**Grupo Se Dice Atómico**

CALCULADORA POR LÍNEA DE COMANDO

Plan de Gestión de las Configuraciones

Autores: Armando, Gaspar

Ceballos, Matías

Zimmel, Ezequiel

Versión <1.0.0>

**Revisiones**

|  |  |  |  |
| --- | --- | --- | --- |
| **Versión** | **Fecha** | **Descripción** | **Autor** |
| <1.0.0> | <24/03/16> | <Versión Inicial. Contempla operación de suma y resta> | <G. Se Dice Atómico> |
| <2.0.0> | <24/03/16> | <Incluye operaciones de suma, resta, multiplicación y división> | <G. Se Dice Atómico> |
| <1.1.0> | <24/03/16> | <Incluye operaciones lanzadas en la versión 1.0.0, y añade operación de multiplicación> | <G. Se Dice Atómico> |
| <1.2.0> | <24/03/16> | <Incluye operaciones lanzadas en la versión 1.1.0, y añade operación de porcentaje> | <G. Se Dice Atómico> |
| <2.1.0> | <24/03/16> | <Incluye operaciones lanzadas en la versión 2.0.0, y añade operación de porcentaje> | <G. Se Dice Atómico> |

**Requerimientos de aprobación**

A continuación se listan aquellas personas cuya aprobación debe ser requerida para cada cambio mayor de este plan.

\*Las aprobaciones no son necesarios para casos de cambios menores

\*Las aprobaciones son solo necesarias si el cambio del plan es realizado por personas externas al plan.

|  |  |  |
| --- | --- | --- |
| **Engineering Manager** | **Date** | **Signature** |
| Armando, Gaspar |  |  |

|  |  |  |
| --- | --- | --- |
| **Release Manager** | **Date** | **Signature** |
| Ceballos, Matias |  |  |

|  |  |  |
| --- | --- | --- |
| **Process Manager** | **Date** | **Signature** |
| Zimmel, Ezequiel |  |  |

|  |  |  |
| --- | --- | --- |
| **CM / CM Backup** | **Date** | **Signature** |
| Zimmel, Ezequiel / Armando, Gaspar |  |  |

**Table of Contents**

**1 INTRODUCTION ......................................................................................................................... 4**

1.1 PURPOSE AND SCOPE ................................................................................................................ 4

1.2 PURPOSE OF SCM PRACTICES................................................................................................... 4

1.3 REFERENCES / ACRONYMS / GLOSSARY.................................................................................... 4

1.4 CONFIGURATION MANAGEMENT TOOLS ................................................................................... 5

**2 CONFIGURATION MANAGEMENT ROLES......................................................................... 6**

2.1 PROJECT CONFIGURATION MANAGERS ..................................................................................... 6

2.2 CONFIGURATION MANAGEMENT RESPONSIBILITIES .................................................................. 7

**3 CHANGE MANAGEMENT......................................................................................................... 8**

3.1 SCOPE ....................................................................................................................................... 8

3.2 IN CORE RELEASES AND CUSTOMER MODULES......................................................................... 8

*3.2.1 Technical CCB (Change Control Board) ........................................................................ 8*

3.3 IN SCRUM TEAMS .................................................................................................................... 11

**4 CONFIGURATION IDENTIFICATION ................................................................................. 12**

**5 <PROJECT> TEAMS ................................................................................................................. 13**

**6 SOURCE CODE CONFIGURATION MANAGEMENT ....................................................... 14**

6.1 CORE MODULES ...................................................................................................................... 14

*6.1.1 Branch definition: .......................................................................................................... 15*

*6.1.2 Tag definition: ............................................................................................................... 16*

*6.1.3 Auxiliary CM files: ........................................................................................................ 16*

*6.1.4 Merge strategy:.............................................................................................................. 18*

6.2 CUSTOMER MODULES ............................................................................................................. 20

6.3 QUALITY LEVELS AND DONE CRITERIA .................................................................................. 21

**7 BUILD MANAGEMENT ........................................................................................................... 23**

**8 RELEASE MANAGEMENT ..................................................................................................... 24**

**9 BACKUP AND DISASTER RECOVERY INFORMATION ................................................. 25**

**1 Introducción**

El objetivo de este documento es presentar la estrategia general para la Administración de Configuraciones de la aplicación CALCULADORA POR LÍNEA DE COMANDO (en adelante CLC).

**1.1 Propósito**

Este documento describe las actividades de gestión de configuración de software que deben ser llevadas a cabo durante el proceso de desarrollo del proyecto. Aquí se definen tanto los artefactos que se pondrán bajo control de configuración como los procedimientos que deben ser seguidos por los integrantes del equipo de trabajo.

**1.2 Alcance**

El tiempo de duración del proyecto está limitado a 3 semanas, por lo tanto se busca una rápida respuesta a los cambios, tratando que este procedimiento sea lo menos burocrático posible.

El Modelo de Proceso se basa en un desarrollo en cascada, dado que se ordena rigurosamente las etapas del ciclo de vida o producción del software. Resulta importante tener control sobre cada una de las instancias, de los productos generados en estas y de los cambios surgidos, evaluados y aprobados.

Se deben incluir al control de configuración la mayor cantidad de elementos posibles, tomando en cuenta siempre las restricciones dadas por la duración del proyecto y por la capacidad organizativa del grupo.

Ateniéndose a las condiciones fijadas, se propone:

* Garantizar la coherencia en la práctica de actividades fijadas en el plan.
* Definir los organismos competentes para apoyar las prácticas de SCM.
* Mantener la integridad del producto durante todo el ciclo de vida.
* Informar a los individuos afectados al proyecto sobre su estado.
* Crear un historial verificable de los estados actuales y anteriores de los productos de trabajo.
* La mejora continua de procesos de trabajo.

**1.3 Terminología**

* CLC Calculadora por línea de comando.
* CCB (Configuration Control Board) Comité de Control de Configuración.
* CI/ARTEFACTO (Configuration Item) elemento bajo gestión de Configuración.
* SCA (Software Change Authorization) Autorización de Cambio en el Software.
* SCM (Software Configuration Management) Gestión de Configuración del Software.
* SCMR (SCM Responsable) Responsable de SCM.
* SCR (System/Software Change Request) Petición de Cambio en el Sistema/Software.
* SQA (Software Quality Assurance) Aseguramiento de la Calidad del Software.
* SQAR (SQA Responsable) Responsable de SQA.

**1.4 Referencia**

[1] ANSI/IEEE Std 828-1990, IEEE Standard for Software Configuration Management Plans.

[2] 2002, Modelo de Proceso.

**1.4 Herramientas de Gestión de la Configuración**

|  |  |  |
| --- | --- | --- |
| **Herramientas** | **Propósito** | **Control** |
| Microsoft Word / PDF | Informes de estado del proyecto. Gestión y control de versiones de documentos. | Control de revisión electrónica. |
| Git | Implementar un [sistema de control de versiones](https://es.wikipedia.org/wiki/Sistema_de_control_de_versi%C3%B3n), mantiene el registro de todo el trabajo y los cambios en los archivos del proyecto. Permite que los distintos miembros del grupo colaboren. | Electronic revision control. Check-in/Check-out.  Source code branching and merging capabilities for parallel development. |
| StarUML 2 | Diseño diagramas de secuencia. |  |
| SmartDraw | Diseño de mapa de proceso de control del cambio. |  |
| Gradle/Travis | Herramientas destinadas a la [compilación](http://es.wikipedia.org/wiki/Compilaci%C3%B3n) y [ejecución de test](http://es.wikipedia.org/wiki/Testeo_de_software) de todo el proyecto, a los fines de poder detectar fallos cuanto antes. | Builds are automatically triggered based on changes to the repositories.  Manual build generation. |

**2 Gestión del CM**

**2.1 Roles y responsabilidades**

El equipo de CLC en lo relacionado con la Administración de la Configuración está conformado por las siguientes personas: un administrador de la configuración, un administrador del proyecto y un administrador de release. A continuación en la tabla se describen los roles y responsabilidades:

**2.2 Responsabilidades de gestión de configuración**

|  |  |
| --- | --- |
| **Rol** | **Responsabilidad** |
| ADMP | Se encarga de supervisar que el equipo de desarrollo utilice el repositorio durante el ciclo de vida del proyecto. Asimismo, se coordinará con el Administrador de la Configuración para la creación y ejecución del control de versiones.  Su tarea se extiende a definir los componentes de desarrollo, el control de accesos, las políticas generales, y la consecución de objetivos. |
| ADMC | Se encarga de las tareas de creación, administración y mantenimiento del repositorio de versiones.  Implementa las políticas definidas por el ADMP. |
| ADMR/DES | Aplicar los items de configuración al desarrollo del software.  Crear vistas de diseño con herramientas UML.  Promover cambios.  Implementar los pasos de los elementos al entorno de producción a partir de las líneas base creadas por el Sistema de Administración de Configuraciones |

**3 Gestión del cambio**

**3.1 Alcance**

La gestión del cambio es un proceso que ocurre después que la documentación y el código fuente del producto de referencia es identificada y aprobada. Los cambios incluyen cambios internos en el enfoque de documentado original debido a la simulación o resultados de pruebas o peticiones externas de cambios en las características o funciones.

**3.2**  **Proceso de Administración de Cambios**

Cada vez que se realiza una solicitud de cambio es deber llenar un formato que debe quedar publicado en el sitio web del proyecto, toda la información necesaria debe ser diligenciada y entregada al responsable de recibir esta solicitud.

* **Cambios en los Requerimientos**: Debe ser diligenciado el formato CLC.REQ.D.CambiosRequerimientos.docx y este se entregará al Líder del Grupo quien será el responsable de evaluar el cambio.
* **Cambios en el Diseño:** Este formato es el denominado CLC.AYD.D.CambiosDisenio.docx y el responsable deber ser el Líder de Desarrollo quien evaluará el impacto y realizará los ajustes necesarios de ser el caso.
* **Cambios en la Arquitectura:** Cambios en la arquitectura del proyecto debe llenarse el documento CLC.AYD.D.CambiosArquitectura.docx y se entregan al Líder de Desarrollo.
* **Cambios en las Herramientas:** Las herramientas usadas son responsabilidad del Líder de Soporte, quien evaluará licencias, disponibilidad en usar determinada herramienta entre otros aspectos a evaluar en el desarrollo del proyecto a esta persona se le entregará el documento CLC.CM.D.CambiosHerramientas.docx y se aprobara o rechazara el cambio.
* **Cambios en la Documentación:** La documentación del proyecto está a cargo del Líder de Calidad se deberá evaluar aspectos sobre la viabilidad de agregar nuevos documentos, se debe diligenciar el documento CLC.CM.D.CambiosDocumentacion.docx y se entregará a esta persona encargada.

**3.2.2 Junta de control de cambios técnicos (Change Control Board)**

El T\_CCB es un comité que asegura que cada cambio se considera adecuada por todas las partes y está autorizado antes de su aplicación . El T\_CCB es responsable de aprobar , supervisar y controlar las solicitudes de cambio, de establecer líneas de base de los elementos de configuración.  
Los elementos para ser revisados ​​por el T\_CCB son los cambios provocados por el Coordinador de Emisión (Desarrollador, ADMR).  
El ámbito de trabajo será la de aprobar / rechazar los cambios necesarios en los planes, documentos y código. Las decisiones se tomarán sobre las acciones que deben tomarse en base al resultado de las actividades de aseguramiento de la calidad del producto y del estado del producto después de cada ciclo de pruebas .

**3.2.1 Miembros**

La siguiente tabla muestra los miembros del equipo que asisten a las reuniones técnicas CCB de exigencia obligatoria.

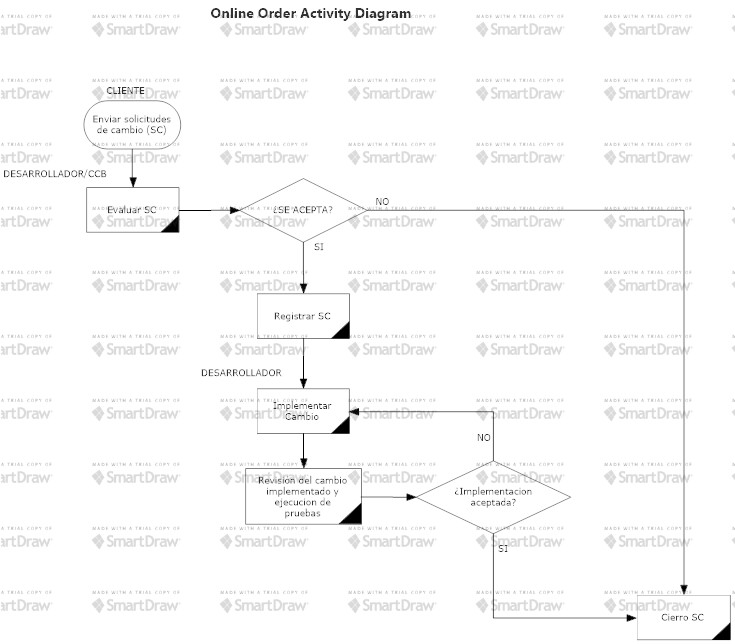
|  |  |
| --- | --- |
| **T\_CCB Rol** | **Nombre** |
| Engineering Manager - CCB Chair | <Eng. Name> (backup <Eng. Name>) \* |
| Release Manager - Issue Coordinator | <Eng. Name> (backup <Eng. Name>) \* |
| Engineering Manager | <Eng. Name> (backup <Eng. Name>) |
| PSO representative | <Eng. Name> (backup <Eng. Name>) |
| Engineering Director | <Eng. Name> (backup <Eng. Name>) |

**3.2.2 Herramienta de Gestión del Cambio**

Git se utiliza para gestionar los cambios aplicados al proyecto:

|  |  |
| --- | --- |
| **Campo** | **Valor** |
| URL | http://... |
| Schema Repository | ‘cq#432’ |
| Database | ‘cqqlewn’ |

**3.2.3 Mapa del proceso de control del cambio**



**3.2.4 Consideraciones:**

* El IC determinará si el CR es duplicado de otro CR , en cuyo caso se remitirá a la CR padres.
* El IC sólo puede rechazar una CR recién presentada si este tema fue discutido previamente en una reunión CCB. Un CR será rechazada después de que los miembros están de acuerdo. Si se rechaza la CR , el CI deberá dar una razón para el rechazo .
* Si un CR es un tema menor / cosmética o tiene una solución recta hacia adelante que no necesita ser discutido durante la reunión CCB , el CI puede asignar el CR previo a la reunión , teniendo en cuenta las condiciones siguientes :
  + El IC enviará una lista de los CR que fueron asignados con anterioridad a la reunión de los miembros del CCB opinión ( el CI debe comunicar todas las tareas realizadas y no se discuten en una reunión CCB )
  + Durante la reunión CCB , los miembros del equipo están de acuerdo en los CR previamente asignados , de lo contrario el SIR no acordadas se discuten en esta reunión

**4 Programa de administración de configuraciones**

**4.1 Identificación de la configuración**

**4.1.1 Métodos de identificación**

La nomenclatura para los elementos del proyecto, ya sean estos documentos, planes, modelos, etc., seguirá el siguiente patrón, con excepción de aquellos elementos ya construidos o que no puedan ser renombrados:

**CLC.DIS.T.NombreArtefacto-v.x.y.z**

En donde,

**CLC**: Sigla para identificar de manera única el proyecto.

**DIS**: Sigla de máximo tres (3) caracteres para identificar la disciplina a la que pertenece el elemento que está siendo nombrado en la metodología, puede ser una de las siguientes:

* UMLDS – Modelado del Diagrama de Secuencia
* UMLSC – Modelado del Diagrama de Control del cambio
* REQ – Requerimientos
* IMP – Implementación
* AYD – Análisis y Diseño
* PRU – Pruebas
* CM – Administración de Cambios y Configuraciones
* SRC – Código fuente

**T**: Carácter que identifica el tipo de elemento, puede ser uno de los siguientes:

* D: Documento
* P: Plan
* M: Modelo
* H: Archivo especial de herramienta.

**NombreArtefacto**: Nombre descriptivo del artefacto, no debe contener espacios y cada palabra debe empezar con su primera letra en mayúscula, debe incluir su extensión.

**v.x.y.z**: Versión del elemento al que se referencia. Se asignan 3 números que serán incrementados cuando:

* *x (mayor)*: el software sufre grandes cambios y mejoras.
* *y (menor)*: el software sufre pequeños cambios y/o correcciones de errores.
* *z (micro)*: se aplica una corrección al software, y a su vez sufre pocos cambios.

Ejemplos:

CLC.REQ.D.DocumentoVisión.doc (Documento de Visión del proyecto CLC)

CLC.GPY.P.PlanDesarrolloSoftware.mpp (Cronograma del Plan de Desarrollo de Software del proyecto de mantenimiento de CLC)

La estructura de carpetas definida para la aplicación CALCULADORA POR LÍNEA DE COMANDO es la siguiente:

**bin**

**classes**

**data**

**docs**

**lib**

**source**

**test**

**Especificaciones**

**Bin:** Los componentes ejecutables que son generados automáticamente por el IDE para generar documentación y versiones de la aplicación.

**Classes:** Archivos en bytecode de la aplicación generados automáticamente por el IDE que son de solo lectura son propios de la aplicación.

**Data:** Dedicada al almacenamiento y lectura de archivos a ser usados en la aplicación como por ejemplo archivos de parámetros o de almacenamiento temporal.

**Docs:** Documentación requerida para el desarrollo de la aplicación como diagrama de clases, descripción de requerimientos funcionales, descripción de la aplicación entre otros que sean útiles para el desarrollo de la aplicación, no es permitido definir el nombre del archivo relacionado con fechas.

**Lib:** Librerías adicionales requeridas en el uso de la aplicación para ser importadas, debe estar empaquetadas en archivos Java.

**Source:** La estructura creada para este componente fue especificada por fábrica y se debe mantener el estándar en las entregas, de igual forma los fuentes siempre se deben entregar con el nombre allí especificado sin incluir: fechas, consecutivos y/o números de requerimientos.

**Test:** Se contiene una estructura definida para la ejecución de pruebas automáticas, la estructura de este componente fue especificada por fábrica y se debe mantener el estándar igual que las entregas de código fuente, además los fuentes siempre se deben entregar con el nombre allí especificado sin incluir: fechas, consecutivos y/o números de requerimientos.

### ***4.1.2*** ***Líneas Base del Proyecto***

Todos los cambios autorizados deben hacerse sobre un estándar oficial en términos de líneas base (*baselines*) del proyecto. Estas líneas base deben ser creadas periódicamente y gestionadas durante el ciclo de vida, de tal manera, que representen configuraciones estables que se utilizan como “puntos de corte”, en los cuales se “congelan” los artefactos del proyecto en un punto de tiempo determinado.

Durante el ciclo de vida del proyecto, se deben crear líneas base cada vez que se tenga una planificación de liberación de un conjunto de requerimientos. El contenido de esta línea base son todos los artefactos que se hayan creado y modificado a lo largo del proyecto en la versión en la que se encontraban al momento de la creación de la línea base.

El nombramiento de la línea base se debe hacer utilizando el siguiente estándar:

**PROYE\_X.Y.Z\_DDMMAA**

En donde,

**PROYE:** Abreviación de máximo cinco (5) caracteres para identificar de manera única el proyecto o la aplicación

**X.Y.Z:** Identificador número de la versión del producto que representa esta línea base (*baseline*):

**X**: Número de la versión mayor del producto que va a ser liberada externa y oficialmente.

**Y**: Representa la versión consecutiva del producto en el entorno de integración y aseguramiento de calidad. Cada liberación (*release*) que tenga cualquier cambio debe tener un número consecutivo diferente.

**Z**: Número de la versión del producto cuando se libera de manera no intencionada, es decir, cuando los cambios hacen que se libere una versión correctiva o “*fix pack*” sobre una versión liberada previamente

**DDMMAA**: Día, Mes y Año de la creación de la línea base.

Ejemplo:

CLC\_1.0.0\_060416

**5 <PROJECT> Teams**

The scope of this document covers source code managed by the following teams:

· **Scrum Teams:** they are in charge of developing new features and functionalities through stories following the agile process. They will commit their code in sprint or feature branches so then be merged to the integration branch at the end of the sprint. Scrum teams also can be assigned to do extra work as bug fixing for core or customers modules, in those cases they will commit in the corresponding branch that is receiving those fixes.

· **Release Management Team:** RMT is in charge of performing all necessary testing to turn the core products Release state that will be defined according the release to be used then by customer components. Bug fixing of what the RMT finds will be committed on branch where core release builds are being done.

· **PSO Teams:** these teams are responsible for the delivery to end customers of the available product release. They will add specific customizations and scripts in the corresponding customer module that will be based on a specific core, das and standard package versions.

· **Rapid Reaction Team:** this team is responsible for addressing customer issues requiring urgent resolution in the form of hot fixes (i.e. that cannot wait to be incorporated and delivered with the next scheduled release) and these fixes shall comply with the DONE criteria. RRT team also can be assigned to do extra work as development of stories and bug fixing for core or customers modules, in those cases they will commit in the corresponding branches that receive those stories or fixes.

· **Components Team:** the goal of this team is to find out and identify different applications and modules that can be integrated to EGDE that add value to the product. Generally they work in dummy repositories until it is decided and approved to be implemented into production repositories.

· **Product Documentation Team:** this team is responsible of creating and maintaining the product documentation that will be delivered to the customer. Part of that documentation is integrated as part of the <PROJECT> product in the source code repositories and other part is maintained in Compass. Example of assets they manage: Admin guide, User guide, online Helper in <PROJECT> product, etc.

*For detailed information about branching schema refer to next section.*

**6 Administración del código fuente**

En esta sección se describen los distintos elementos de la gestión de fuentes. Cubre algunos aspectos sobre el esquema de ramificación, el etiquetado, la fusión de ramas y los niveles de calidad esperados para todo el producto.

**6.1 Core modules**

**6.1.1 Definición de ramas:**

Tipos de ramas que serán utilizados:

* Rama de integración: el master se define como la rama principal de la integración. Se crea de forma predeterminada por Git cuando se inicializa el repositorio.
* Ramas de desarrollo : son aquellas ramas donde se codifican nuevas características / funcionalidades.

Este tipo de rama es creada por el CM y son utilizados por los equipos de desarrollo. Aquí el desarrollo de funcionalidades estará comprometida hasta que se fundan a la rama principal master.

La creación de este tipo de rama puede fundarse en 2(dos) casos. Trabajo sobre funcionalidades Oficiales, ya previstas, que serán implementadas en la próxima release; y las No oficiales que pueden llegar integrarse en futuras versiones del software.   
El formato de nombre a utilizar en estas ramas es: PROYE-des<#>T por ejemplo: la CLC-des1Of para el desarrollo oficial, CLC-des2NOf para el desarrollo no oficial, etc.

* Rama Release: este tipo de ramas se crean para declarar las versiones liberadas de código y que se bifurcan de la rama principal master. El nombre de la rama de salida será PROYE-<versión>, por ejemplo: la CLC-1.0.0, CLC-2.1.0, etc. En correspondencia con la rama, la release lanzada se le agregara al final la fecha de publicación, quedando como sigue CLC-1.0.0\_040416 y CLC-2.1.0\_070416 respectivamente.  
  Si se realizan cambios menores en el código, la liberación de la nueva versión se realizará sobre la misma rama, por ejemplo: Release CLC-2.1.0\_070616 se liberó en la CLC-2.1.0, entonces hay una versión 2.1.1 qué es la continuación de 2.1.0 con correcciones menores , por lo que 2.1.1 está comprometido en la misma CLC-2.1.0.

DIAGRAMAAAAAAA

La idea principal de tener una definición del tipo de ramas, es mantener el código inestables aislado en ramas de desarrollo, con respecto al código estable que potencialmente será liberado.

**6.1.2 Definición de etiquetas:**

· Root tags: every time any branch is created, a tag formatted Root\_<branch\_name> shall be applied on the source version from where the branch is created; if the branch is created using Eclipse tool and is not based on an existing tag, this tag is applied automatically; if not, apply it manually. Examples: Root\_branch-5\_2\_1-rel, Root\_branch-5\_3\_1-rel, etc.

· Merge from tags: they are suggested to be applied every time a merge is performed in the source versions. The format is merge-from-<source>-<yyyymmdd>-<nn>. Examples: merge- from-5\_2\_1-rel-20101205-01, merge-from-story12748-20110222-01, etc. For details, please refer to merge strategies section.

· Build tags: they will be applied every time a formal build is created. These tags are used in order to have versions identified that are used to create an specific build and they shall be applied automatically when the corresponding Bamboo Blessed plan is run by the authorized CM.

The format of this type of tag depends on the CVS customer/component repository:

- in bezoar module: <Project><major\_version>-<version>-<buildid>, for example:

<Project>5-5\_3\_1-a1, <Project>5-5\_3\_2-ga1.

- in customer/component modules: <Project><major\_version>[-customer/component]-

<version>-<buildid>, for example: <Project>5-Telefonica-5\_3\_2-ga22, <Project>5-Telecom-

5\_3\_1-a3, <Project>5-das-5\_3\_1-ga2.

- in <Project>5\_standard\_package module: release-<major\_version>\_<buidid>, for example:

release-1\_78, release-1\_81, release-2\_8.

**6.1.3 Archivos auxiliares:**

Hay 3 archivos auxiliares que serán utilizados en el repositorio Git, ellos son:

* NotaMerge.txt: en este archivo se registra el historial de fusiones realizadas entre ramas específicas, con alguna información no detallada.
* NotaRelease.txt: este archivo se utiliza en las ramas para tener un registro de los cambios introducido. Se podrá añadir una nota en la parte superior de este archivo sobre el cambio que se esté realizando.
* NotaRequerimiento.txt: este archivo contiene los requerimientos que deberán ser añadidos al código.

**6.1.4 Estrategia de fusión:**

To perform merges in cvs modules, you have to specify the End tag (version to be merged) and the

Start tag (common version).

Before performing a merge, it is recommended to apply merge-from tag in source versions especially in those branches which is likely that more than one merge can exist. Merge-from tags are used to identify versions that where merged so in case another merge is needed in that same branch, we are able to use the last merge-from tag applied in the prior merge as the Common base version (start tag) and so avoid dealing with the same conflicts that we resolved in the prior merge.

Also optionally, you can apply a Pre-merge tag in case you have to deal with a very complex merge so you can have identified the stable versions before the merge is performed.

For example in the graph above, we need to merge the branch-5\_3\_1-rel to the HEAD to propagate some bug fixes to the integration branch.

The first merge is performed at 15-Mar-2011 and Start tag used is the Root\_branch-5\_3\_1-rel and the End tag is the merge-from-5\_3\_1-rel-20110315-01 that was just applied before the merge. For this merge also it was decided to apply the Pre-merge tag pmerge-from-5\_3\_1-rel-20110315-01 before perform the merge since it is a complex merge just in case we need to identify the prior stable versions.

Then we need to perform another merge from the same branch because other fixes were committed, so the Start tag used is the merge-from-5\_3\_1-rel-20110315-01 (the one utilized as end tag during prior merge), and the End tag is the just applied merge-from-5\_3\_1-rel-20110321-01. In this second merge we can see how the start tag is specified by the merge-from tag used in the prior merge, that way we’ll be only analyzing the gap introduced and avoiding dealing with conflicts that were resolved in the

prior merge.

In third merge the same approach as the second is followed, but start tag is merge-from-5\_3\_1-rel-

20110321-01 and End tag is merge-from-5\_3\_1-rel-20110402-01.

To register the merge history and Start / End tags utilized, we can use the MergeNotes.txt file which is located in the root of the project; this file will have its own version evolution in those branches that receive the changes from a merge and it is necessary to track this information, specially the HEAD branch. As optional information, list of changes merged can be listed. (See auxiliary files for details).

Also as part of merge activity, if applies, we can update the ReleaseNotes.txt file with list of changes that are being propagated to the branch so we can have the track of list of changes just introduced during the merge.

Merges from sprint branches to the HEAD or feature branches to the HEAD can be performed by the

CM or by an expertise developer tracked by the CM.

A scrum team can decide to isolate some stories in different feature branches during a sprint, so then the same scrum team can perform the corresponding merge from those features branches to the corresponding sprint branch or advice to the CM to merge it to the HEAD directly depends on what merge strategy is decided by the CM.

Merges from release branches to other release branches or to the HEAD to propagate fixes shall be performed by the CM.

*To see details of MergeNotes.txt and ReleaseNotes.txt files refer to Auxiliary CM files for details.*

**6.2 Los niveles y criterios de calidad**

El siguiente diagrama muestra los niveles esperados de calidad para todo el producto.

DIAGRAMAMAAMAMAMA

Como se muestra en la figura anterior, las ramas de Desarrollo serán de baja calidad hasta que estén listos para ser fusionado con la rama principal, para su posterior liberación.

La rama ( master) es el lugar desde donde se tomarán las versiones que potencialmente serán liberadas, por lo tanto se requiere de estabilidad en el software en esta instancia.

**7 Build Management**

Types of Build defined are:

· *Continuous Integration Builds:* they are executed on those branches that demands tracking the code is not broken during the development. They are useful to identify as early as possible code errors to be fixed before going into a formal build. They are setup to not apply any tag and to be run automatically for every commit running unit and system tests to generate the corresponding reports that have the results. In the corresponding Bamboo CI instance, useful information can be found like summary of unit and system tests, the list of changed files, the author and the date of the commit, etc. Since they are considered as development builds and their information is not necessary to have any persistence to be reproducible, every certain period of time that information is set to be automatically deleted from Bamboo.

· *Blessed Builds:* they are used to generate formal builds. They don’t run unit and system tests and are set to apply formal build tags. This type of builds shall be run only by the authorized configuration management. In the corresponding Bamboo Blessed instance, useful information can be found like list of files versions that changed, author of changes, link to download the build artifacts, Bamboo logs, etc. These builds are persistent and they can be reproducible.

· Informal local builds: these types of builds are created in a development environment and they are created mounting necessary CVS projects on the desired version in the development workspace and running deploy or dist ant targets. Corresponding release.properties and build

properties files shall be set according the needs. These will generate a deployable or zip file component. This process will not apply any tag and won’t generate persistence builds since they are informal. They are useful for example for developers to check their code before committing or for scrum testers to generate a deployable component to test its functionality. An example could be a developer codes a feature and, before committing, he generates a

deployable component to check it works as expected, after committing the code and letting the scrum tester know that the code is ready in the repository; the tester refreshes its project/s and generates other deployable of the component in his workspace to perform the exploratory test.

Having defined the types of builds we’ll see how they will be implemented in different branches in all modules:

· In development branches:

- Continuous integration builds management shall be implemented specially in sprint branches.

Feature branches might use continuous integration builds but it’s not mandatory, that will be analyzed if it needs a track of its development depending on the situation.

- Blessed builds here are not mandatory, although they are recommended to be used for exploratory tests and/or to generate builds artifacts for sprint reviews.

- Informal local builds here can be used, instead of Blessed (or in conjunction), to generate a deployable component for exploratory tests.

· In Release branches: Continuous integration and Blessed builds shall be implemented here since this type of branches will contain the code that will be used to generate formal builds potential to be released.

· In HEAD branch: This branch will have always a Continuous Integration build since it is the integration branch and all product versions will end up here. Also it might temporary have an associated Blessed build in case a version is being developed here until that is moved to a release branch (see section 6: source code management).

\* Informal local builds, from the developers’ point of view, can be used in any branch to check their code.

The tool that will be used to implement and manage that defined build approach is Bamboo and it will be configured to send emails to different groups subscribers reporting the status of each Build. Information and details about each Build can be found in **[<PROJECT>\_Bamboo]**.

**8 Release Management**

The Release build will be done in the corresponding release branch, starting from the release candidate of choice and adding the corrections of defects found along the system test cycles. For each cycle, the build shall be properly identified and, at the end of the cycle, evaluation of the product health against the defined release criteria will be conducted.

Release builds will be made available in compass and e-mail notification will be sent to the project team distribution list. Refer to section 4 to see the assets location definition.

**9 Backup and Disaster Recovery Information**

Project documents are backed up according to Compass policies.

<Code> server has a full backup every Monday and Thursday to a tape backup system on the corp net where <ServerAdminsitrator> has access.

Source Code:

cvs backups

The cvs repository is targz'd placed

in /mnt/san/cvsbackups/cvsbackup-date.tar.gd, and rotated Monday-Friday by

/home/cvs/bin/cvsbackup.pl. The most recent backup is hard linked to

/mnt/san/cvsbackups/cvsbackup.tar.gz. The current daily dump is transferred to tape with Retrospect

Retrospect

cvs backups are pushed from cron on cvs.ks.netopia.com as user cvs to

/Users/steven/sourcecode\_backup/cvs

Retrospect Schedule

Each backup happens the day after the labeled day. For example Monday backups run on Tuesday morning, Friday backups happen on Saturday morning. Retrospect backs up the following directories Tuesday-Saturday.