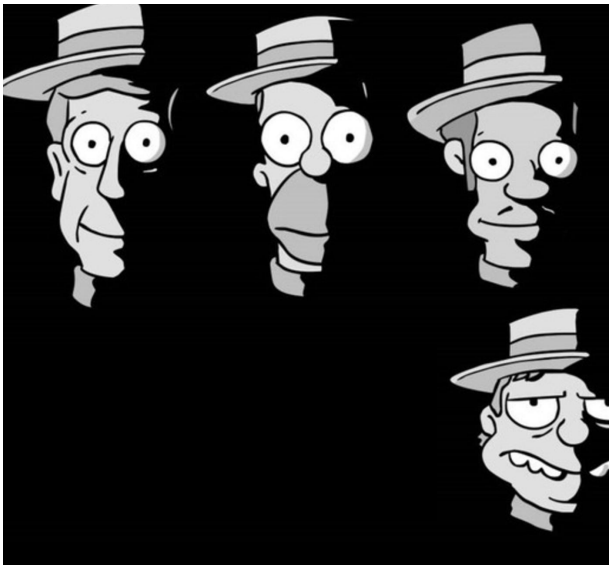
Los Borbotones

Llavero Digital

Configuration Management Plan



Autor(es):

Lorenzo, Facundo

Ojeda, Gastón

Riba, Franco

Roig, Patricio

Versión actual del documento: 0.2

Fecha:23/04/2022

Índice

Historial de Cambios

Versión	Fecha	Detalle
0.1	23/04/2022	Tomamos la decisión de hacer un llavero digital. Comienzo de la planificación.
0.2	27/05//2022	Aplicación de correcciones basadas en feedback de los supervisores especialistas, previo a 2da entrega
0.3	10/06/2022	Correcciones posteriores al feedback proporcionado

Glosario de Acrónimos

Acrónimo	Descripción
CI	Continuous Integration
IDE	Integrated Development Environment
CR	Change Request

Referencias

Referencia	Descripción	Enlace
Organización	Perfil del grupo de desarrolladores	https://github.com/LosBorbotonesISOFT

Repository	Repositorio principal del proyecto	https://github.com/LosBorbotonesISOFT/TP_Final_ISOFT

1. Introducción

1.1. Propósito y Alcance

Este documento presenta el plan de Gestión de las Configuraciones (Configuration Management Plan) del proyecto Llavero Digital. Mediante el plan de CM se pretende organizar las responsabilidades del equipo de trabajo, controlar la configuración de los requerimientos, documentos, software y herramientas utilizadas en este proyecto.
El proyecto busca en principio generar un software que corra sobre la plataforma Java VM de manera local. El sistema en resumen podrá gestionar de forma segura y cómoda el uso de contraseñas e información sensible de diferentes usuarios del producto.
Luego a futuro debería correr de forma remota en un servidor web y ser versátil en diferentes dispositivos para el usuario final.

1.2. Propósito del Plan de Gestión de las Configuraciones

El plan de configuración busca establecer una jerarquía de responsabilidades dentro del equipo de trabajo, dichas responsabilidades están distribuidas en diferentes acciones necesarias para llevar a cabo el proyecto de forma ordenada. El plan de gestión de la configuración detalla cómo vamos a registrar, supervisar, controlar y auditar la configuración.

Todos los requisitos de configuración de un proyecto, incluidos los requisitos de funcionalidad, los requisitos de diseño y cualquier otra especificación, deben identificarse y registrarse. A medida que se modifica el alcance del proyecto, se debe evaluar, aprobar y documentar su impacto en la configuración. Realizar un seguimiento de la configuración de nuestro proyecto en todo momento, saber en qué versión se encuentra tu configuración y tener un historial de las versiones anteriores.

1.3. Herramientas de Administración de Configuraciones

Herramienta	Propósito
IntelliJ IDEA	IDE
Git y GitHub	Control de versiones
Microsoft Visio	Diagramación UML
Selenium	Pruebas Automatizadas
Documentos de Google	Documentación
GitHub Issues con Github Desktop	Gestión de defectos
Integración Continua	GitHub Actions

1.4. Roles y Responsabilidades

Administrador de configuraciones *:* Gaston Ojeda
Encargado de los test(tester): Franco Riba
Encargado de programar el código: Facundo Lorenzo
Administrador de proyecto: Patricio Roig
Diseñador: Facundo Lorenzo
Asegurador de calidad del programa: Franco Riba
Documentador: Gaston Ojeda
Encargado del release: Patricio Roig
Administrador de versiones: Facundo Lorenzo

2. Esquema de Directorios

2.1. Estructura de Directorios y Propósito de cada uno

La estructura de directorios del proyecto está totalmente contenida en un repositorio remoto de github.

- **/BrainStorm** diagramamos con imágenes los posibles proyectos a tomar.

- **/Documentación** se encuentra el documento del plan de gestión de las configuraciones, el documento de requerimientos, la matriz de trazabilidad y un documento que identifica los casos de uso y pruebas del sistema.
- **/mvnSistemaLlavero** es el directorio raíz de trabajo del proyecto
    - **/.idea** es un directorio que guarda las configuraciones del IDE Intellij IDEA
    - **/META-INF** es un directorio reconocido por la plataforma de Java para configurar aplicaciones.
    - **/src** contiene el source code del proyecto más el archivo de configuración pom.xml.
        - **/main**
            - **/java** contiene el código escrito en lenguaje Java.
                - **/Entidad**: en este directorio se distinguen las entidades mediante sus atributos.
                - **/Interfaz**: en este directorio se guardan los archivos .form y .java que tiene como propósito mostrar interfaces gráficas (ventanas).
                - **/Logica**: en este directorio se escriben clases con el código correspondiente al back end del proyecto.
                - **/Repository**: en este directorio se escriben clases con el código responsable de persistir la información del sistema.
            - **/resources:** en este directorio se encuentran ciertos archivos de configuración.
                - **/Images**: directorio para guardar imágenes a utilizar dentro del sistema.
        - **/test**
            - **/java**
                - **/PruebasUnitarias:** en este directorio encontramos clases que implementan la librería de JUnit para hacer test unitarios referido al código fuente.
                - **/PruebasComplemento:** En este directorio encontramos clases destinadas a correr tests para chequear el compartimiento en conjunto de ciertas funcionalidades.
                - **/BigBang:** En este directorio encontramos código de test que busca correr el sistema en su totalidad.
    - **/target:** En este directorio encontramos archivos de compilación y package relacionados con maven y también es donde generamos nuestro ejecutable .jar con sus respectivas dependencias.

2.2. Normas de Etiquetado y Nombramiento de Archivos

El nombramiento de archivos dentro del código fuente se define por el nombre de la clase más el tipo de clase en cuestión y se utiliza el formato CamelCase, entonces, a modo de ejemplo, dentro del directorio Entidades encontraríamos a la clase PersonaEntidad.java. El nombramiento de los directorios se divide por responsabilidades a la hora de escribir código fuente.

3. Gestión de la Configuración del Código

3.1. Esquema de Ramas

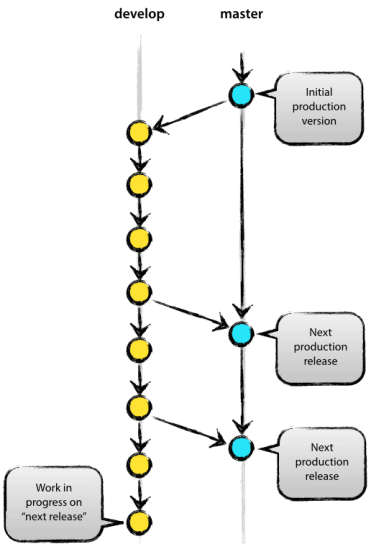
3.1.1. Main Branches

El repositorio cuenta con dos ramas principales, ambas con líneas de vida infinitas, estas ramas son:

- master (MB): rama de producción, en ella el código fuente de *HEAD* (último commit de la rama actual) se encuentra siempre en un "production-ready state" indicando que esa pieza de código es capaz de satisfacer la demanda de los usuarios, o los requisitos establecidos.
- develop (DB): rama de integración, en ella el código fuente de *HEAD* contiene los cambios de desarrollo más recientemente realizados. Cuando el código en esta rama alcanza un estado estable y se encuentra listo para convertirse en release se realiza un merge back hacia la rama *master* donde posteriormente se aplica un taggeo con el número de release.

Desprendidas de: master

Debe hacer back merge hacia: master



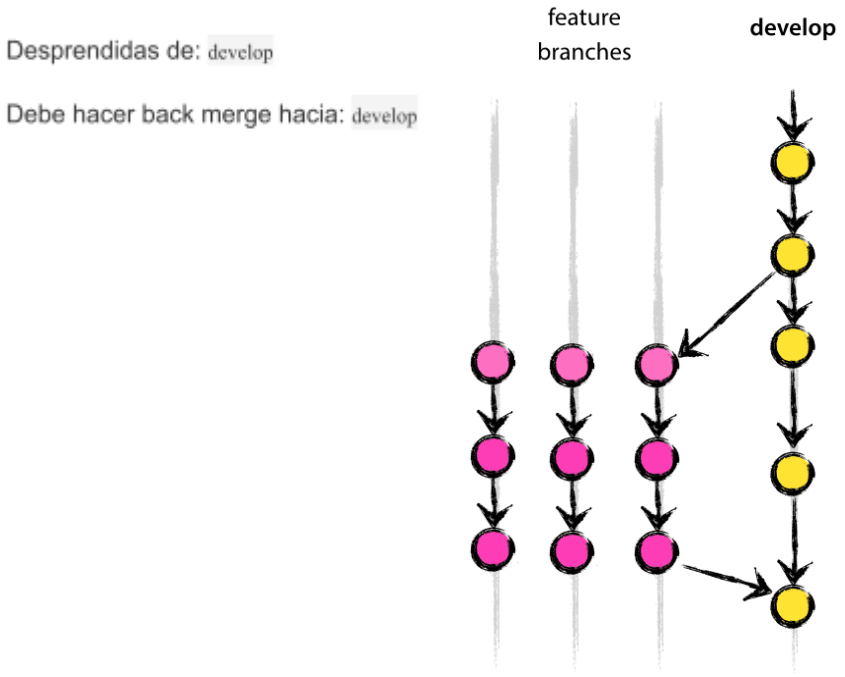
3.1.2. Supporting Branches

Ramas con un periodo de vida limitado, ya que serán eliminadas eventualmente.Contaremos con tres tipos, cada uno con un propósito específico y reglas claras, y dependencias bien establecidas:

- Feature Branches (FB)
- Hotfix Branches (HB)
- Release Branches (RB)

3.1.2.1. Feature Branches

Se contará con múltiples ramas de este tipo, cada una dedicada a la evolución de una feature particular, por lo que existirán durante todo el tiempo de desarrollo de la feature y eventualmente se hará un back merge hacia la rama *develop*. Además estas ramas solo existirán en los repositorios locales de los desarrolladores,y no en origin del repositorio remoto.

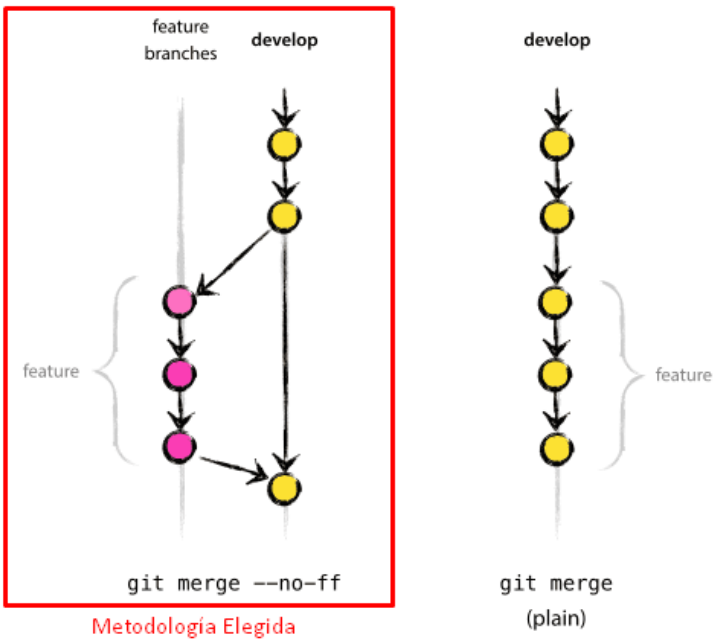


3.1.2.1.1. Generación de nueva Feature Branch

```
$ git checkout -b myFeature develop
Switched to a new branch "myFeature"
```

3.1.2.1.2. Incorporación de Feature Finalizada a la DB

```
$ git checkout develop
Switched to branch 'develop'
$ git merge --no-ff myfeature
Updating ea1b82a..05e9557
(Summary of changes)
$ git branch -d myfeature
Deleted branch myfeature (was 05e9557).
$ git push origin develop
-> PULL REQUEST
```



Metodología Elegida

3.1.2.2. Release Branches

Están destinadas a la preparación de nuevas versiones de producción, permiten el fixeo de **minor bugs** y la **preparación de los metadatos del release** (número de versión, fechas de compilación, etc.). Gracias a la utilización de estas ramas, la rama develop queda rápidamente liberada para recibir features de la siguiente

versión. La bifurcación de una nueva rama de lanzamiento desde develop sucederá cuando la develop branch (casi) refleje el estado deseado de la nueva versión. Al menos todas las características que están dirigidas a la versión que se va a construir deben ser fusionadas en develop en este momento.

Desprendidas de: develop

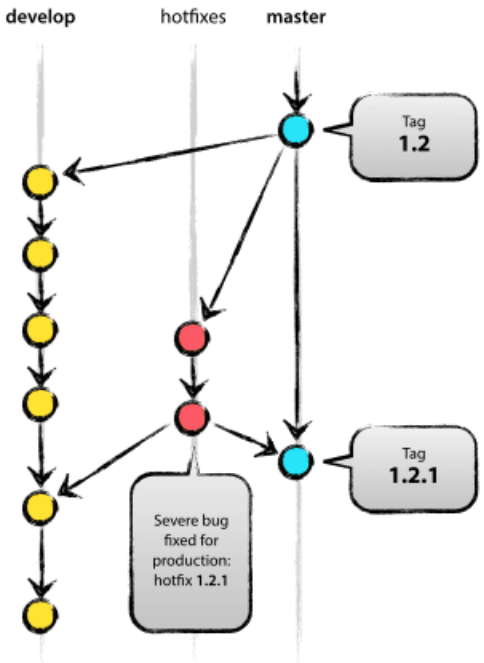
Debe hacer back merge hacia: develop, master

3.1.2.3. Hotfix Branches

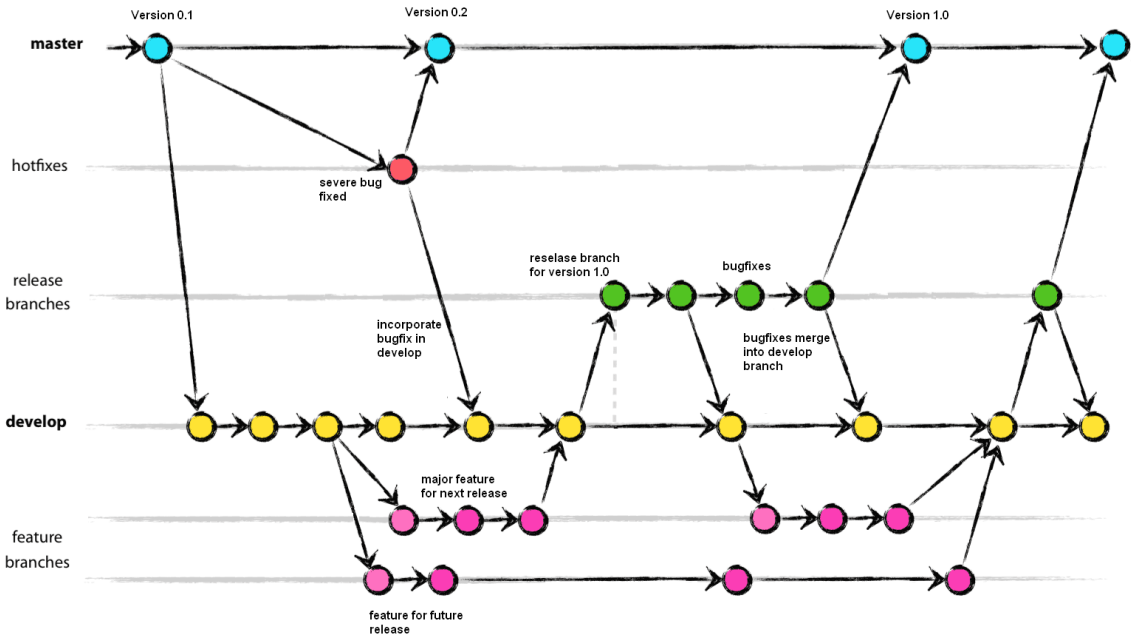
Estarán destinadas a solucionar errores urgentes y/o críticos, es decir **major bugs**, en código de una release ya lanzada. Cuando se halla un bug crítico en una versión de producción se desprende una hotfix branch desde master en el tag de la versión que presenta en bug. Esto permitirá a ciertos miembros del equipo seguir trabajando ininterrumpidamente en la rama develop mientras otros miembros del equipo realizan el hotfix.

Desprendidas de: master

Debe hacer back merge hacia: develop, master



3.2. Branching Scheme Overview



3.3. Política de Nombramiento de Ramas

Tipo de rama	Máxima cantidad de ramas activas	Nombre asignado
main	1	master
main	1	develop
hot fixes	1	hotfix
release	1	release
feature	Tantas como sean necesarias	feature:<briefDescription>

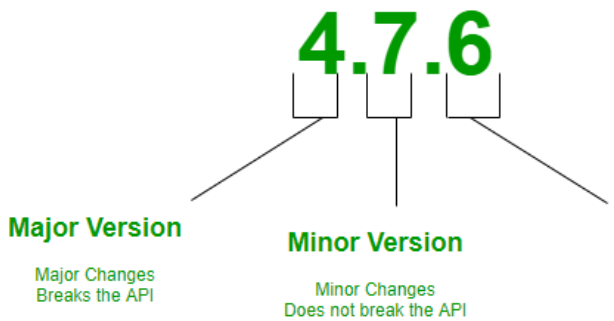
3.4. Política de Etiquetado de Releases / Versiones

Previous Head of master: <versionType>X.Y.Z	
Current Head of master: merge back from	**Tag Format**
release branch	<Patch >X.Y.(Z+1) <MinorVersion >X.(Y+1).0 < MajorVersion>(X+1).0.0
hotfix branch	<Patch >X.Y.(Z+1)

3.4.1. Numeración

Se ha optado por utilizar el sistema de versionado conocido como SemVer(Semantic Versioning), el mismo consiste de una estructura formada por 3 componentes (números enteros positivos), X.Y.Z
Donde:

- X: major version. Este número se incrementa de uno en uno. Cuando se incrementa, los componentes Y y Z se resetean (es decir comienzan en 0 nuevamente).
- Y: minor version. Este número se incrementa de uno en uno cada vez que se lanza una nueva funcionalidad en el sistema. Cuando se incrementa, el componente Z se resetea y el componente X permanece sin modificación.
- Z: patch version: Este número se incrementa cuando se debe indicar la evolución respecto al fixeo de bugs, por lo tanto será el resultado de la salida de una hotfix branch y/o feature branch.



Stage	Semver
Alpha	1.2.0-a.1
Beta	1.2.0-b.2
Release candidate	1.2.0-rc.3
Release	1.2.0
Post-release fixes	1.2.5

3.5. Política de Etiquetado de Commit Messages

3.5.1. Estructura del Mensaje

<type>[scope]: <description>
[optional body]
[optional footer(s)]

3.5.2. Descripción de los Tipos

feat: nueva funcionalidad incorporada.(relación directa con MINOR en el versionado semántico).
fix: corrección de bugs. (relación directa con PATCH en el versionado semántico).
build: cambios que afectan el sistema de compilación o dependencias externas (ámbitos de ejemplo: gulp, broccoli, npm)
ci: cambios en nuestros archivos y scripts de configuración de CI (ámbitos de ejemplo: Travis, Circle, BrowserStack, SauceLabs)
docs: Solo cambios de documentación
perf: cambio que mejora la performance
refactor: un cambio de código que no corrige un bug ni agrega una funcionalidad
style: cambios que no afectan el significado del código (espacios en blanco, formato, puntos y coma faltantes, etc.)
test: adición de pruebas faltantes o corrección de pruebas existentes
dev: desarrollo de un bloque de código del proyecto.

3.5.3. Breaking Changes

Un commit que contiene el texto **BREAKING CHANGE** al inicio del *[optional body]* o la sección de *[optional footer(s)]* introduce un cambio que rompe la compatibilidad en el uso de la API (se correlaciona con MAJOR en el versionado semántico). Un cambio en el uso de la API puede ser parte de commits de tipo. e.g., a fix:, feat: & chore: todos tipos válidos, adicional a cualquier otro tipo.

3.6. Política de Fusión de Archivos

En éste esquema de ramas la fusión de archivos funciona de la siguiente manera:

- En primer lugar la rama master se fusiona con archivos provenientes tanto de la rama hotfixes (versiones preliminares) como de la rama de release (versiones finales).
- Si una versión presenta alguna modificación en la rama de hotfixes, sus archivos modificados deben fusionarse tanto con los archivos de la rama master como con los archivos de la rama develop, para que el desarrollo continúe con los archivos modificados.
- De igual manera los que se encuentran en la rama de release deben fusionarse tanto con los de la rama master como los de la rama develop para continuar el desarrollo de futuras versiones.
- Por su parte las modificaciones en la rama develop no se fusionan con ninguna rama sino que se envía la versión próxima a lanzar a la rama de release que se encargará de enviarla a la rama master. La rama develop también se encarga de crear las ramas de features para implementar nuevas características.
- En última instancia las características implementadas en la rama de features deben fusionarse con los archivos de la rama develop para poder continuar con el desarrollo.
- A la hora de realizar la fusión de archivos se realizará una revisión del código, dicha revisión deberá hacerse como mínimo por dos miembros del equipo. La cantidad de aprobaciones necesarias para hacer merge será por lo tanto 2. La metodología utilizada para aprobar las fusiones será la de los Pull Request que provee GitHub.

4. Gestión de Cambios

4.1. Change Control Board (CCB)

4.1.1. Introducción y Objetivos

El grupo que conforma la CCB (Comité de Control de Cambios) tomará las decisiones relacionadas a la aprobación o el rechazo de los CR (Pedidos de Cambio) realizados ya sea por clientes o por desarrolladores, se evaluarán para ello los potenciales impactos generados por dichos cambios respetando prioridades que se les ha asignado previamente a cada uno. El objetivo de la CCB es la toma de decisiones en tiempo y forma, siguiendo una agenda clara

respecto a cambios en la documentación, código fuente, fechas de releases, mutación en los requerimientos, y cualquier otro elemento sobre el que se evalúa un posible cambio.

4.1.2. Miembros

Roll dentro de la CCB	Titular/es	Suplente/es	Función
General Coordinator (GC)	Roig, Patricio	Ojeda, Gastón	● Organización de Reuniones ● Supervisión de actividades ● Mediador ante discrepancias entre miembros de la CCB
Change Request Analyst (CRA)	Lorenzo, Facundo Riba, Franco	Ojeda, Gastón Roig, Patricio	● Análisis de factibilidad de los CR que le fueron asignados de forma previa a las sesiones ● Exposición ante la junta de los posibles riesgos, ventajas, consecuencias negativas y positivas de los CR discutidos.
Change Request Receiver (CRR)	Ojeda, Gastón	Riba, Franco	● Recepción de las planillas con los CR de clientes y desarrolladores. ● Clasificación del tipo de pedido de cambio. ● Asignación a los analistas según la clasificación asignada. ● Planificación de CR a discutir en cada sesión..

4.1.3. Frecuencia de Reunión de Trabajo

Motivo	Frecuencia	Duración
Reunión de reporte periódico y coordinación	Hasta 3 veces por semana, de forma virtual o presencial a convenir por los miembros.	Hasta 1 hora

Sesión de Agenda	2 veces por semana, de forma virtual o presencial a convenir por los miembros.	Hasta 2 horas

4.2. Proceso de Control de Cambios

Etapas del proceso:

1. Inicio de la solicitud de cambios: el cliente o desarrollador debe presentar ante el Change Request Receiver la planilla de solicitud, respetando el formato establecido y proporcionando todos los detalles solicitados en la misma.

2. El CRR revisa las planillas y las clasifica en función del área (desarrollo, marketing, control de versiones, etc) del proyecto a la cual se refiere la solicitud de cambio, luego informa y releva el CR a los analistas correspondientes a cargo de esa área.

3. Análisis de la solicitud de cambio: se determina por parte del analista asignado si el CR tiene razón de ser o no. Luego, se determinan las implementaciones, los impactos en el proyecto, tiempo, dinero, etc., que deberán ser expuestos antes los miembros de la CCB en las Sesiones de Agenda.
4. 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: la petición no resulta factible a nivel técnico o por el contexto actual y el contexto previsto del proyecto.
c) Postergada: la petición es factible, el contexto actual del proyecto no permite su aceptación, pero en el futuro será aceptada potencialmente.
d) Aprobada: la petición resulta factible.
e) Removida: el solicitante desestima la idea de cambios.

5. Implementación de la solicitud de cambio: se lleva a cabo la organización temporal, de recursos y de implementación.
6. Verificación y cierre de la solicitud: la implementación de los cambios es corroborada y se cierra la petición.

6.1. Herramientas de Control de Cambios

Para el control de cambios se hace uso de la herramienta GitHub Issues, siendo esta utilizada principalmente para la gestión de Bugs y siendo de preferencia para el equipo el acceso a través de la aplicación de escritorio Github Desktop. De forma complementaria y de ser necesario utilizan los siguientes tags a la hora de reportar un Issue:

- Brainstorming
- Build System
- Questions and Help
- Feature

6.2. Proceso de Gestión de Bugs con Issues de GitHub

Gestión de bugs: Gaston Ojeda.

El proceso consiste en que, por un lado, todos los integrantes del proyecto serán capaces de reportar bugs y generar issues al respecto. Todos estos issues son recibidos por el encargado de la gestión quien se encarga de asignarlos a una o varias personas, con su respectivo label de bug.

6.3. Reporte de Bugs

Cuando se quiere reportar un bug se se debe asignar un nombre con uno de los tags ya mencionados:

- Brainstorming
- Build System
- Questions and Help
- Feature

Seguido de una breve descripción del mismo ya que probablemente el bug no sea manejado por la persona que lo reportó. Una vez creado el issue a éste se le asignará automáticamente un número identificador #id

Al momento en que el bug fue tratado se debe realizar un commit en el cual se especifique el id del mismo con una descripción de qué se realizó, para que este comentario quede en la sección del issue y pueda ser revisado por posibles modificaciones.
Una vez que se corrobora que está todo correctamente solucionado se debe realizar otro commit en el cual se debe utilizar la palabra fixes seguido del id (fixes #id) para cerrar el reporte.

7. Releases

7.1. Formato de Entrega del Instalador

El software será entregado como archivo con extensión .Zip disponible de forma pública en el repositorio de la organización

7.2. Instrucciones Mínimas de Instalación

Dentro del archivo .Zip se encontrará el instalador con extension .exe el cual deberá ser ejecutado por el usuario y seguir los pasos que se indican