UNIVERSIDAD NACIONAL DE CORDOBA

FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES

Trabajo Final Ingeniería de Software.

Alumnos:

- Gasparini, Roman.
- Tarres, Martin.
- Yepez, Franz.

Plan de Gestión y Control de las Configuraciones.

1. Introducción.

1.1. Propósito y alcance

Este documento trata sobre el plan de Administración de la Configuración (CM) para el "Proyecto Final". La intención del plan de CM es el control de la configuración de los requerimientos, documentos, software y herramientas usadas para este proyecto.

Las herramientas usadas son de código libre, pertenecientes a distintos entes.

1.2 Propósito de las prácticas de SCM

- Asegurar consistencia en las actividades prácticas de SCM
- Definir los organismos competentes para apoyar las prácticas de SCM
- Mantener la integridad del producto durante su ciclo de vida
- Informar a los grupos e individuos involucrados sobre el estado del proyecto
- Crear un historial de los estados actuales y anteriores de los productos en desarrollo
- Mejorar procesos

1.3 Referencias / Siglas / Glosario

- CCB : Change Control Board
- CM: Configuration Management
- PCM: Project Configuration Management
- SCM: Software Configuration Management
- Tag: Etiqueta
- CI: Configuration Item

1.4 Herramientas del CM

- Git: software de control de versiones.
- GitHub: plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones de Git.

  https://github.com/

- Travis: herramienta para la integración continua

https://travis-ci.org/

- Gradle: herramientas para la automatización de pruebas.

  http://gradle.org/

- Issues: herramienta de control de defectos. Lleva un registro de los defectos o cosas pendientes del proyecto. Propio de GitHub.

2. Direccion y forma de acceso a la herramienta de control de versiones.

   La herramienta que se utilizara en este proyecto como control de versiones es la plataforma de GitHub.
   Allí se creara un repositorio donde se subirán las diferentes versiones, y tendremos un control de las mismas con todos los beneficios que esta plataforma nos ofrece.
   El link del repositorio es el siguiente:

   https://github.com/martintarres/Final_IngSoft

3. Dirección y forma de acceso a la herramienta de Integración Continua.

   La herramienta que utilizaremos para la integración continua es Través Cl. Dicho servidor puede ser encontrado en el siguiente link:

   https://travis-ci.org/martintarres/Final_IngSoft

4. Administración del cambio

   4.1

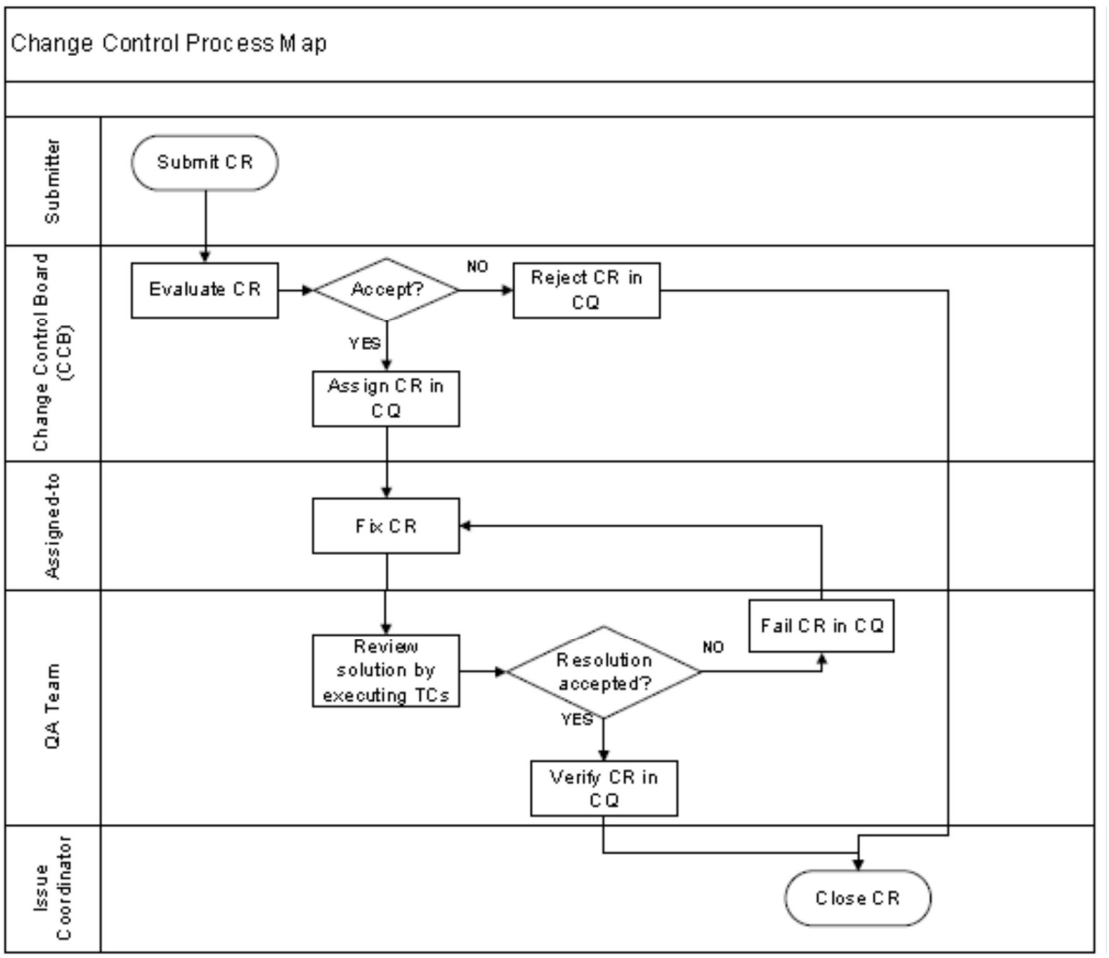
Integrantes de la CCB
Administrador de Ingeniería- CCB Chair
Administrador de Entregas – Coordinador de Fallas
Director de Ingeniería
GPCM
La forma de contacto de los integrantes del equipo es a través de mail, celular y redes sociales en general.

   4.2

La CCB se reúne cada vez que surja una nueva petición de cambio o problemas a solucionar en el software, por lo cual las reuniones no tienen una periodicidad definida. Es de vital importancia que se presenten el GPCM y Vice GPCM para cualquier toma de decisión.

4.3

Esquema del proceso de control de cambio



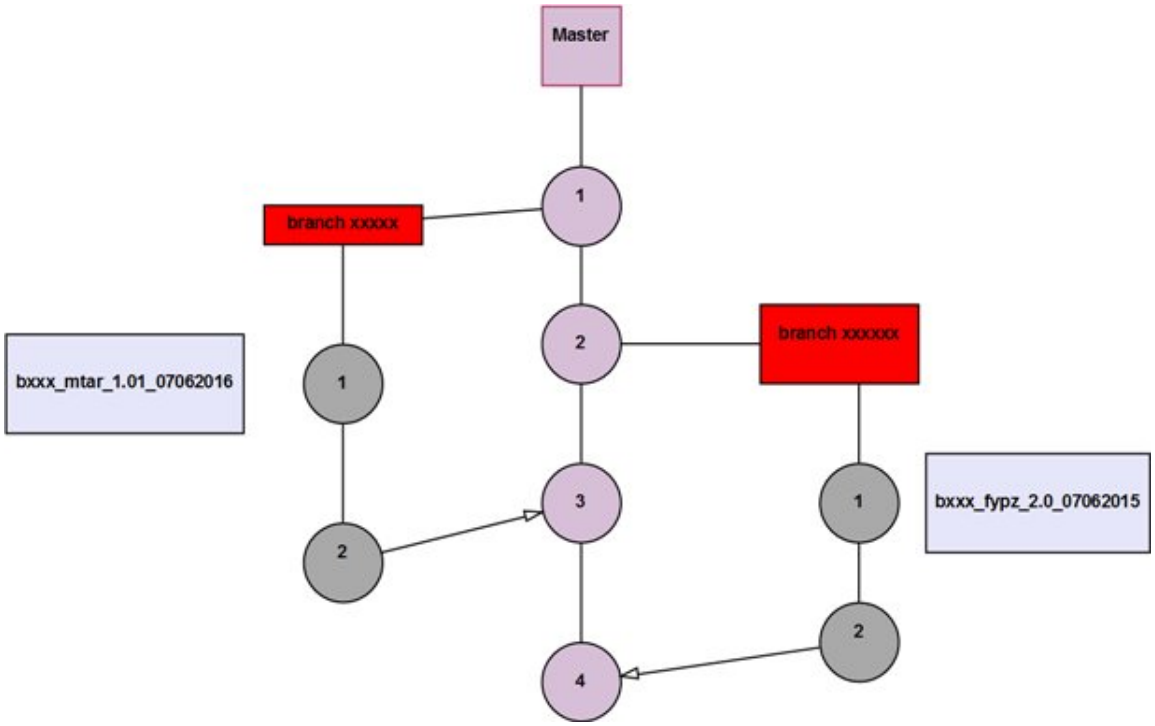
5. Normas de etiquetado y nombramiento de los archivos.

- Las primeras 4 letras del nombre deberán ser: b (correspondiente a branch), seguido de una alusión al branch donde se está realizando la suba del archivo.
- Luego de cada inciso deberá separarse uno de otro mediante un guion bajo. (_).

- Seguido del inciso a, se deberá colocar el nombre identificatiorio que le fue asignado a cada persona que trabaje en el proyecto.
- Próximo a eso se deberá poner a que versión corresponde, incrementando en uno su número. Ejemplo: El código está en la versión 1.03, cuando se agregue el próximo add será con la versión 1.04.
- Para finalizar ira la fecha en que se subió dicho archivo. (dd/mm/aaaa)

- Ejemplo:
  bste_mtar_1.03_09042016.
  Siendo: bste : branch siguiente.
  mtar: Martin tarres.
  1.03: numero de versión.
  09042016 : El día 9 de abril de 2016.

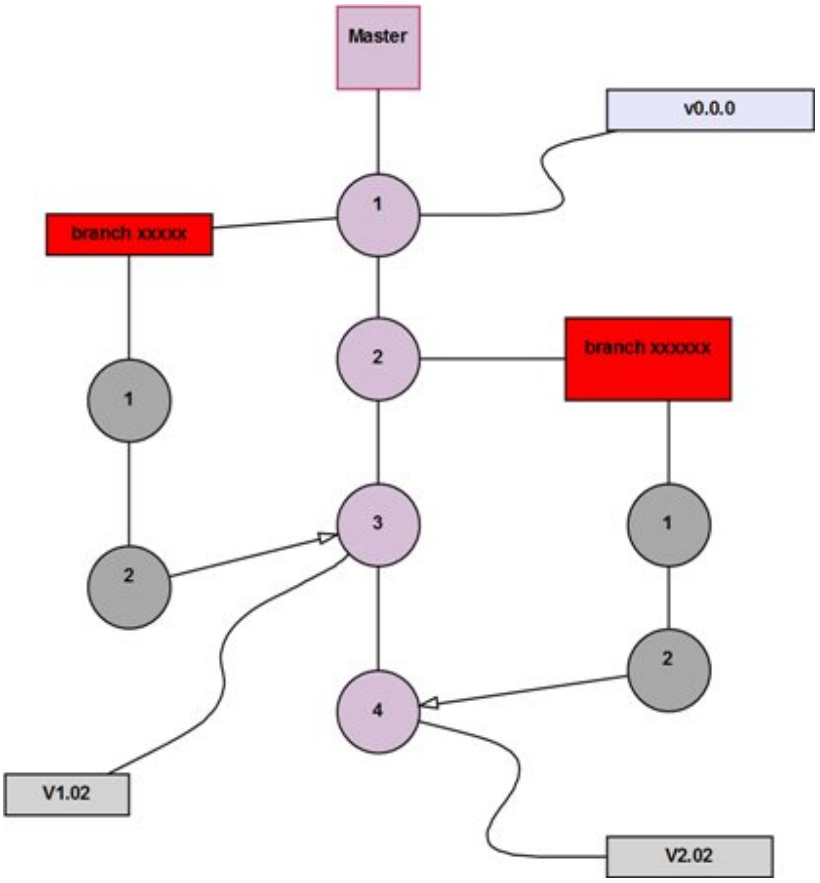
6. Plan de esquema de ramas a utilizar.
   - Master: Rama principal de integración, donde todas las nuevas funcionalidades serán añadidas. Siempre que las funcionalidades estén terminadas, testeadas y aprobadas.
   - Ramas de desarrollo: En donde se realizarán individualmente las nuevas funciones del software. Una vez terminado se procede al mergeo en la rama principal.



7. Archivos auxiliares de CM.
    - MergesNotes.txt: En ella se detallará de manera concisa las acciones de mergeo producidas durante el desarrollo del código. En ella se especificarán las ramas involucradas, la fecha, el desarrollador que la llevó a cabo, así como si el mergeo dio origen a la conclusión de alguna de las ramas mencionadas.
      Formato:
      DD/MM/AAAA (Nombre del desarrollador) Merge from (branch) to master
      Comentario
    - ReleaseNotes.txt: En ella se detallará el historial de release del proyecto.
      Formato:
      Fecha: DD/MM/AAAA
      Autor:
      Titulo:
      Comentarios:

8. Estrategias de mergeo.

Se realizará el mergeo cada vez que haya se agregue una funcionalidad que será reutilizada.
Se debe mergear sólo cuando la nueva funcionalidad de la rama a mergear se encuentre completa.



9. Nivel de calidad y criterios de realización.

Se seguirá una rama principal, llamada master, cuyo nivel de calidad corresponderá al necesario para realizar un release en la versión correspondiente del proyecto.
Ante una solicitud de cambio aprobada del cliente, se procederá a crear una nueva rama de desarrollo en la que el nivel de calidad puede variar antes de su finalización. Antes de ser mergeada a la rama master, es necesario que el código cumpla con los criterios de realización propios de la rama master, lo que implica que el nivel de calidad de la rama principal siempre se mantenga. Si luego del mergeo los Unit Test advirtieron un resultado no esperado, se procederá a corregir el problema. Si no hubiera una solución, la rama principal volverá a una versión anterior al mergeo.

10. Build Management.
Builds de Integración continua: Son ejecutados en aquellas ramas que demandan que el código no se rompa durante su desarrollo. Son útiles para identificar lo más temprano posible errores de código antes de su construcción formal. Se basan en el testo automático. Por cada commit se corren test que generan reportes sobre el estado actual del proyecto. En este caso fue utilizado Travis-CI en sincronización con el repositorio GitHub donde se encuentre el proyecto Calculadora. Los test son ejecutados por Gradle.