch-release-<version>.
  En estas ramas se realizará la corrección de bugs para el release específico y se almacenarán en la misma rama releases posteriores para aplicar estas correcciones.

- Ramas de Hotfix: Estas ramas seran creadas con el propósito de realizar la corrección de bugs detectados en código productivo. Su origen será la rama donde se encuentra la versión de release a corregir y utilizaran la nomenclatura branch-release-<version>-hotfix-<bug>

La idea principal detrás de este esquema de ramas es aislar código fuente inestable en ramas de desarrollo de manera de no corromper la estabilidad y el correcto funcionamiento de la versión funcional del proyecto, y de crear un historial del código fuente de los distintos releases del proyecto.

8.1.2 Definición de Etiquetas

Las etiquetas a utilizar durante el transcurso del proyecto son:

- Root Tags: Cuando una rama es creada, se incluirá una etiqueta en la versión del código fuente donde se originó con el formato Root-<nombre_rama>. Será responsabilidad del CM aplicar esta etiqueta a la hora de la creación de la rama.

- Merge Tags: Cuando se realiza un merge de una rama hacia Main, será responsabilidad del CM agregar esta etiqueta luego de haberse producido un merge por un miembro del equipo. Se utilizará el formato Merge-from-<nombre_rama>

8.1.3 Estrategia de Merge

A la hora de encontrarse lista una funcionalidad desarrollada en una rama, se aplicará un merge hacia la rama main para incorporar la misma al proyecto. Los merge serán realizados por el equipo asistidos por el CM, quien será responsable de aplicar la etiqueta Merge a la versión del código en la rama main luego de producido el merge.

Merges pueden provenir de una rama de funcionalidad o de una rama de release con el propósito de reparar bugs. Los merges provenientes de una rama de release deben ser realizados antes que los merge de funcionalidad siempre que sea posible, con el objetivo de

evitar conflictos de merge o corregir errores antes de agregar nuevos componentes al código fuente que pueden ser afectados por el error.

Para la realización de un merge se utilizaran Pull Requests, los cuales serán realizados por el desarrollador que quiere realizar el merge. Para poder integrarse al código de la rama principal el código de la rama a mergear deberá pasar las pruebas de CI del pipeline integrado por GitHub Actions, y además el Pull Request deberá ser aprobado por al menos un miembro del equipo que no sea el que inicio el PR.

El siguiente gráfico ilustra la estrategia de merge a utilizar durante el transcurso del proyecto.

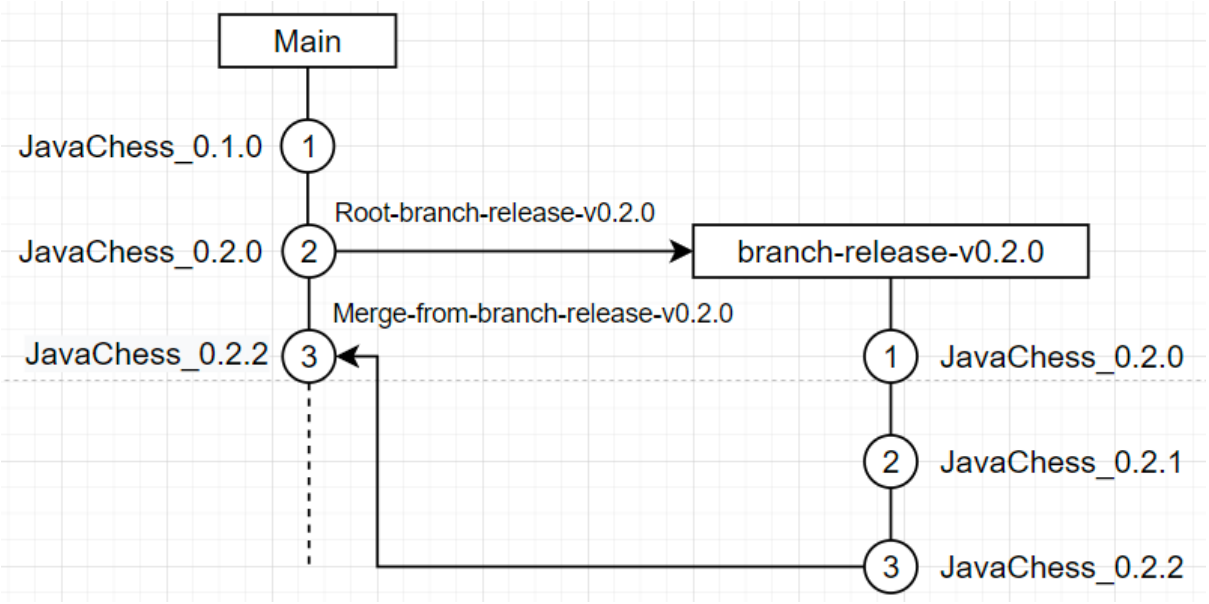

Gráfico 3 - Estrategia de Merge

8.2 Niveles de Calidad

Para las ramas de desarrollo se espera un bajo nivel de calidad, donde hasta el final de la una rama de desarrollo el proyecto es inestable con funcionalidades incompletas.

Para la rama principal Main, se espera estabilidad en la build con funcionalidades completas, con potencial para release. Se esperan funcionalidades testeadas y comprobadas, y un proyecto compilable sin problemas.

Para las ramas release, se espera una aplicación con todas sus funcionalidades comprobadas con un margen de al menos 90%. El proyecto debe ser estable y compilable, y haber sido testeado en la mayor medida posible.

El reporte de Bugs será realizado mediante el apartado de GitHub Issues utilizando la etiqueta pertinente. Estos podrán ser realizados por el equipo de desarrollo o usuarios de la aplicación. Deberá identificarse la versión del código donde se encontró el error en estos.

9 Administración de Build

Se utilizarán dos tipos de builds en la realización del proyecto.

En primer lugar tenemos las builds informales realizadas por el equipo de desarrollo en sus entornos locales utilizadas para realizar tests sobre funcionalidades antes de realizar un push hacia la rama de desarrollo.

Luego tenemos builds realizadas por técnicas de integración continua, específicamente GitHub Actions (GitHub_Actions), las cuales se ejecutan automáticamente ante ciertos disparadores:

- ➢ Cuando se realiza un push a la rama de desarrollo/funcionalidad, se ejecutará un pipeline donde se compilara la versión del código fuente en el push dentro de una máquina virtual con el sistema operativo especificado (windows_latest) y se realizará el testeo de la aplicación en este entorno.
- ➢ Cuando se realice un merge hacia la rama main desde una rama de desarrollo, el mismo pipeline se ejecutará con el objetivo de comprobar que la integración de las nuevas funcionalidades no haya comprometido la estabilidad de la rama principal.
- ➢ Cuando se genere un nuevo release, se ejecutará un pipeline que realizará los mismos pasos que el mencionado anteriormente, con la excepción de que nos entregará el archivo ejecutable para incluir dentro del release.

Es la responsabilidad del Build Manager la creación y mantenimiento de los pipelines de build.

10 Administración de Release

Ante la realización de un release, el CM será responsable de generar una rama nombrada con el correspondiente formato (ver 8.1.1) donde se realizará la build de release. La versión compilada se hará disponible mediante la utilización de un pipeline.

Cada release será acompañado de documentación general acerca del programa, documentación específica al release, la versión compilada del programa, y el código fuente al momento del release.

Cada release será identificado con el tag v<mayor><menor><parche> (ver 6), y el último release será identificado con el tag latest.

Es la responsabilidad del release manager generar el cuerpo del release con la documentación específica al mismo donde se declara información tal como el propósito del mismo, cambios con respecto al release anterior, entre otros.

11 Backup

Se realizarán Backups del código fuente con frecuencia semanal en los equipos locales de los miembros del equipo en una carpeta con el formato JavaChess-BackUp-<Fecha>.

Además, posteriormente se implementará un sistema de backup remoto automático.

12 Nomenclatura

Se utilizará CamelCase para el nombramiento de paquetes, clases, funciones y variables dentro del proyecto, además se utilizará snake_case en mayúsculas para el nombramiento de constantes de clase.

Se utilizará la estructura de carpetas estándar de Maven y Spring, donde tenemos un paquete para el código fuente y un paquete para los tests. Además, el código fuente se contendrá en un paquete llamado com.<companyName>.<proyectName>.

Todos los paquetes y archivos estarán denominados en inglés, con las clases comenzando en mayúscula y los paquetes comenzando en minúscula.

13 Esquema de Directorios