Intel y Gentes

Juego de plataformas inspirado en “bubble bobble”

Plan de Gestión de Configuraciones (CM_PLAN)

Autores: Agüero, David - Bastida, Lucas - Miranda, Noelia - Palmiotti, Mauro

Fecha: 09/05/2020

Version del Documento: 1.0.0

Historial de Cambios

Version	Fecha	Resumen de Cambios	Autor
1.0.0	09/05/2020	Primera version	Intel y Gentes

1. Introducción

1.1 Propósito y alcance del plan

Este documento cubre el plan de Gestión de configuraciones para nuestro juego de plataformas inspirado en Bubble Bobble correspondiente al trabajo final de Ingeniería en Software.

El propósito del Plan de gestión de configuraciones es controlar la configuración de los requerimientos, documentos, software y herramientas utilizadas en este proyecto. También se definen los pasos a seguir por los integrantes del proyecto como la especificación de los elementos del proyecto que estarán bajo CM.

La gestión de configuraciones de software (SCM) es el proceso por el cual métodos y herramientas son identificadas para controlar el software a lo largo de la utilización y desarrollo del proyecto.

1.2 Propósito de las prácticas de SCM

- Garantizar la coherencia en la práctica de actividades de SCM
- Definir autoridades para apoyar las prácticas de SCM
- Mantener la integridad del producto durante todo su ciclo de vida
- Informar a los grupos e individuos afectados sobre el estado del proyecto
- Crear un historial verificable de los estados actuales y anteriores de productos de trabajo
- Mejora de procesos

1.3 Acrónimos/Glosario

- **SCM**: Gestión de configuración de software (Software Configuration Management)
- **CM:** Administración de la configuración (Configuration Management)
- **GPCM:** Gerente Global del Proyecto de Configuraciones (Global Project Configuration Manager)
- **TPCM:** Gerente de configuración del proyecto (Project Configuration Manager)
- **CCB:** Grupo de desarrollo del producto (Change Control Board)

1.4 Herramientas para la administración de configuraciones

Herramienta	Propósito	Controles
Git	Sistema de control de versiones distribuido para rastrear cambios en el código fuente durante el desarrollo de software.	Uso de funciones Branch y Merge para desarrollo de código en paralelo.
GitHub	Servicio de alojamiento de repositorio Git.	Uso de repositorios, etiquetado, control de tareas y control de lanzamientos.
GitHub Project boards	Aplicación de gestión de tareas de github.	Creación y asignación de tareas
GitHub Issues	Herramienta de gestión de defectos que proporciona GitHub. A través de ella podemos solucionar conflictos de diversos tipos, generando un nuevo Issue para cada conflicto, pudiendo etiquetar a cada uno con su correspondiente categoría.	Informar defectos de software.
TravisCI	Sistema de integración continua compatible con GitHub, utilizado para verificar el código fuente del proyecto automáticamente.	Enlazado automático con GitHub.
Gradle	Gradle es una herramienta de Construcción automática open source, diseñada para ser lo suficientemente flexible como para construir casi cualquier tipo de software.	Construcción. Ejecución de pruebas. Generación de Reportes. Documentación.
draw.io diagrams.net	Herramienta para creación de diagramas (UML, etc.)	Diagramas de Actividades, Paquetes, Casos de Uso, Componentes, entre otros

1.5 Formas de acceso.

- Forma de acceso al repositorio:
https://github.com/lucasbastida/bubble-bobble

- Forma de acceso a la herramienta de control de defectos:
https://github.com/lucasbastida/bubble-bobble/issues

- Forma de acceso a la herramienta de integración continua:
https://travis-ci.com/github/lucasbastida/bubble-bobble

2. Roles y Responsabilidades

2.1 Responsabilidades de gestión de configuraciones

Las actividades de gestión de las configuraciones dentro del proyecto son coordinadas por el Gerente Global del Proyecto de Configuraciones (GPCM), rol asignado a una persona.

Las actividades de manejo de configuraciones, procesos y sus respectivos procedimientos deben ser seguidas y respetadas por todos los miembros. Es la responsabilidad de cada persona seguir y aplicar el procedimiento de CM apropiado, de acuerdo con su rol.

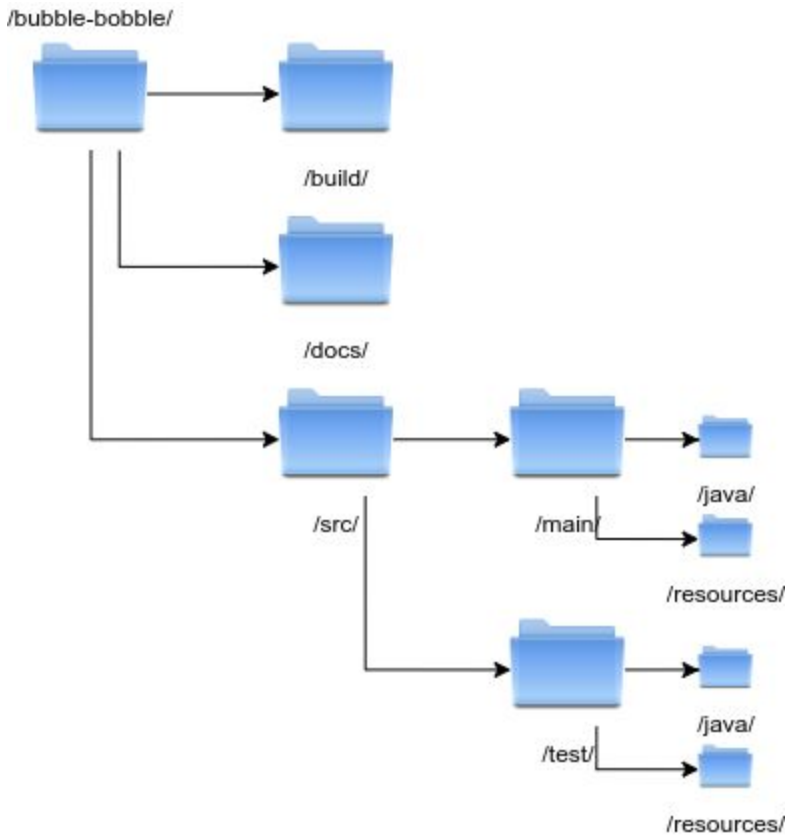
Nombre	Rol	Responsabilidades
Lucas Bastida	Gerente Global del Proyecto de Configuraciones (GPCM)	● Coordinar actividades de CM dentro del proyecto. ● Controlar las ramas principales y las versiones de lanzamiento ● Determinar cuándo deben ser creadas las ramas ● Determinar qué actividades y características corresponden a cada rama. ● Garantizar la integridad del producto ● Asistir en actividades de merge a las ramas principales y de release.
Noelia Miranda	Gerente de configuración del proyecto (TPCM)	● Asistir en la creación de etiquetas y ramas ● Asistir en actividades de merge ● Construcción de actividades en ramas específicas del equipo ● Garantizar la integridad del

		producto
	Equipo	● Ayudar a resolver conflictos durante el proceso de combinación de ramas. ● Asegurarse de que los criterios de calidad de los releases se cumplan. ● Seguir todos los procesos asociados, políticas y prácticas definidas por sus roles asignados.

3. Esquema de directorios.

3.1 Estructura de directorios y propósito de cada uno

Se utilizará un esquema de directorios en base al esquema de directorios estándar de Apache (Apache Standard Directory Layout) siguiente:



Ref.: http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html

3.2 Descripción del contenido/propósito de los directorios.

En la carpeta **src** se almacenará todo lo relacionado con el propio funcionamiento de la aplicación.

- En la carpeta denominada **main** se guardará el código fuente principal. Dentro de main estarán los archivos .java, los cuales deben comenzar en mayúscula y deben tener un significado que represente la identidad de la clase a la que pertenece.
- En la carpeta **test** se guardarán los documentos relacionados con las pruebas unitarias.

La documentación será almacenada en la carpeta **docs**. Estos documentos son: el plan de gestión de las configuraciones, documento de requerimientos, diagramas utilizados en los documentos, etc.

En la carpeta **build** se colocan los archivos compilados y el armado del programa como resultado de la herramienta de construcción gradle. Esto es lo que se entregará al usuario.

4. Normas de etiquetado y nombramiento de los archivos.

Cada **release** tendrá una etiqueta con el siguiente formato: *"vMAYOR.MINOR.PATCH"*

- *MAYOR* inicialmente será 0 hasta que el juego de plataforma sea jugable. DEBE ser incrementado al realizar grandes cambios en la estructura del proyecto, que no sean compatibles con versiones anteriores. PUEDE incluir cambios de nivel minor y/o patch. Las versiones patch y minor DEBEN ser reseteadas a 0 cuando se incrementa la versión major.
- *MINOR* se iniciara en 0 y DEBE incrementarse al agregar funcionalidades menores a una versión compatibles con versiones anteriores. La versión patch DEBE ser reseteada a 0 cuando la versión mayor es incrementada.
- *PATCH* se inicia en 0 y DEBE incrementarse al realizar bug fixes. Un arreglo de bug se define como un cambio interno que corrige un comportamiento erróneo. DEBE incrementarse cuando se introducen solo arreglos compatibles con la versión anterior.

Se considera al release tag v1.0.0 como el minimal viable product.

Para el versionado de los **documentos** se seguirá un esquema similar al descrito anteriormente (para los releases). En este caso, se definirá que un cambio realizado a una versión es de tipo *PATCH, MINOR o MAYOR* dependiendo del tamaño o la relevancia de las modificaciones a realizar en el documento.

5. Gestión de la configuración del código

5.1 Esquema de ramas

El modelo de branching utilizado está basado en un modelo conocido como:

"Trunk based development: short-lived featured branches"

Ref.: https://trunkbaseddevelopment.com/short-lived-feature-branches/

Este enfoque reduce la complejidad de fusionar eventos y mantiene el código actualizado al tener menos líneas de desarrollo y al hacer fusiones pequeñas y frecuentes. El esquema de ramas que adoptaremos será el siguiente:

master/trunk branch: rama en donde todos los desarrolladores trabajarán con acceso libre al mismo. Las modificaciones a esta rama serán mediante pull-requests desde las ramas de desarrollo únicamente si pasan los tests.

Feature branches de corta duración: Son ramas que resultan de la bifurcación directa del master branch. Estas ramas están destinadas a regresar como "pull requests" en la master branch para que el sistema de integración continua verifique el código antes de ser incluido en la master branch.

- La duración de la vida de la rama (antes de fusionarse y eliminarse) es de corta duración. Sólo debe durar un par de días (no más de 2 días). Por lo tanto se deberá integrar su trabajo a la main branch aun si su trabajo está en progreso.
- La cantidad de desarrolladores por rama debería mantenerse en uno. Estas ramas de característica de corta duración no se comparten dentro de un equipo para la actividad de desarrollo general. Se pueden compartir con el propósito de revisión de código.

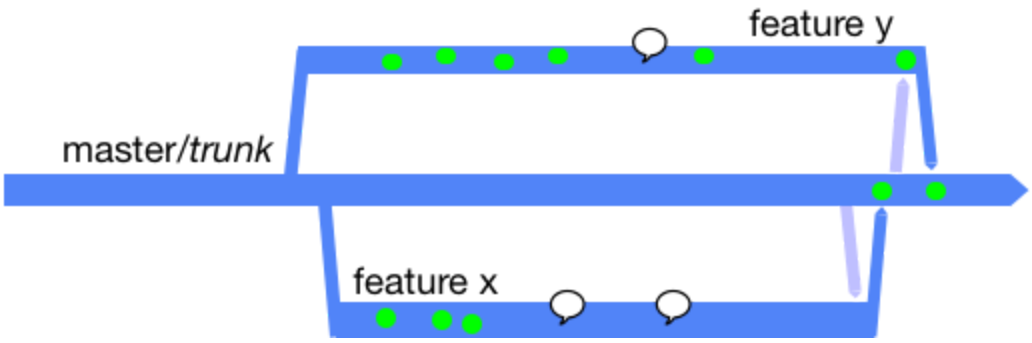
5.2 Política de etiquetado de las ramas

El nombre de las Feature branches será definido con el siguiente formato:

"feat/descripción-corta."

El valor de "descripción-corta" es una palabra o un par de palabras que nos ayudarán a entender de lo que se trata. algunos ejemplos son: *feat/arma-nueva, feat/arma-nueva2, feat/branch-story, etc*.
La corrección de defectos o bug fixing también sucede en dichas ramas.

5.3 Política de fusión de archivos.



Antes de terminar el trabajo en una de las feature branch, es necesario actualizar esta rama con el master branch (pull). Por lo tanto se debe realizar un merge a la feature branch para traerla más cerca del HEAD del master branch.

Solo se realiza el merge a master de la feature branch cuando se termine de trabajar en ella, justo antes de borrarla.

6. Gestión de cambios

La gestión de cambios es un proceso que ocurre después de que la documentación y el código fuente están identificados y aprobados. Los cambios incluyen modificaciones internas al enfoque documentado originalmente debido a simulaciones, resultados de tests o solicitudes externas de cambios a funciones.

6.1 Introducción y objetivos

La CCB es un comité que se asegura de que cada modificación es considerada apropiadamente, y autorizada antes de su implementación. La junta es responsable de aprobar, monitorear y controlar las solicitudes de cambios para establecer ramas de desarrollo de ítems de configuración. Las decisiones sobre las solicitudes de cambios son tomadas según el resultado de tests de calidad del producto. Los integrantes de la junta se contactaran via email o por teléfono para concertar una reunión extraordinaria, o discutir cambios referidos al proyecto fuera de ellas.

6.2 Miembros

La siguiente tabla muestra los integrantes del equipo que asiste a las reuniones de la CCB.

Rol en la CCB	Nombre
Gerente de ingeniería - CCB Chair	David Agüero
Gerente de lanzamiento - Coordinador de Issues	Mauro Palmiotti
GPCM	Lucas Bastida

6.3 Frecuencia de reuniones de trabajo

Reunión realizada semanalmente o según demanda.

6.4 Proceso de control de cambios

Ante un Change Request (solicitud de cambio), la CCB se reunirá y evaluará los siguientes puntos:

- Consecuencias de no realizar el cambio
- Beneficios del cambio
- Cantidad de usuarios afectados por el cambio
- Costos de realizar el cambio

Si ante la evaluación de estos factores, donde además se tendrán en cuenta sugerencias y retroalimentación por parte de los desarrolladores, la CCB decide que el cambio en cuestión está justificado, entonces se implementarán los cambios y se actualizarán los planes.

6.5 Herramienta de gestión de cambios

Para gestionar los cambios se utilizará la herramienta Issues de GitHub.

7. Gestión de defectos.

Ante la manifestación de un error/bug/defecto encontrado por algún usuario se deberá generar un nuevo Issue mediante “GitHub Issues” (ver sección Herramientas para la administración de configuraciones) con el siguiente formato:

Title	<corta descripción del defecto>
Write:	Steps: 1. <pasos para reproducir un bug> <información adicional/sugerencias>

A partir de esto se genera una nueva tarea en el “GitHub Projects board” que al ser finalizada por algún desarrollador supondrá la corrección del defecto. De ser necesario un cambio al enfoque documentado originalmente se procede según la gestión de cambios definido anteriormente.

8. Gestión de entregas.

8.1 Formato de entrega de releases

Se entregará un archivo .jar conocido como “Fat Jar” (también como “uber-jar”) el cual es un archivo autosuficiente que contiene las clases y dependencias necesarias para ejecutar la aplicación. No será necesario el uso de un instalador.

Este archivo se podrá descargar desde la página del proyecto en GitHub.

8.2 Instrucciones mínimas de instalación

1. Instalar Java Runtime Environment 8 o mas nuevo,
2. Ejecutar el archivo utilizando java

Ejecucion via Terminal:

```
>java -jar packagename.jar
```