Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

EL PROCESO DE GESTIÓN DE CONFIGURACIÓN EN LAS EMPRESAS DE DESARROLLO DE SOFTWARE EN GUATEMALA

WALTER ERNESTO MÍNCHEZ SUTUC

ASESORADO POR EL ING. MARLON PEREZ TURK

GUATEMALA, OCTUBRE DE 2005

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

EL PROCESO DE GESTIÓN DE CONFIGURACIÓN EN LAS EMPRESAS DE DESARROLLO DE SOFTWARE EN GUATEMALA

TRABAJO DE GRADUACIÓN

PRESENTADO A JUNTA DIRECTIVA DE LA FACULTAD DE INGENIERÍA

POR

WALTER ERNESTO MÍNCHEZ SUTUC
ASESORADO POR EL ING. MARLON PEREZ TURK

AL CONFERIRSELE EL TÍTULO DE INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, OCTUBRE DE 2005

HONORABLE TRIBUNAL EXAMINADOR

Cumpliendo con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

EL PROCESO DE GESTIÓN DE CONFIGURACIÓN
EN LAS EMPRESAS DE DESARROLLO DE SOFTWARE
EN GUATEMALA,

tema que me fuera asignado por la Dirección de la Escuela de Ciencias y Sistemas con fecha 16 de Febrero de 2004

Walter Ernesto Mínchez Sutuc

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Murphy Olympo Paiz Recinos
VOCAL I	
VOCAL II	Lic. Amahán Sánchez Álvarez
VOCAL III	Ing. Julio David Galicia Celada
VOCAL IV	Br. Kenneth Issur Estrada Ruiz
VOCAL V	Br. Elisa Yazminda Vides Leiva
SECRETARIA	Inga. Marcia Ivonne Véliz Vargas

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Murphy Olympo Paiz Recinos
EXAMINADORA	Inga. Ligia María Pimentel Castañeda
EXAMINADORA	Inga. Elizabeth Domínguez Alvarado
EXAMINADOR	Ing. Luis Alberto Vettorazzi España
SECRETARIO	Inga. Marcia Ivonne Véliz Vargas

AGRADECIMIENTOS

A mis padres, por el apoyo incondicional y comprensión que me brindaron en todo momento a lo largo de mi carrera.

Mis tíos, Héctor Morales y Virgilia de Morales, por haberme recibido en su hogar dándome la posibilidad de iniciar la carrera.

Mis amigos y compañeros de estudios Ramiro Girón, Héctor Mendía, Edgar González y Juan Miguel Indekeu. Que con su apoyo y orientación pude lograr mi objetivo.

Toda mi familia por estar siempre al pendiente de mí y darme palabras de apoyo y consejos. Y por compartir conmigo las alegrías y penas que pasé a lo largo de la carrera.

DEDICATORIA

A Dios, por darme la fe, sabiduría y espíritu para seguir adelante y continuar en los momentos difíciles.

Mis padres, quienes son mis motivadores y quienes siempre me han apoyado y me seguirán apoyando.

Mis amigos dentro y fuera de la universidad, que siempre me apoyaron para lograr mis metas.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

GLOSARIO

AAS	Administración de Artefactos de Software
Artefacto	Es todo lo que envuelve el ciclo de vida del software, como documentos de requerimientos, código fuente, modelos de diseño o pruebas.
Baseline	Una *baseline* es un elemento de configuración o conjunto de elementos de configuración, formalmente, revisados, acordados o designados en cierto punto en el tiempo del ciclo de vida del proyecto.
Configuración	Término usado para identificar una versión específica de un producto completo.
DCT	Defect and Change Tracking -Seguimiento de Cambios y Defectos-.
Elemento de Configuración	-EC-. Una agregación de hardware o software que satisface con una determinada función y es diseñada para Gestión de Configuración
GCS	Gestión de Configuración de Software

PVOB	Project versioned object base. Repositorio de archivos, directorios y otros objetos de determinado proyecto.
Release	Versión de un producto de software que es entregada al cliente y puesta en producción.
SAM	Software Artifacts Management -Administración de Artefactos de Software-.
SCM	Software Configuration Management -Gestión de Configuración de Software-.
UCM	Unified Change Management -Administración Unificada de Cambios-.

OBJETIVOS

General:

Describir y evaluar el proceso de Gestión de Configuración de Software como una alternativa para tener un control óptimo acerca de los cambios al software desarrollado por empresas guatemaltecas, utilizando una de las soluciones de control de cambios como lo es el Unified Change Management "UCM" de Rational.

Específicos:

1. Identificar todas las tareas que conforman el proceso de Gestión de Configuración de Software.
2. Evaluar cómo la herramienta de Rational permite automatizar el proceso de administración de cambios.
3. Comparar los beneficios que tendrían las empresas guatemaltecas de implementar el proceso de Gestión de Configuración utilizando la solución de Rational, contra los métodos actuales de control de cambios.

RESUMEN

La gestión de configuración ha sido descrita como uno de los procesos de la ingeniería de software. Entre los beneficios que otorga se puede mencionar que asegura una alta productividad a un bajo costo.

La gestión de configuración de software es una disciplina creada para controlar la evolución de sistemas de software. Las actividades que involucra la gestión de configuración de software son la identificación, control, verificación de estado, auditoria, control de interfaces y control de subcontratados.

Las herramientas de gestión de configuración fueron desarrolladas para ayudar automatizar el trabajo necesario. Entre las principales capacidades que ofrecen están por ejemplo el mantener una biblioteca o repositorio de archivos, crear y almacenar múltiples versiones de archivos y proveer un mecanismo de bloqueo para modificaciones simultáneas. Rational ofrece el proceso Unificado de Gestión de Cambios, una solución fiable y comprensible que integra las características de dos herramientas: ClearCase para administración de artefactos de software y ClearQuest para seguimiento de cambios y defectos. Provee un proceso predefinido que organiza el trabajo a través de actividades y artefactos.

En este trabajo se evaluó los métodos de control de cambios y de gestión de configuración en las empresas de desarrollo de software en Guatemala y sobre las herramientas que utilizan. La evaluación se realizó por medio de una encuesta, entregada a los jefes de proyecto de cada empresa. De los resultados obtenidos se observa que son pocas las empresas de desarrollo que conocen el proceso de gestión de configuración, y todavía menos las que tratan de implementarlo. Esto depende del tamaño de la empresa, ya que en las empresas pequeñas, los métodos simples de control de cambios son suficientes para controlar los proyectos. La necesidad de métodos más efectivos se ve en las empresas grandes, que tienen varios proyectos al mismo tiempo, con varios programadores por proyecto.

INTRODUCCIÓN

La importancia de un buen control de cambios en el software desarrollado y en todo el proceso de desarrollo de software, radica en la calidad que alcance el producto final y el tiempo que tome desarrollarlo. El proceso de Gestión de Configuración de Software permite aumentar la calidad en el producto final, incrementar la velocidad de desarrollo y mejorar el rendimiento y productividad del equipo de trabajo. Las bases de éste proceso son:

- Que todos puedan conocer lo que está desarrollando cada miembro del equipo.
- Tener control respecto de las versiones del producto, no sólo con el código fuente sino, también, con la documentación y demás artefactos que se utilicen en el desarrollo.
- Administrar, adecuadamente, los cambios que se van a realizar en un producto final, ya sea por algún error que deba corregirse o por una nueva funcionalidad que se tenga que añadir.

La solución de Rational, llamada Unified Change Management -UCM-, permite automatizar todo el proceso de Gestión de Configuración y Seguimiento de Cambios y Defectos, a través de sus dos herramientas ClearCase y ClearQuest.

Es importante que las empresas guatemaltecas de desarrollo de software tengan un método para controlar los cambios al software que desarrollen. La aplicación del proceso y la solución UCM tendría muchas ventajas, por eso la importancia de describir éste proceso y las herramientas para llevarlo a cabo.

1. EL PROCESO DE GESTIÓN DE CONFIGURACIÓN DE *SOFTWARE*

Gestión de Configuración de *Software* -GCS- ha sido descrita como una disciplina que abarca la identificación, control, verificación de estado y auditoría de una entidad, como por ejemplo un programa de *software* o un sistema. También ha sido descrito como uno de los procesos que se dan dentro de un ambiente de ingeniería de *software*, donde varios procesos se realizan simultáneamente. Para poder entender el significado de la gestión de configuración y el por qué de su aplicación, se deben conocer sus inicios y su evolución desde su presentación en los años sesenta. La historia que abarca este capítulo inicia en los años sesenta, cuando el término de gestión de configuración fue formalizado y avanza hasta el presente con los principales estándares que se desarrollaron y que son utilizados hoy en día.

Este capítulo define la posición de la gestión de configuración en un proyecto, junto con otras disciplinas como la de control de calidad y su relación con el equipo de desarrollo. Se describe el plan de gestión de configuración y se dan algunas guías de lo que debe contener. Algo también indispensable es el uso de una biblioteca de *software* como herramienta de control de estado y control de acceso.

Las herramientas de *software* son una ayuda importante en el proceso de desarrollo, incluyendo el análisis, codificación, pruebas y demostración. Se dan algunas sugerencias para el análisis de una herramienta y el criterio de selección que se debe tener.

1.1 La necesidad e importancia de la Gestión de Configuración en el desarrollo de *software*

Hace poco tiempo, un producto de *software* normalmente era desarrollado por una persona, y no había mucha necesidad de Gestión de Configuración. Conforme los productos de *software* crecieron en tamaño y complejidad, su desarrollo requirió más de una persona sola. Los proyectos se mantuvieron relativamente fáciles de administrar cuando los equipos de desarrollo se formaban por dos o tres personas trabajando juntas, una a la par de la otra. Sin embargo, no tardó mucho para que los equipos de desarrollo crecieran a diez o hasta cien desarrolladores, que no necesariamente trabajaban en el mismo lugar.

Por esto, los procesos de GCS fueron desarrollados para administrar el cambio. En un principio, estos procesos fueron implementados manualmente. Uno o más *bibliotecarios* se dieron a la tarea de controlar quién podía accesar a los archivos de código fuente. Para modificar un archivo, el desarrollador llenaba un formulario -en papel- y lo llevaba con el bibliotecario. Éste formulario decía cuáles archivos se necesitaban modificar y por qué. El bibliotecario se aseguraba que ningún archivo fuera modificado por dos personas al mismo tiempo. Si un archivo estaba libre, el bibliotecario daba una copia al desarrollador y apuntaba por qué y a quién lo había entregado. El desarrollador, cuando terminaba, entregaba la copia modificada al bibliotecario, quien grababa el nuevo archivo y lo colocaba en el directorio apropiado.

Los beneficios del proceso de GCS son:

- Facilita la habilidad de comunicar el estado de documentos y código a medida que se vayan haciendo cambios.
- Debido a la administración, se asegura una alta productividad a un bajo costo.
- Incrementa la habilidad de dar mantenimiento y soporte una vez el *software* fue instalado o sacado a la venta. Esto se logra a través de los elementos de *software* bien definidos y el historial de desarrollo, que permiten modificaciones de bajo costo y con poco impacto para los usuarios y clientes.

1.2 Historia de la Gestión de Configuración

La Gestión de Configuración tuvo sus inicios en la industria de defensa, como una técnica de administración y una disciplina para resolver problemas de baja calidad, partes mal despachadas y partes mal fabricadas, que generaban costos muy altos. Por otro lado, otras grandes industrias como Polaroid y AT&T iniciaron procedimientos de control de cambios que les permitieran construir, respectivamente, cámaras y teléfonos de clase mundial.

El término fue formalmente definido en el ambiente gubernativo, así como muchos otros procesos e invenciones que se volvieron comunes para entidades civiles. La necesidad de una disciplina para identificar y controlar el diseño de equipos complejos y comunicar esa información, fue más aparente en la industria de la defensa. En 1962 la Fuerza Aérea respondió a los críticos problemas de comunicación y control, autorizando y publicando un estándar para Gestión de Configuración, AFSCM 375-1.

El comunicado, firmado por el General Bernard Schriver, comandante del Comando de Sistemas de la Fuerza Aérea, inicia diciendo:

'El Comando de Sistemas de la Fuerza Aérea está consciente del aumento en la necesidad de administración de requerimientos en nuestros programas. Constantemente debemos tomar ventaja de los nuevos y mejorados métodos que son desarrollados.

Éste manual de Gestión de Configuración contiene algunos de los aspectos más importantes de algunos de éstos nuevos métodos que son desarrollados.

Estoy totalmente consciente que los procedimientos, formatos y requerimientos de este manual son diferentes y en algunos casos presentan un cambio radical para algunos de nuestros métodos presentes y pasados de administración de programas. Como sea, es mi deseo que todos los requerimientos de éste manual sean implementados en todos los nuevos programas y sean incluidos en los programas presentes en donde sea apropiado.'

Los procedimientos, formatos y requerimientos fueron de hecho diferentes, especialmente para los ingenieros experimentados. Las reglas habían cambiado para enfrentar los retos de la década y deberían ser puestas en práctica. El comunicado AFSCM 375-1 indicaba un proceso para diseñar, desarrollar, construir, probar y entregar de una forma ordenada. La Gestión de Configuración fue la pieza principal para el diseño, desarrollo, construcción, prueba y operación del nuevo elemento a ser entregado porque era el comunicador y controlador del proceso.

Para éste tiempo, se habían hecho esfuerzos para asegurar que la documentación utilizada para desarrollar y construir un producto había cambiado por lo menos a una forma ordenada. Después del AFSCM 375-1, varios estándares fueron creados, la mayoría basados en el comunicado de la Fuerza Aérea. Entonces, de 1970 a 1971 la Fuerza Aérea emitió MIL STD 483, Prácticas de Gestión de Configuración para Sistemas, Equipo, Municiones y Programas de Computadora. Esta fue la primera vez que un estándar reconocía la necesidad de Gestión de Configuración tanto en el *software* como en el *hardware*.

1.2.1 La llegada de estándares de desarrollo de *software*

MIL STD 1679 fue desarrollado a finales de los 70, y fue emitido en Diciembre de 1978. Aunque fue estrictamente coordinado con la asociación de industrias y ampliamente revisado y comentado; muchas organizaciones de *software* sintieron que el documento final era demasiado restrictivo y aumentaría el costo de desarrollar *software*, especialmente considerando la imposición de 13 especificaciones y documentos. -*Software*, no importando el tamaño, parecía requerir la misma cantidad de documentos-. También hubo confusión por la resistencia que podría originar el estándar, porque era la primera imposición real en los requerimientos de *software*; tomó un tiempo para ver sus beneficios principales, mientras tanto se seguía con los métodos actuales de desarrollo de *software* conocidos como Análisis y Programación Top-Down.

En 1979 y después en 1981, dos conferencias de sistemas de *software* fueron llevadas a cabo por el Joint Logistics Commander en Monterrey, California. Fueron conocidas como Monterrey I y II. Uno de sus principales resultados fue la decisión de crear un estándar universal para el desarrollo de *software*.

El estándar, entonces conocido como MIL STD SDS y finalmente como DOD STD 2167, -DOD por sus siglas en inglés, Deparment of Defense-, pasó por tiempos difíciles y muchas revisiones. El estándar fue muy bueno desde el punto de vista de Gestión de Configuración. Dividía la Gestión de Configuración en las fases del ciclo de vida. En cada fase, describía las actividades a ser desempeñadas, el producto esperado de esas actividades, las revisiones de diseño que eran requeridas para la aprobación, y, más importante, el rol de GCS de mantener las descripciones documentadas y manejar los cambios subsecuentes. Era el tipo de documento que una persona nueva en la Gestión de Configuración de *software* podía leer y aprender en un período corto de tiempo.

1.2.2 El desarrollo de estándares para *software* comercial

Por muchos años la EIA -*Electronics Industries Association*- ha escrito numerosos estándares electrónicos, eléctricos y de protocolos de comunicación. La Sociedad de Ingenieros Automotrices -SAE, *Society of Automotive Engineers*- es famosa por estándares en el desarrollo y producción de automotores.

El Comité de Gestión de Configuración y Datos ha desarrollado varios estándares en forma de boletines y guías para el administrador de *software*. Estos incluyen los boletines en inglés:

4-1A – "*Glossary of Software CM Terms*"
4-2 – "*Software CM Identification*"
5A – "*Subcontractor Control*"
6-1A – "*Configuration and Data Management References*"
6-2 – "*Configuration and Data Management In-House Training Plan*"
6-3 – "*Configuration Identification*"
6-4 – "*Configuration Control*"
6-5 – "*Configuration Status Accounting – Textbook* "
6-6 – "*Configuration Audits – Textbook* "

Uno de los principales líderes en el área de estándares de desarrollo de *software* ha sido la Sociedad de Computadoras de la IEEE y el Departamento de Estándares de la IEEE. Ellos han desarrollado y siguen desarrollando un conjunto de estándares de *software* que están disponibles para organizaciones militares y comerciales, algunos de éstos son:

- Especificación de Requerimientos de *Software*, IEEE STD 830-1984.
- Aseguramiento de la Calidad del *Software*, ANSI/ IEEE STD 730.1-1988.
- Guía de IEEE sobre la Planeación para el Aseguramiento de la Calidad del *Software*, IEEE STD 983-1986.
- Planes para la Gestión de Configuración de *Software*, IEEE STD 828-1990.
- Guía para la Gestión de Configuración de *Software*, IEEE STD 1042-1987.
- Revisiones y Auditorias de *Software*, IEEE STD 1028-1988.
- Documentación de Pruebas de *Software*, IEEE STD 829-1983.
- Unificación de Pruebas de *Software*, IEEE STD 1008-1987.
- Planes de Administración de Proyectos de *Software*, IEEE STD 1058.1-1987.
- Planes de Verificación y Validación de *Software*, IEEE STD 1012-1986.

1.3 Definición de la Gestión de Configuración

Gestión de Configuración de *Software* -GCS, SCM siglas en inglés de *Software Configuration Management*- es una disciplina creada para controlar la evolución de sistemas de *software*. El proceso de GCS identifica los atributos físicos y funcionales de un *software* en varios puntos en el tiempo y realiza un control de cambios sistemático para los atributos identificados, con el propósito de mantener la integridad del *software* y poder darle seguimiento a través del ciclo de vida del *software*. Define de una forma más avanzada la necesidad de darle seguimiento a los cambios y la habilidad de verificar que la entrega final del *software* tiene todas las mejoras planeadas que se suponen forman parte de ese *release*.

Los problemas de *software* más frustrantes son a menudo causados por una pobre Gestión de Configuración. Los problemas son frustrantes porque toma tiempo corregirlos, a menudo suceden en el peor momento y son totalmente innecesarios. Por ejemplo, un error complicado que fue corregido a un costo muy alto reaparece de repente; una funcionalidad desarrollada y probada falla misteriosamente ó un programa completamente probado no trabaja.

La gestión de configuración ayuda a reducir éstos problemas coordinando el trabajo de varias personas que trabajan en el mismo proyecto. Sin ese control, el trabajo a menudo crea conflictos, y resultan problemas como por ejemplo:

- Actualizaciones Simultáneas: cuando dos o más programadores trabajan separadamente en la misma parte del programa, el último en hacer cambios puede fácilmente destruir el trabajo de los otros.
- Código compartido: a menudo, cuando los errores son corregidos en código compartido por muchos programadores, algunos de ellos no son notificados.

- Código común: en sistemas grandes, cuando funciones comunes de un programa son modificadas, todos los usuarios deben saberlo. Sin una efectiva administración del código, no hay manera de asegurar que todos los usuarios sean informados.

1.3.1 Definición formal de las actividades de GCS

El término *configuración* es un término usado para identificar una versión específica de un producto completo. El término *elemento de configuración* -EC- se define como una agregación de *hardware* o *software* que satisface con una determinada función y es diseñada para Gestión de Configuración. El término *gestión* se describe como el acto o el arte de evaluar y hacer decisiones acerca de la configuración de un producto de *hardware* o *software* que viene de una forma conceptual hacia un elemento físico para entregar.

Gestión de Configuración de *Software* -GCS- se describe entonces como el procedimiento de administración que incluye lo siguiente:

- Identificación de una Configuración: la selección de documentos que identifican y definen las características de configuración principales de un elemento.
- Control de una Configuración: el control de cambios a una configuración y sus documentos de identificación.
- Verificación de Estado de una configuración: el recuento y reporte de la implementación de cambios a una configuración y sus documentos de identificación.
- Auditoria de una Configuración: la revisión de un elemento para ver que cumpla con la identificación de una configuración.

- Control de Interface: el proceso de identificar todas las características relevantes para la interconexión de dos o más elementos de configuración proveídos por una o más organizaciones y el control de estas características.
- Control de Subcontratados: la administración de subcontratados o vendedores.

Con más detalle, las seis actividades son definidas como:

1.- *Identificación*: un *software* es normalmente hecho por varios programadores. Cada programa, su respectiva documentación y datos pueden ser llamados un "elemento de configuración" -EC-. El número de EC's en cualquier proyecto de *software* y el grupo de artefactos que forman un EC, dependen del proyecto. El producto final está formado por un grupo de EC's. El estado de los EC's a determinado punto en el tiempo es llamado *baseline*. Una *baseline* sirve como punto de referencia en el ciclo de vida del *software*. Cada nueva *baseline* es la suma de una vieja *baseline* más una serie de cambios aprobados hechos en el EC. Una *baseline* tiene los siguientes atributos:

- Funcionalidad completa: Las características y funciones de una *baseline* en particular son documentadas y están disponibles como referencia. Así, las capacidades del *software* en una *baseline* en particular pueden ser conocidas por todos.
- Calidad conocida: la calidad de una *baseline* está bien definida. Por ejemplo, todos los errores conocidos serán documentados y el *software* será sometido a una ronda completa de pruebas antes de ser definido como *baseline*.
- Puede ser recreada pero no cambiada: una *baseline*, una vez definida, no puede tener cambios. También, todos los EC's están bajo un control de versiones para que la *baseline* pueda ser recreada en cualquier punto del tiempo.

2.- *Control*: El proceso de decidir, coordinar los cambios aprobados para los EC's propuestos e implementar los cambios en la *baseline* apropiada se denomina control de la configuración. Algo importante es que el control de la configuración sólo guía el proceso después de que los cambios fueron aprobados. El acto de evaluar y aprobar cambios al *software* está definido en un proceso completamente diferente llamado Control de Cambios.

3.- *Verificación de Estado*: es el proceso de llevar un registro para cada release. Este procedimiento da un seguimiento a lo que hay en cada versión del *software* y a los cambios que se definieron para la versión actual. La descripción del estado de una configuración mantiene un registro de todos los cambios hechos a una *baseline* para llegar a una nueva.

4.- *Auditoria*: es el proceso de asegurar que la nueva *baseline* tiene incorporada todos los cambios planeados y aprobados. El proceso define una verificación de que todos los aspectos funcionales del *software* están completos y también estén completos los programas, documentos y datos que van a ser entregados. Es una auditoría realizada al producto que se va a entregar antes de que sea puesto en marcha.

5.- *Control de Interface*: es la evaluación, coordinación y aprobación o desaprobación de todos los cambios propuestos para establecer interconexiones físicas y funcionales como fueron definidas en las especificaciones, documentos y diagramas.

6.- *Control de subcontratados*: es la evaluación, coordinación y aprobación o desaprobación de todos los cambios acordados por el subcontratado a la documentación aprobada de una configuración y el monitoreo del desempeño del subcontratado. Se refiere al monitoreo de aquellos que son contratados para desarrollar *software* para el sistema en algunas áreas específicas que la empresa principal no trabaja.

1.4 Planeación y organización de la Gestión de Configuración

La planeación y organización de la actividad de GCS dentro de una compañía, abarca varias consideraciones. La consideración más importante es el alcance y magnitud del ambiente en que la actividad va a ser desempeñada. La actividad de GCS, sin importar qué tan bien estructurada esté, no puede existir sin procedimientos acordados y documentados. A cualquier nivel, debe ser el documento guía para todos los involucrados en el proyecto, incluyendo al cliente -algunos quieren saber cómo el *software* que ordenaron está siendo desarrollado-.

1.4.1 El plan de Gestión de Configuración

La clave para una GCS exitosa es el plan escrito durante las primeras etapas de la preparación del sistema, detallando cómo la GCS va a ser desempeñada. Hay varios documentos de Planes de GCS dados por varias fuentes, incluyendo al Departamento de la Defensa de Estados Unidos, la Agencia Federal de Aviación de los Estados Unidos, NASA -National Aeronautics and Space Administration-.

También la IEEE ha emitido varias guías de Planes de GCS para varias funciones. La estructura principal de los planes es la siguiente:

- Alcance y Propósito
- Organización y recursos
- Especificación de las actividades de GCS
- Objetivos
- Notas y apéndices

La elaboración de un plan parece sencilla al ver los requerimientos, pero la sencillez desaparece al momento de empezar a escribirlo. La primera inclinación al momento de no saber cómo empezar, es buscar un plan que ya ha sido escrito y preferiblemente que ya haya sido aceptado por el cliente. El resultado de esto, muchas veces, que el plan es rechazado porque parece no dirigir el proyecto actual o parece no haber sido escrito para el sistema que va a ser implementado.

A continuación se muestran algunos de los errores más comunes al escribir un plan:

ERROR: Plagiar el plan de otro plan del proyecto

RESULTADO: El plan muchas veces no dirige todos los requerimientos del proyecto actual.

ERROR: El plan fue escrito como propuesta, pero no fue actualizado con la aprobación del contrato.

RESULTADO: No refleja requerimientos nuevos o modificados por el cliente. También ha sido escrito más para vender un proceso de GCS que para realmente proveer el control necesario en el proyecto.

ERROR: Los objetivos del proceso de GCS no permanecen actualizados con los cambios hechos al proyecto

RESULTADO: Resulta en información equivocada para preparar las fechas de revisiones y auditorias.

ERROR: Jerarquía de *software* equivocada.

RESULTADO: Puede causar que sean creados más documentos de los necesarios, o menos.

ERROR: Proceso de cambios a los procedimientos y datos del *software* incompleto, muy simple o muy restrictivo.

RESULTADO: Un proceso de cambios pobre y costoso, que lleva mucho tiempo desarrollarlo debido a que es muy restrictivo. Causa pérdida de tiempo.

ERROR: No hay medios para archivar o almacenar al cerrar el proyecto.
RESULTADO: Pérdida o descontrol en los documentos o código. Incapacidad de reconstruir el sistema para reutilización o para demostración.

Hay riesgos que uno debe prever al utilizar un programa que ya haya sido escrito, puede utilizarse y no hay ningún impedimento para hacerlo, especialmente si uno escribió ese programa. El principal propósito de preparar un plan de GCS es diseñar y documentar el proceso de GCS lo más pronto dentro del ciclo de vida del *software*, para que sea identificado y aceptado por el equipo del proyecto antes de comenzar el desarrollo.

1.4.2 Alcance y objetivos

Esta es la introducción del plan y explica al lector qué es el proyecto, los objetivos del proyecto, las entidades de conexión que están envueltas y la razón por la que el plan es escrito. Esta sección también puede incluir un revisión a la función de GCS. En algunos documentos puede ser una sección aparte, pero en cualquier caso es una buena oportunidad para expandir la filosofía y metodología que es implementada en la GCS para cualquier proyecto.

1.4.3 Organización y recursos

Esta sección describe la organización del proyecto en forma gráfica y narrativa, e indica el nivel al que se posiciona la GCS en el proyecto y dentro de la compañía, indicando quién tiene las responsabilidades de la GCS. El segmento de recursos en ésta sección describe el número de personas, y sus habilidades, que van a llevar a cabo la GCS durante una parte específica del ciclo de vida del *software*. También incluye responsabilidades y autoridades.

1.5 Establecimiento y mantenimiento de la biblioteca de *software*

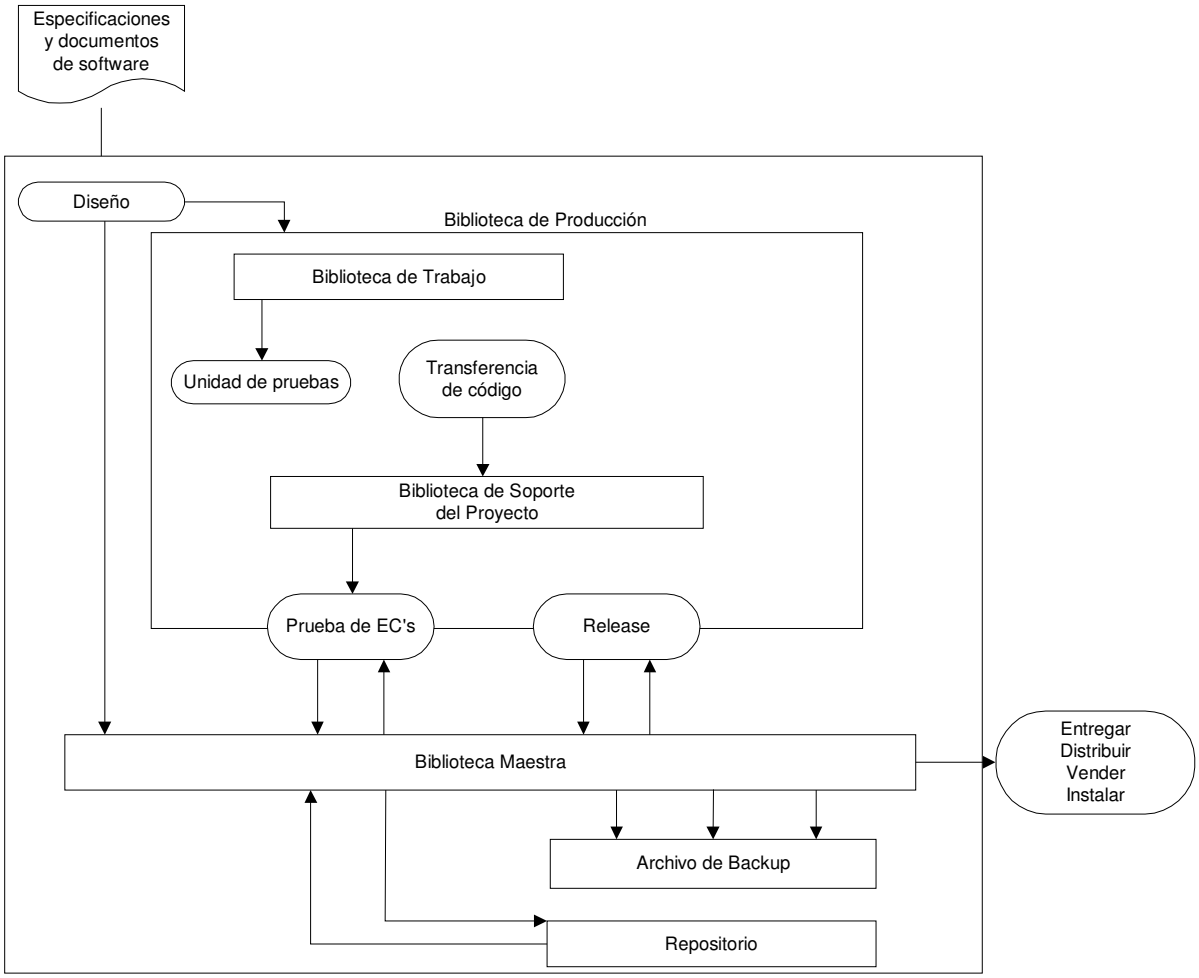
La biblioteca de *software* es uno de los puntos más fuertes del proceso de GCS, especialmente para las actividades de control y verificación del estado.
Una biblioteca de *software* es definida como una colección controlada de *software* y su respectiva documentación diseñada para ayudar en el desarrollo, uso o mantenimiento de *software*. Esta incluye la biblioteca maestra, la biblioteca de producción, biblioteca de desarrollo, repositorio de *software* y biblioteca de sistema.

La biblioteca de producción es descrita como la biblioteca de trabajo para la producción de código fuente y la biblioteca de soporte del proyecto -BSP- para el almacenamiento de código listo para ser integrado y probado.
La biblioteca maestra es descrita como la entidad que almacena código aprobado o liberado y también para la documentación aprobada y liberada que va a ser presentada al cliente. El repositorio de *software* es la entidad que archiva *software* y su documentación para un proyecto que ya fue terminado. Una parte del repositorio puede ser definido para almacenar temporalmente una copia del *software* y documentos, lista para ser usada en caso se destruyan las bibliotecas maestra y de producción. Otro segmento del repositorio puede ser usado para almacenar *software* diseñado para ser reutilizado por otros proyectos que se trabajan en paralelo con el proyecto actual.

1.5.1 Las funciones de la biblioteca de producción

La siguiente figura muestra un modelo básico de biblioteca de *software*

Figura 1: Modelo de Biblioteca de *Software*



1.5.1.1 Biblioteca de trabajo

La biblioteca de trabajo, parte de la biblioteca de producción, es conocida como el archivo de desarrollo o el área de trabajo de los programadores y es establecida al inicio del proyecto para permitir a los programadores tomar su área asignada para el desarrollo de código fuente como fue asignada por el líder del proyecto. Los programadores desarrollarán la función de codificación, corrección de defectos y pruebas en su área asignada. Una vez que un programador ha definido que las pruebas fueron aprobadas completamente y que el código ha sido revisado y aceptado, la GCS debe iniciar la transferencia del código de la biblioteca de trabajo a la biblioteca de soporte del proyecto -BSP- para pruebas más avanzadas. De todas formas, el programador debe quedarse con una copia para futuros cambios o mejoras. En este caso se hace una anotación en la BSP, porque la copia que se queda el programador no está bajo control, y no debe ser aceptada en el sistema a menos que un cambio sea solicitado y aprobado. Así la nueva versión se sobrescribe en la anterior.

1.5.1.2 Biblioteca de soporte del proyecto

La biblioteca de soporte del proyecto -BSP- es el lugar de almacenamiento de aquellos elementos de *software* transferidos desde la biblioteca de trabajo junto con la documentación de diseño detallada usada para producir ese código. Al momento de la transferencia, el código y la documentación son considerados bajo control de configuración y cualquier cambio solicitado es procesado por los procedimientos establecidos en la GCS. La regla debería ser que los cambios no pueden ser hechos arbitrariamente para los elementos del *software* que se encuentran en la BSP.

La razón de esto es que muchas veces una unidad ya transferida, por ejemplo, en la noche, es tomada a la mañana siguiente por el equipo de pruebas para integración con otra unidad. Si fueron hechos cambios después de que esto ocurrió y los cambios no fueron notificados al equipo de pruebas, los resultados de las pruebas podrían ser inválidos y causar otros problemas durante la secuencia de las pruebas.

Una ventaja de tener esta biblioteca intermedia es que la biblioteca maestra se puede mantener solo con entidades aprobadas y aceptadas. Así, el proceso de GCS nos asegura que únicamente documentación y código aprobado van a ser entregados a quienes lo soliciten y la entrega al cliente va a ser válida y confiable.

1.5.1.3 Biblioteca maestra

La biblioteca maestra -BM- es el punto de almacenamiento de todas las unidades, componentes, EC's y documentación que ha sido probada y revisada rigurosamente y que ha sido trasladada a la BM bajo un control de configuración más fuerte que cuando se encontraban en la biblioteca de soporte del proyecto. El traslado hacia la BM se da con el aviso de que se ha logrado un nuevo release del *software*, el cual describe cada elemento de la *baseline*, su nomenclatura y el identificador de versión. El aviso del nuevo release informa a los miembros del proyecto que los nuevos elementos están listos para ser integrados con la parte de *hardware* del sistema o para ser entregado al cliente como un producto de calidad.

El control de acceso es bastante estricto para la BM. Normalmente, sólo el bibliotecario tiene permiso para leer y escribir. Puede haber casos en los que la BM sea una computadora de escritorio específicamente para esta función o algún otro tipo de computadora colocada en un área donde sea requerido un acceso especial.

1.5.1.4 Repositorio de *software*

El diccionario define repositorio como un lugar para almacenar cosas, por ejemplo *software*, sería un lugar seguro donde se almacene *software*. El término archivar es frecuentemente usado para referirse al almacenamiento de *software* y su respectiva documentación. El requisito para archivar en un repositorio es que se pueda llevar registro de lo que contiene y que pueda ser extraído en caso fuera necesario. Archivar *software* en un repositorio debería ser cuidadosamente planeado para que en las actividades de la GCS sea posible recuperar *software* que sea necesario para otros proyectos.

1.5.1.5 Biblioteca de *backup*

La biblioteca de backup contiene duplicados de las versiones de *software* y su documentación al momento que se hicieron las copias desde las bibliotecas activas. Esta acción puede ser cada noche, semanalmente o mensualmente dependiendo de qué tan críticos sean los cambios en el contenido de las bibliotecas activas y el grado de confiabilidad del *software* para no causar pérdida de archivos importantes.

Muchos, en el trabajo de producir *software*, han sufrido la pérdida de archivos debido a falla del equipo, fallas en la energía eléctrica o por presionar el botón de Salir en el momento equivocado. El backup de ese archivo puede ser recuperado y así salvarnos el día.

Los discos o cintas de la biblioteca de backup también son almacenados en un lugar diferente para protegerlos de algún desastre como por ejemplo un incendio. Estos dispositivos pueden ser enviados muchos kilómetros lejos del repositorio de la biblioteca de backup.

1.5.2 Estableciendo las bibliotecas

La base de las bibliotecas de un proyecto debe ser planeada al inicio, si no se hace en la etapa de planeación se hacen previo al inicio del proyecto. En algunas ocasiones, el cliente puede preguntar por la descripción de la biblioteca de desarrollo de *software* "a ser usada para controlar el *software* y su documentación... incluyendo los procedimientos y métodos para establecer e implementar la biblioteca y los procedimientos de control de acceso para los datos almacenados en ella". Y otras veces, los desarrolladores se pueden dar cuenta que una biblioteca bien establecida y controlada puede convencer al cliente de que contrate sus servicios.

La biblioteca maestra es la primera en ser establecida para que al momento de tener *software* y documentación aprobados sea puesta bajo el control de la configuración. Normalmente lo primero en ingresar a la biblioteca maestra es la documentación. Las especificaciones de requerimientos del *software*, especificación de requerimientos de interface, documentos de diseño y planes de prueba deben ir en la biblioteca maestra.

La biblioteca de trabajo debe ser establecida también desde el inicio. La biblioteca de soporte del proyecto se activa en el momento que se comienza la codificación y pruebas unitarias de los elementos de configuración -EC's-, que usualmente se da cuando se termina de revisar un diseño y significa que el diseño está lo suficientemente detallado para empezar la codificación. Entonces, la biblioteca de soporte está lista para recibir transferencias de *software* para pruebas posteriores e integración a bajo nivel -unidades integradas en componentes y componentes en EC's-.

El segmento para archivar puede ser establecido en cualquier momento. Normalmente no es necesario sino hasta el final del proyecto, pero ha habido ocasiones en las que componentes de proyectos pequeños estaban desbordando los discos y era necesario que se archivaran. La biblioteca de backup debería estar lista para cuando se empiece la producción de *software*. Lo que se encuentra en el backup no debe ser considerado como un archivo. Aunque haya control de acceso, no habría gestión de configuración.

1.5.3 Responsabilidades

La actividad de bibliotecario es normalmente asignada a uno o varios miembros del equipo de desarrollo. El bibliotecario asume la responsabilidad de capturar, almacenar y mantener el código generado por los programadores. Así el *software* se encuentra almacenado, controlado y registrado entre cada ciclo de construcción y al final van a representar el sistema completo. Casi toda la responsabilidad de la biblioteca debe caer sobre el jefe de proyecto. Esto se hace importante cuando se aplican controles de acceso y alguien debe determinar quién puede tener permiso de lectura y escritura en la biblioteca maestra.

La experiencia indica que tanto el bibliotecario como el jefe de proyecto deben permiso de acceso a todos los archivos de la biblioteca maestra. Lo importante aquí es que las bibliotecas contienen el corazón de la compañía, organización o grupo, y los datos y código deben ser protegidos por individuos responsables y con la autoridad para hacerlo.

1.6 Herramientas de *software*

Las herramientas GCS fueron desarrolladas para ayudar a automatizar el trabajo de los bibliotecarios. Las principales capacidades de control de versiones que ofrecían estas herramientas fueron:

- Mantener una biblioteca o *repositorio* de archivos
- Crear y almacenar múltiples versiones de archivos
- Proveer un mecanismo de bloqueo para modificaciones simultáneas
- Identificar colecciones de versiones de un archivo
- Extraer y devolver versiones de archivos del repositorio

Las herramientas GCS proveyeron estas capacidades básicas del control de versiones y automatizaron el trabajo manual de los bibliotecarios. Un desarrollador podía extraer un archivo sin la intervención del bibliotecario. Mientras el archivo era revisado, nadie más lo podía modificar. Cuando el desarrollador terminaba, el archivo era ingresado y una nueva versión de él era creada automáticamente. En la actualidad, el modelo de extraer / ingresar permanece sin cambios.

Una de las primeras herramientas de GCS muy usada fue el 'Sistema de Control de Código Fuente' -SCCS*, source code control system*-, creado por los laboratorios Bell a principios de 1970. Una alternativa para SCCS fue el 'Sistema de Control de Revisiones' -RCS, *revision control system*-, creado por Walter Tichy de la Universidad Purdue. Ambos RCS y SCCS se convirtieron en las herramientas principales de GCS para la plataforma Unix. Muchas de las máquinas mainframe tenían su propia herramienta de GCS. Por ejemplo, el 'Sistema de Gestión de Configuración' -CMS, *configuration management system*-, fue parte del sistema operativo VAX/VMS de Digital Equipment Corporation -DEC-.

Estas primeras herramientas de control de versiones usualmente ofrecían una manera de etiquetar o marcar una versión particular de un archivo para un conjunto de archivos. Estas herramientas eran más efectivas que la forma manual. Ofrecieron las clásicas capacidades de GCS de identificar componentes de un sistema, controlar el cambio sobre esos componentes y tener una auditoría de quién modificó cuál archivo y cuándo.

El desarrollo de proyectos de *software* continuamente crece en complejidad y tamaño, por eso requirieron equipos de trabajo más grandes y más coordinación. Las herramientas modernas extienden la funcionalidad básica de las primeras herramientas de GCS soportando de mejor manera el desarrollo en paralelo, administración de espacio de trabajo y administración de las construcciones y releases de *software*.

Hoy en día, capacidades avanzadas son añadidas a diferentes herramientas de GCS. Estas herramientas avanzadas tienen objetos y opciones que proveen un soporte para GCS a un nivel de abstracción cercano a otros aspectos de la ingeniería de *software*, como la 'Administración de Cambios de Solicitudes' y 'Administración de Proyectos'. Estas avanzadas herramientas de GCS soportan versionamiento de todos los artefactos del proyecto -no sólo el código fuente-, proyectos de *software* en los cuales los miembros del equipo pueden no estar trabajando en el mismo sitio, desarrollo basado en componentes y Gestión de Configuración basada en actividades.

2. ACTIVIDADES DE LA GESTIÓN DE CONFIGURACIÓN

Este capítulo describe las actividades de la gestión de configuración. Son seis actividades las que se llevan a cabo:

1. Identificación: trata la jerarquía de *software*, la selección de los elementos de configuración, la definición del diseño y sus relaciones, numeración y el desarrollo de *baselines*.
2. Control: trata los procesos de cambios en el proyecto, también el control en los cambios pedidos por el cliente. También la forma de dar a conocer los cambios a todos los miembros del equipo.
3. Verificación de estado: describe cómo verificar los documentos planificados y producidos, cambios realizados, aprobados e incorporados a los documentos.
4. Auditoría: determina si lo que fue desarrollado, probado y entregado fue lo que se entregó. Resalta la importancia de una adecuada documentación de los requerimientos de *software*.
5. Control de Interface: describe la importancia del control de interfaces y la documentación asociada.
6. Subcontratados: los subcontratados con requeridos para asistir o contribuir el desarrollo de un producto de *software*.

2.1 La actividad de identificación

Varias tareas están asociadas a la actividad de identificación, incluyendo el desarrollo de la jerarquía de *software* propuesta por los ingenieros de *software*, la selección de los Elementos de Configuración -EC's- a ser desarrollados y descritos a varios niveles en la documentación como en la especificación del sistema, especificación de requerimientos y otros. Una vez los EC's han sido seleccionados, tareas adicionales son agregadas a la ingeniería de *software* y a la Gestión de Configuración. Esto incluye desarrollar y entregar la versión final de la jerarquía de *software*, establecer un esquema numérico, autorizar las especificaciones de *software* requeridas junto con los documentos para la definición de una *baseline*, y transferir o liberar la documentación aprobada y el código relacionado.

2.1.1 Creación de la jerarquía de *software*

La tarea de crear la jerarquía de *software* es importante en la Gestión de Configuración porque provee la primera vista estructural del sistema y sus elementos. Muchas veces, por la prisa de iniciar el proyecto, la jerarquía no es creada sino hasta que el proyecto está ya avanzado. La jerarquía permite a la Gestión de Configuración preasignar los números a los documentos, dar seguimiento al progreso de cada bloque conforme se va completando y finalmente estimar el personal y recursos que serán requeridos para el proyecto. También de la estructura inicial se eligen los EC's.

2.1.2 Selección de elementos de configuración del *software*

La selección de los EC's es más importante para la toma de decisiones en la administración del proyecto, y en varias ocasiones, en la compañía. Especialmente si es el único proyecto en el que la compañía está trabajando.
La selección de muchos EC's creará un número innecesario de documentos y especificaciones. Y si se seleccionan muy pocos, no proveerán suficiente visibilidad para abarcar todos los requerimientos de diseño y desarrollo. La adecuada selección de EC's puede simplificar el esfuerzo de los ingenieros de *software*, del control de calidad y la Gestión de Configuración en las fases de mantenimiento y de soporte al concentrar funcionalidades similares dentro de un EC y así incrementar las posibilidades de que la actividad de cambios al sistema.

Un EC es una agregación al *software* que satisface una función específica, quiere decir que es un elemento que funciona, se prueba y se utiliza independientemente. Esta definición da varios criterios que se deben tomar en cuenta en el proceso de selección.

Algunos de los criterios que se aplican en la selección de EC's son:

- Uso múltiple: ¿será el elemento utilizado en otros elementos del mismo nivel ó nivel superior?
- Crítico: ¿podría, si falla, afectar la seguridad o confiabilidad?.
- ¿Tiene impacto financiero?
- Mantenimiento: ¿el mantenimiento lo harán diferentes grupos?
- ¿Será reutilizado o diseñado para serlo?
- ¿Está relacionado a otro EC existente?

Si la respuesta a estas preguntas es “sí”, entonces el elemento es designado como un EC. De lo contrario, el equipo que está haciendo la selección debe decidir si lo toma o no. Una vez un elemento de *software* es designado para ser elemento de configuración, requiere:

- Documentos y especificaciones separadas, planes de prueba y manuales de uso y mantenimiento.
- Aprobación del cliente.
- Verificación de estado detallado y un reporte.
- Que se pueda rastrear de arriba hacia abajo en la jerarquía de *software*.
- Revisiones de diseño individuales.
- Numeración, nomenclatura y marcas individuales.

Un EC no se identifica únicamente dándole un número, es definido por una descripción de sus atributos y su contribución a toda la estructura, incluyendo la asignación de un nombre y su texto de descripción que dice qué hace el EC, cómo lo hace, dónde lo hace, qué tan rápido, qué lo ejecuta y qué produce. Una descripción completa no dejará duda a nadie de la selección de ese EC.

2.1.3 Documentación de la definición de diseño

El esquema de documentos de Especificación de Requerimientos de la NASA consiste en las siguientes secciones:

- Introducción
- Documentación relacionada
- Requerimientos de Interface externa
- Especificación de requerimientos
- Requerimientos de procesos y datos
- Requerimientos de desempeño y calidad
- Requerimientos de seguridad
- Requerimientos de privacidad
- Restricciones de implementación
- Metas de diseño
- Rastreabilidad a diseños de nivel superior

La importancia de la Rastreabilidad de Requerimientos para la Gestión de Configuración se ve en tres capas. Provee la información necesaria de cómo el sistema va a ser desarrollado y en la actividad de control muestra cómo proceder, donde revisar y cómo encontrar los documentos que son afectados por un cambio. Permite a la Gestión de Configuración asegurar una identificación única, rastreabilidad hacia arriba y hacia abajo y permite dar mantenimiento al estado de los documentos en todo momento.

En este punto, el ingeniero de *software* desarrolla la especificación de requerimientos de *software* de donde se crean las especificaciones de diseño.

La Guía de Especificación de Requerimientos de *Software* de la IEEE da características para una buena especificación, estas son:

- Que no sean ambiguos
- Completos
- Verificables
- Consistentes
- Modificables
- Rastreables
- Usables -en la operación y mantenimiento-

Para el Documento de Diseño de la Arquitectura del *Software* se define su propósito como: registrar la información lógica y funcional del diseño de componente de *software*. Esto incluye la razón fundamental del diseño, la arquitectura seleccionada del componente incluyendo por lo menos un nivel de descomposición en los sub-componentes de *software* o elementos de diseño.

Este es el nivel preliminar del diseño del sistema dando la descripción del por qué del diseño y otras consideraciones. El esquema incluye las siguientes secciones:

- Descripción de la Arquitectura del sistema
- Diseño de Interfaces externas
- Asignación y rastreabilidad de los requerimientos
- Particionamiento para Desarrollo Incremental
- Cada una de estas secciones mejora la identificación de los EC's descritos en la especificación de requerimientos.

Los documentos de Diseño Detallado son críticos para el éxito de las fases de codificación y prueba. Además, muchos cambios son hechos a éstos documentos y por eso requerirán una inspección del codificador, la función de control de calidad y la Gestión de Configuración para poder entender su formato y contenido. Los esquemas incluyen estas secciones:

- Descripción detallada del diseño
- Rastreabilidad para el Diseño de la Arquitectura
- Diseño detallado de las Unidades de Compilación
- Diseño detallado de la Interface Externa
- Notas de Codificación e Implementación

Hasta aquí se han descrito los tres documentos de identificación principales: requerimientos, diseño del sistema -arquitectura- y diseño detallado. La culminación de éstos documentos de especificación es la especificación del producto o la descripción de cómo el desarrollador entiende los requerimientos para realizar el producto esperado. El esquema de documentos contiene las siguientes secciones:

- Documento de Concepto
- Especificación de Requerimientos
- Documentos de Diseño: Arquitectura y Detallado
- Documento de descripción de versión -cómo fue hecho-
- Manual de usuario -cómo usarlo-
- Manual de mantenimiento

La misión de la Gestión de Configuración será lograda si hay evidencia documentada de la actualización del producto de *software* a la versión que está siendo entregada.

Algunos documentos parecidos son muy importantes en la fase de identificación. Primero, deben estar claras las interfaces para describirlas y proveer la suficiente información para construirlas en la fase de codificación. Por esto, hay dos documentos principales: el de Especificación de Requerimientos de Interfaces, que describe las interfaces que son requeridas, y el Documento de Diseño de Interfaces, el cual describe cómo las interfaces son construidas para acoplarse al sistema. Con la generación de éstos documentos logramos una clasificación, catalogación y numeración de requerimientos que de otra forma serían un trabajo sin planeación enorme.

2.1.4 Relaciones de la definición de diseño

La tarea de catalogación envuelve la definición de cómo el EC y su respectiva documentación se relaciona con otros. Son cinco relaciones:

1. *Equivalencia*. Un EC puede ser almacenado como maestro en un disco o como copia en una unidad magnética u óptica. Debemos saber cuál es cuál y por qué está ahí. En otras palabras, si el EC no está en el disco maestro, no ha sido aprobado y debe estar en el área de trabajo.
2. *Dependencia*. Se deben definir las relaciones padre-hijo, por lo menos los de nivel inferior con su padre. Si esto no se hace, se asume que es un elemento de nivel superior.
3. *Derivación* se describe de dónde viene el código.
4. *Sucesor* se define rastreando el historial de cambios de un elemento de la revisión A a la revisión B.
5. *Alternativo* es similar a la variación en un tema. No tiene efecto en la forma o funcionamiento del elemento más que el alternativo A es más rápido que el B.

Las relaciones descritas muestran de dónde vienen los elementos de *software* y cómo se relacionan con el proyecto.

2.1.5 Numeración de los elementos de *software*

¿Por qué asignar números cuando se está trabajando en un proyecto pequeño con pocas líneas de código? ¿Por qué complicarse colocando números a una documentación de diseño muy pequeña?. La respuesta es que debe haber una forma de recuperar un conjunto de proyectos pequeños, capturados y almacenados en la biblioteca maestra de la compañía, de forma rápida y confidencial.

Hay dos tipos de sistema numeración: el *significativo* y el *no significativo*. Ambos son importantes, pero para determinar el mejor se debe evaluar la complejidad del propósito de los números. Un sistema de numeración no significativa es una sencilla asignación autoincremental, por ejemplo de 0001 a 1999. Este sistema requiere el establecimiento y mantenimiento de un registro de asignación, el cuál proveerá la información de cada documento con su número. La ventaja es que solamente se debe controlar el siguiente número. Las desventajas son que se debe tener un punto central de control y no se sabe dónde está el código, documentación o reporte con sólo ver el número.

El sistema significativo puede ser sencillo o muy complejo. Un sistema significativo describe el elemento que se identifica. Se pueden usar números de la siguiente forma:

Tabla I: Ejemplo de sistema significativo

AÑO	No. Día	NÚMERO
2000	135	012345
2000	138	012345
2000	140	123456

Las ventajas del sistema significativo es que no se debe llevar un registro de asignación ni tener un punto central para la distribución de números. Las instrucciones para construcción y aplicación de números son escritas una sola vez en la compañía. Alguien que lea la numeración solamente debe recordar qué quiere decir cada parte para identificar lo que lee. Las posibilidades de repetir un número son casi nulas, pero siempre debe haber un procedimiento de auditoría para descubrir si hay duplicados.

2.1.6 Desarrollo de las *baselines*

Una *baseline* es un punto de partida acordado, después del cual cualquier cambio debe ser comunicado a todos los miembros del equipo de *software* involucrados. Por ejemplo, si alguien acuerda que todos los miembros deben tener una reunión a determinada hora y esa hora cambia, alguien debe notificarle a los todos del cambio.

En la definición de un producto, una *baseline* es un EC o conjunto de EC's formalmente revisados, acordados o designados en cierto punto en el tiempo del ciclo de vida del proyecto. Lo que se designa para un proyecto A no debe ser igual para un proyecto B. Las *baselines* pueden marcar el final de una fase o segmento de la fase, ó el inicio de una nueva fase. En un sentido genérico hay cinco *baselines* principales para todo trabajo:

1. La *baseline* de *definición* es la culminación de la definición de requerimientos de *software*, la descomposición y la solución final de cómo se diseñará el *software* para cumplir los requerimientos.
2. La *baseline* de *diseño* es el punto en el que se ha determinado cómo se construirá.

3. En la *baseline* de *codificación y pruebas unitarias* se prueba que los programas escritos y probados de acuerdo a un plan de pruebas cumplen los requerimientos.
4. La *baseline* de pruebas es el punto donde se realizan las pruebas formales y se verifica que esté listo para entregarse al cliente.
5. En la *baseline* de *mantenimiento* es donde la funcionalidad y alcances de un producto entregado pueden modificarse, corregirse o mejorarse.

Las tres *baselines* formales que forman las bases para las genéricas son: funcional, de asignación y de producto. Una *baseline* intermedia ocurre entre las de asignación y de producto, que se conoce como Configuración de Desarrollo.

Estas *baselines* formales se definen como:

- La *baseline funcional* es la documentación inicial aprobada que describe las características de funcionalidad de los EC's y los criterios de evaluación para demostrar el logro de una característica de funcionalidad especificada: interfaces, requerimientos de base de datos y satisfacción del diseño de constraints.
- La *baseline* de *asignación* es la documentación inicial aprobada y designada a cargo del desarrollo de EC's que son parte de un EC de más alto nivel, esto es, la especificación del sistema, que contiene las características funcionales y de interface junto con los criterios de evaluación necesarios para demostrar el logro de los requerimientos de los EC's de nivel superior.
- La *baseline* de *producto* es la documentación inicial aprobada y el código fuente que define un EC durante las fases producción, operación y mantenimiento de su ciclo de vida.

Las *baselines* genéricas y formales pueden integrarse de la siguiente forma:

Tabla II: *Baselines* genéricas y formales

Genéricas	*Formales*
Definición	Funcional y de asignación
Diseño	De localización y Configuración de Desarrollo
Codificación y Pruebas unitarias	Configuración de Desarrollo
Pruebas	Configuración de Desarrollo
Mantenimiento	De producto

2.1.7 Configuración de desarrollo

El segmento de configuración de desarrollo puede ser dividido en subsegmentos. El primero marca el punto en la fase de desarrollo cuando los documentos de desempeño, diseño y otras especificaciones del *software* están en su estado inicial, esto es, aprobado para el proyecto pero en espera de una autorización formal del cliente y ya se ha colocado en la biblioteca maestra para su Control de Configuración.

El segundo subsegmento es el punto en la fase de desarrollo cuando los elementos de *software* han sido codificados y son trasladados del área de trabajo del programador a la biblioteca de soporte para el control de cambios.

El tercer subsegmento es el punto en la fase de desarrollo cuando los elementos de *software* son trasladados a la biblioteca maestra para el control de cambios. Este punto de control normalmente concluye con el establecmiento de la *baseline* de producto y la entrega del producto de *software* al cliente.

En la culminación de la integración y fase de pruebas del *software*, un reporte de las pruebas es generado para asegurar que las pruebas desarrolladas justifican la integración final del sistema. Para generar éste reporte, se debe evaluar el histrorial de tipos y cantidad de defectos que fueron encontrados y arreglados.

La configuración de desarrollo termina con las pruebas de rendimiento del *software*, pruebas del sistema completo y la integración del *software* con el hardware. Durane éste período, el documento de especificación del producto -cómo se hizo- es terminado, junto con el documento de descripción de la versión -qué contiene y qué hace-. Estos documentos, para cada EC y el reporte de pruebas formales son algunos de los elementos revisados en las auditorías físicas y funcionales. En el momento que sea aceptado por los equipos de auditoría del cliente y de los desarrolladores la *baseline* de producto es establecida.

2.2 La actividad de control

Tanto como el proyecto mantenga comunicación entre el equipo de desarrollo y y el cliente o comprador, habrán siempre formas o formatos para proponer cambios y los procedimientos para la preparación y aceptación de éstos cambios. Una vez se hayan establecido *baselines*, los cambios empiezan a ocurrir y deben ser esperados y planeados, porque nadie es perfecto y hasta las decisiones hechas concientemente pueden no resultar como se esperaban y entonces deben cambiar.

2.2.1 Control de las *baselines*

Como ya se vió, una *baseline* consiste en documentes aprobados y el código relacionado. Para mantener la integridad de estos elementos, la GCS debe proveer la adecuada protección del medio de almacenamiento, normalmente en la base de datos de una computadora. En todo momento debe haber control de acceso a las copias principales cuando se quiera agregar, borrar o cambiar un elemento del *software*. El equipo de *software* debe tener conciencia que las copias obtenidas de los archivos principales, representan las últimas revisiones o versiones.

Para mantener la integridad establecida de las *baselines*, se puede incorporar cambios a estas definiciones principales sólo desde documentos de cambios aprobados y reemplazarlos solo para una versión superior aprobada. Una verificación de la incorporación es también necesaria para garantizar que el documento es correcto y que ningún otro documento o código ha sido afectado.

El mantenimiento de las *baselines* debe tomar en cuenta también el uso de prototipos de *software* durante las fases tempranas del ciclo de vida. Prototipar quiere decir, realizar una prueba del producto generando código a partir de documentos que todavía no han sido aprobados, con el objetivo de probar la integridad del diseño para alguna teoría o descripción dada.
Los prototipos pueden también ser usados para probar la validez de un cambio propuesto. Muchos cambios en el inicio del ciclo de vida, pueden indicar que la documentación que se prepara es inadecuada y el proceso de revisión está fallando en encontrar los erroes en los documentos. La necesidad de revisión o corrección de errores es el resultado de una pobre preparación de los cambios en los documentos.

2.2.2 Procesando cambios internos

No hay un estándar para el procesamiento de cambios internos o de un tipo de proyecto específico, ningún procedimiento se ha descrito. Sin embargo, hay una serie de guías que se han publicado, como las de la IEEE: Proceso de Ciclo de Vida del *Software* y la Guía para la Gestión de Configuración del *Software*.

Algunas veces la necesidad de cambio es simplemente obvia, el sistema no funciona como se requiere. A menudo el sistema trabaja inconsistentemente y no se sabe si efectivamente existe un problema, ni mucho menos dónde encontrarlo. Muchas veces, conforme el sistema se usa, nuevas funcionalidades son evidentes. El proceso de cambio debe asegurar la modificación de los documentos para todos estos casos. Cada solicitud de cambio debe ser tomada en consideración y el solicitante debe ser tratado con respeto.

Un problema mayor es si la corrección va a ser por un error, falla o defecto en el producto o para mejorar. Los ingenieros tratarán que los cambios sean clasificados como mejoras debido a que los defectos normalmente reflejan un mal trabajo de los ingenieros y programadores. Los clientes los clasificarán como correcciones de defectos porque así pueden tomarlos como parte de la garantía que se haya ofrecido. De todas formas, al no haber un acuerdo sobre el manejo de los cambios, el cliente siempre deberá pagar por éstos. En estos casos, una especificación de requermientos completa y acuerdos realizados ayudarán a resolver éste dilema.

Por esto es que la Gestión de Configuración debe controlar los requerimientos y ser capaz de rastrearlos hasta la parte defectuosa, así como la parte defectuosa hacia un requerimiento. En ciertos casos, un requerimiento puede estar defectuoso, por ejemplo si dice que “La temperatura debe mostrarse en grados centígrados”, cuando debería mostrarlos en grados Farenheit.

Un pequeño número de cambios caerán en la categoría de ‘imposible de duplicar’. El ingeniero de sotware dirá que el usuario comete el error, pero el usuario dirá que el sistema es defectuoso. 80% del tiempo de los ingenieros de soporte es utilizado en análisis e investigación de un número pequeño de cambios reportados. Buenos casos de prueba permiten a los ingenieros y usuarios trabajar más cercanamente y resolver éstos tipos de defectos con mayor rapidez.

La Gestión de Configuración implementa y desarrolla el proceso de control de los cambios de acuerdo al Plan de gestión de configuración o, si no ha sido escrito, con los procedimientos internos. El bibliotecario de *software* provee la técnica apropiada para asegurar calidad en el proceso de incorporación de cambios al código o documentación.

En general, todos los integrantes de un proyecto son responsables de comunicar los cambios que han ocurrido, que han sido procesados y que han sido incorporados. Algunos de los requerimientos generales para control incluyen describir el procedimiento de cambios a seguir en el plan de desarrollo de *software* y en el plan de GCS. Las directivas del programa de desarrollo deben ser escritas para los involucrados en el proyecto, pero es importante que estén de acuerdo a los requerimientos del cliente, a las directivas y prácticas de la compañía y procedimientos de GCS de la compañía.

Un requerimiento muy importante es que el control debe ser aplicado también a *software* que es interno, como herramientas de soporte o editores. Esto asegura que *software* desarrollado para la misma compañía no es cambiado sin que se comunique a todos los usuarios y que no hay ningún cambio que no esté bajo la actividad de control de cambios.

Un último requerimiento es que un cambio propuesto, que se aprueba internamente, no puede ser incorporado sin la aprobación del cliente o sin haber llegado a un acuerdo entre ambas partes.

2.2.3 Propuesta de cambios al cliente

Normalmente, cuando ordenamos un producto a la medida, como una casa o un pantalón, tenemos el derecho de inspeccionar el progreso, hacer cambios, consultar sobre posibles cambios e incluso de rechazar el producto si no nos satisface. Este proceso ocurre también cuando un producto de *software* es desarrollado para el mercado, un cliente en particular o para nuestra propia compañía.

Para asegurar que el cliente quedará satisfecho, el equipo del proyecto debe establecer un proceso para que de forma eficiente y efectiva haya comunicación con el cliente para discutir los cambios. Muchas veces el acuerdo a que se llega es que el cliente pagará por los cambios que vayan más allá de lo estimado en el contrato inicial. La regla general es: si un cambio afecta la funcionalidad de un elemento, debe ser aprobado por el cliente antes de ser incorporado.

También hay veces en que el cliente requiere que todos los cambios sean aprobados. Esto ocurre a menudo cuando el proyecto no está yendo bien, ha habido muchos cambios o la calidad es muy baja.

2.3 Actividad de verificación de estado

Es la actividad de registro, da seguimiento a la identificación de documentos de la configuración actual, la configuración actual del *software* entregado, el estado de los cambios que se están revisando y el estado de la implementación de cambios aprobados.

La verificación del estado es información. Es también parte de la toma de decisiones en el proyecto de *software* que determinan cómo se está llevando el proyecto, cuáles son los problemas mayores y de dónde se originan. De los reportes y consultas que se crean, el jefe del proyecto puede determinar cómo el proyecto se está llevando en relación a la documentación requerida, las *baselines* establecidas, cambios y la clasificación de los cambios -cuántos cambios por día, semana, etc.-. Se puede ver también los tipos de cambios, causas, costos y la calendarización y progreso de los cambios.

Una razón más importante para llevar los registros de la verificación del estado y los reportes, es que se utilizan en el mantenimiento del *software*. Es importante reconocer que el mantenimiento empieza cuando se tiene que realizar el primer cambio. Esto normalmente ocurre durante las primeras etapas de pruebas y continúa en toda la vida del producto. Antes de hacer correcciones, mejoras o modificaciones al *software* existente, se debe referir a la historia del componente para información que dice qué cambios ocurrieron en el pasado, qué problemas se encontraron durante el desarrollo, pruebas e integración.

Los registros históricos en un sistema de verificación de estado son también usados para analizar y mejorar el proceso de desarrollo de *software*. El número de cambios procesados, el tiempo que tomó analizarlos y el tiempo para corregirlos pueden decir mucho acerca de qué tan eficiente fue el proyecto y qué tan apegado estuvo a la calendarización y los costos estimados. Estas estadísticas pueden servir como bases para futuras estimaciones de costos. Y más importante, se pueden derivar mejoras en el proceso que llevarían a la reducción de costos y mayor competitividad en el mercado.

Otra razón igual de importante para la verificación del estado es poder mantener al personal del proyecto informado de las características de las versiónes actuales en desarrollo, el *software* bajo pruebas y el *software* en espera de aceptación final.

2.3.1 La plataforma de verificación del estado

La plataforma de verificación del estado se ha conocido como el Archivo de Desarrollo de *Software*. Al principio, era conocida como el Folder de Desarrollo de *Software* porque se basaba en el uso de folders de manila. La intención del folder de desarrollo de *software* era contener todo lo que se quisiera saber acerca de una unidad de *software* desde el tiempo que fue concebida hasta que fue aceptada, integrada y entregada. El contenido principal del archivo incluye:

- La especificación de requerimientos inicial
- El diseño de requerimientos resultante
- El código subsiguiente
- Otros datos como las interfaces o información de la base de datos
- Todos los datos de prueba y reportes
- Copia de los cambios desde el momento que se solicitadon hasta que se incorporaron

En un ambiente automatizado, el contenido del archivo de desarrollo es el mismo, pero la referencia a datos necesarios, datos sobre los cambios y datos sobre las pruebas se hace con mayor rapidez. Un sistema de verificación del estado solamente es útil si la información que posee también lo es.

2.4 Actividad de auditoria

Ya que se ha pasado por las actividades anteriores, se debe establecer si el producto de *software* ha sido construido de acuerdo a los requerimientos y que el *software* es verdaderamente representado en la documentación creada en la fase operacional. Esta es la fase crítica del *software*, sólo si trabaja como se había pensado y definido se puede considerar un éxito. Por esto, es importante que revisiones formales sean desarrolladas en el *software* y documentación antes de pasar a la fase operacional.

Las auditorias de configuración proveen este análisis. No son lo mismo que auditorías para calidad, en cambio, una auditoría de configuración del *software* desarrollado asegura que lo que fue requerido fue lo que se construyó. Las actividades para calidad tienen otros criterios de desarrollo para determinar la integridad del producto. Hay dos tipos de auditorías: funcionales y físicas. Las dos son prerrequisitos para establecer la *baseline* del producto una vez la documentación ha sido autenticada por el cliente.

2.4.1 La auditoria funcional

El objetivo de la auditoría funcional es verificar que la funcionalidad actual de los EC's va de acuerdo con los requerimientos planteados en la Especificación de Requerimientos y la Especificación de Interfaces.

El procedimiento para determinar la aceptación es revisar los resultados de las pruebas y compararlos con los datos de aceptación de los requerimientos.

Para sistemas más complejos, la auditoría funcional debe ser incremental, llevándola por cada segmento del EC a ser completado. Un resumen de la auditoría confirmará que todos los segmentos fueron revisados, todas las interfaces fueron verificadas y que todos los elementos han sido completados.

Para preparar y realizar una Auditoría Funcional, el encargado debe cumplir las siguientes tareas:

- Seleccionar el personal que participará en la auditoría
- Identificar y describir los EC's a ser auditados en determinada sesión
- Preparar una agenda
- Dar una copia de la agenda a cada participante, los requerimientos e interfaces a ser auditadas y el listado de código fuente.

Normalmente las auditorías comienzan con una descripción de:

- los métodos de prueba utilizados
- cómo se realizaron las pruebas y los reportes
- quiénes presenciaron las pruebas. El encargado también debe describir las pruebas que no cumplieron los requerimientos, la razón o posibles razones por las que no se cumplieron y las soluciones sugeridas.

2.4.2 La auditoria física

El objetivo de la auditoría física es determinar si el diseño, especificación y documentación representan al *software* que fue codificado y probado para determinados EC's. El procedimiento es examinar la documentación de diseño con el código fuente de los EC's y los manuales del *software* con los que será entregado. Además, los documentos de especificación del *software* y descripción de versión deben ser aprobados por el cliente antes de la auditoría. Normalmente la auditoría física viene después de completar satisfactoriamente una auditoría funcional y será realizada tan pronto como se sepa que todo el *software* relacionado a un EC ha terminado de la auditoría funcional.

En el momento que una auditoría va a iniciar, el encargado informará sobre cualquier problema que haya con el EC y la solución o posibles soluciones al problema, y el estado de este problema. Después que se haya informado, se desarrollan las siguientes tareas:

- Llevar a cabo una completa revisión de la Especificación del *Software* para ver que esté completo
- Revisar en todas las descripciones de diseño los símbolos, etiquetas, referencias y descripciones de datos.
- Comparar las descripciones de diseño de componentes de primer nivel con las descripciones de unidades de bajo nivel para ver la continuidad y el flujo de información.
- Revisar el código fuente para ver que cumple con los estándares de codificación.
- Asegurar que el código fuente utilizado produce los ejecutables aceptados para la auditoría funcional.

Estas tareas proveen la primera evidencia de que hay una revisión formal de los manuales del *software*. Es importante para la función de soporte del *software* saber que los manuales que se entregaron van de acuerdo al *software* desarrollado. Es más importante que los clientes entiendan los manuales y puedan realizar las tareas necesarias para operar el sistema. Los manuales deben ser considerados tan importantes como las especificaciones del *software*. Además, todo el equipo del proyecto debe conocer las leyes y reglas para la protección de derechos de autor. Deben ser determinados los derechos que se desean para el *software* y su documentación.

2.4.3 El rol de la Gestión de Configuración

Estas son las tareas desarrolladas por la gestión de configuración para cualquiera de las auditorías que se vieron:

- Revisar los requerimientos de auditorias del cliente
- Advertir al equipo del proyecto sobre qué datos y participantes son requeridos y cuándo
- Logística para los clientes y desarrolladores que participarán y los datos de cada auditoría
- Tener una presentación final de los documentos de especificación para que sean revisados por el cliente antes de la auditoría.
- Asegurar que todos los datos para la auditoría están disponibles

La Gestión de Configuración debe saber cuántas auditorías se llevarán a cabo y cuándo para poder determinar los recursos necesarios.

2.5 Actividad de control de interface

El control de interface debe ser parte permantente de la definición de GCS. Interface son las características funcionales y fisicas requeridas para un límite común que existe entre dos o más productos de *software* y sistemas de computadoras que son provistas por diferentes organizaciones. Un documento de control de interface define las interfaces que pueden afectar la operación de EC's que funcionan junto.

Mientras que la definición de interfaces y los requerimientos de interface son actividad de la ingeniería del sistema, la responsabilidad de identificar las interfaces definidas, el control, el estado y desarrollo de auditorías de configuración resta en la GCS.

La actividad de interface también es responsable del control de la documentación que pueda generar un cambio que afecte a las interfaces. Un ejemplo del contenido de un Documento de Control de Interface es el siguiente:

- Datos: entradas, salidas
- Mensajes: formato, contenido, almacenamiento
- Protocolos: procesos, detección de errores y recuperación
- Interface: diagramas, estándares y convenciones

2.6 Actividad de control de subcontratados

Esta es una actividad importante de la GCS, que muchas veces no es llevada a cabo cuando se solicitan subcontratados. Muchas veces un administrador de subcontratados no proveerá las cláusulas y requerimientos apropiados para asegurar que los requerimientos para GCS son comunicados al subcontratado y éstos son capaces de lograr el objetivo y controlar la magnitud del trabajo a desarrollar.

Hay tres categorías principales y tres secundarias de subcontratos en las que la Gestión de Configuración tiene un rol importante.
Las tres categorías principales son determinadas por el grado de responsabilidad dado al subcontratado:

(1) en la primera categoría total autoridad es dada para el diseño, desarrollo, construcción, pruebas y entrega de un EC o varios.
(2) en la segunda categoría la responsabilidad es limitada en el diseño, como cuando sólo se modificará *software* ya existente o se realizará la etapa de pruebas.
(3) en la tercera categoría no se da ninguna responsabilidad al diseño.

Las categorías secundarias incluyen:

(1) la asociación de los contratados con la compañía.
(2) miembros del equipo de desarrollo que trabajen directamente con los subcontratados.
(3) trabajar con terceras compañías, no definidas en un contrato formal como los subcontratados directos.

Para que los subcontratados entiendan lo que se les está requiriendo, la Gestión de Configuración debe preparar una sección específica para los subcontratados, describiendo los requierimientos.

Esta actividad de Gestión de Configuración también debe proteger cualquier información recibida de un subcontratado y colocarla bajo un control de configuración. Además, toda información recibida de un subcontratado debe ser integrada para que al momento de distribuirla de nuevo, todos reciban la versión correcta.

3. LA SOLUCIÓN GCS DE *RATIONAL*: UNIFIED CHANGE MANAGEMENT -UCM-

Rational ofrece el proceso Unificado de Gestión de Cambios -UCM, *Unified Change Management*-, una solución de GCS fiable y comprensible que integra las características de dos herramientas: *Rational* Clear Case para Administración de Artefactos de *Software* -AAS- y *Rational ClearQuest* para seguimiento de cambios y defectos. Juntas, estas herramientas representan un sistema de GCS que proveen una solución que acelera los ciclos de desarrollo en el proceso. Proveen flexibilidad al proceso de configuración que maneja los cambios en el desarrollo de *software*. Cuando se utilizan juntas, estas dos herramientas simplifican el proceso de realizar cambios identificando las asociaciones entre los cambios requeridos, ya sea para mejorar o para corregir, con el código fuente que se modificará.

Unified Change Management -UCM- provee un proceso predefinido que organiza el trabajo a través de actividades y artefactos. Acelera el proceso de desarrollo ajustándose a las necesidades propias del equipo de desarrollo de *software*.

3.1 Actividades y artefactos

Cuando los equipos de desarrollo crecieron en tamaño y complejidad, se volvió muy importante que los equipos organizaran lógicamente las actividades para cada release y manejaran de forma eficiente los artefactos para implementar estas actividades. Una actividad puede involucrar corregir un defecto o construir una mejora para un producto existente.

Un artefacto es cualquier cosa que se produce y evoluciona en el transcurso del proyecto, como un documento de requerimientos, código fuente, modelo de diseño o un script de prueba. Los equipos realizan actividades y producen artefactos. UCM integra la administración de actividades de *Rational ClearQuest* con la administración de artefactos de *Rational ClearCase*.

3.1.1 Administración de actividades

En UCM, los cambios son vistos como actividades. *Rational ClearQuest* provee un flujo de trabajo ajustable, para el seguimiento y administración de actividades. Esto permite a los equipos:

- Asignar actividades a cada individuo
- Identificar prioridades, estado y otra información de las actividades
- Generar reportes y diagramas automáticamente.

El flujo de trabajo de *ClearQuest* puede ser adaptado de acuerdo a los requerimientos del equipo. Los equipos que no tengan un proceso definido pueden usar el que ya trae *ClearQuest*, sus formularios y las reglas asociadas para acelerar la implementación.

Un flujo de trabajo para administrar y rastrear los defectos y otros cambios a través del proceso de desarrollo es esencial para lograr los estándares de alta calidad de hoy en día. UCM aumenta el nivel de abstracción de estos cambios para visualizarlos como actividades.

3.1.2 Administración de artefactos

Rational ClearCase provee una infraestructura para la Administración de Artefactos de *Software* que los equipos pueden utilizar para administrar todos sus artefactos a lo largo del ciclo de vida del proyecto. *ClearCase* ofrece:

- Seguridad en el almacenamiento digital de artefactos y versionamiento.
- Permite compartir código automáticamente.
- Administración de espacios de trabajo para la correcta selección de las versiones correctas de los artefactos.
- Completa escalabilidad desde pequeños grupos de desarrollo hasta grandes equipos distribuidos en locaciones diferentes.

Además, provee una infraestructura flexible para la implementación de GCS, los equipos pueden personalizar sus procesos de GCS. Por ejemplo, para realizar las auditorias y mantenimiento de requerimientos, los equipos pueden adjuntar comentarios. Pueden crear políticas que administren el uso de estos comentarios para asegurar que existen, cumplen los estándares y son interpretados adecuadamente. Con *ClearCase*, las organizaciones se benefician de la flexibilidad de usar diferentes políticas para diferentes equipos o proyectos. Con UCM, tienen un proceso predefinido que automáticamente implementa estas políticas de proyectos basados en procesos de desarrollo probados y exitosos.

3.1.3 Las cinco áreas del proceso

UCM provee procesos predefinidos para cinco áreas específicas:

1. Aislamiento de desarrolladores, permitiendo compartir artefactos de código común.
2. Unir grupos de artefactos relacionados que son desarrollados, integrados y liberados juntos en componentes de UCM.
3. Organización de artefactos en *baselines* de componentes y promoverlas como *grupos completos* de acuerdo a los estándares de calidad establecidos.
4. Organización de cambios en *grupos de cambios*.
5. Administración e integración de actividades y artefactos.

3.2 Artefactos a lo largo del ciclo de vida

Los beneficios de UCM se aprecian más cuando es utilizado para administrar cambios por miembros del equipo que no necesariamente son desarrolladores. Estos incluyen analistas, diseñadores y probadores quienes producen artefactos relevantes para sus áreas de trabajo. Por ejemplo, analistas producen documentos de requerimientos, los diseñadores modelos de diseño y los probadores los scripts de prueba, datos de prueba y resultados.

Para comunicar y coordinar el trabajo eficientemente, los miembros del equipo necesitan poder compartir estos artefactos tan eficientemente como se comparte el código fuente. UCM provee el proceso para administrar la información compartida para todos los tipos de artefactos.

3.2.1 Artefactos del análisis

El análisis desempeña la importante tarea de definir el alcance del proyecto, que requiere determinar lo que la solución hará y sus límites. Durante el análisis, estos profesionales crean una variedad de diferentes artefactos que ayudan a definir la solución propuesta incluyendo documentos de requerimientos y modelos visuales. Es esencial que los equipos continúen utilizando estos artefactos para administrar los cambios a través del proceso de desarrollo.

Los proyectos exitosos cuentan con una apropiada administración de requerimientos. Una administración efectiva también involucra el seguimiento adecuado a los cambios en los requerimientos.

3.2.2 Artefactos del diseño

Los diseñadores modelan la arquitectura del sistema y definen la forma del sistema en términos de modelos de diseño. Estos artefactos reducen la complejidad de todo el sistema a través de la abstracción. Los esfuerzos hechos por el equipo de diseño disminuirán considerablemente los esfuerzos a realizar en la etapa de codificación, dependiendo del nivel de detalle y a menudo de la estructura.

Muchos equipos fallan en utilizar éstos modelos de diseño como los inicios formales de la codificación. Esto es un error, ya que estos modelos sirven para dirigir todo el esfuerzo, ayudar a que nuevos miembros se integren rápidamente, disminuyan el impacto de los cambios solicitados y disminuyan los riesgos de todo el proyecto. En un proyecto de UCM, una *baseline* de componentes contiene tanto modelos de diseño como artefactos de código.

3.2.3 Artefactos de pruebas

Tradicionalmente las pruebas se sitúan al final de la etapa de desarrollo, después de la codificación. Los equipos de desarrollo saben que las pruebas unitarias son críticamente importantes para el éxito de un proyecto. Además de probar la integración del sistema, se prueban unidades, subsistemas y sistemas. Todas estas pruebas producen numerosos artefactos de pruebas -requerimientos de pruebas, scripts, datos, resultados-. Con UCM, los equipos pueden administrar sus artefactos de pruebas concurrentemente con los artefactos de desarrollo relacionados.

En un proyecto de UCM, las *baselines* de componentes contienen todos los artefactos de código junto con los requerimientos de prueba relacionados, scripts y datos de prueba. Los cambios conllevan a extender los casos de prueba, scripts y datos para validar los cambios.

3.2.4 Artefactos de análisis, diseño, codificación y pruebas

Son muchos los beneficios ofrecidos por *Rational* UCM. Las *baselines* de componentes consolidan todos los artefactos del proyecto en una *baseline*, desde documentos de requerimientos hasta casos de validación, ventaja aún mayor para los equipos que se encargan de las tareas de mantenimiento y mejoras. Con los niveles de *baselines* de UCM, equipos de trabajo relacionados pueden más fácilmente:

- Identificar las actividades de una versión de componentes específica
- Identificar las actividades que requieren pruebas
- Verificar los artefactos del proceso

Además, los niveles de *baselines* de UCM ayudan a identificar cuándo es apropiada una actividad de integración, por ejemplo, saber cuándo una *baseline* ha alcanzado un nivel de calidad apropiado para pruebas de integración.

En UCM, el monitoreo de calidad es mucho mejor con los reportes provistos por *Rational ClearQuest*. Con estas capacidades de reporte, se puede generar datos claros y concisos del estado de un proyecto para detectar problemas más rápidamente, evaluar riesgos y alcanzar soluciones. El trabajo es organizado según las actividades de más alta prioridad y todos los releases logran niveles de calidad predefinidos.

3.3 Diferencias entre UCM y Base *ClearCase*

Base *ClearCase* consiste en un conjunto de herramientas para establecer un ambiente en el cual los desarrolladores pueden trabajar en paralelo en un conjunto de archivos compartidos, y los directores de proyecto pueden definir políticas que establecen cómo los desarrolladores trabajan juntos.

UCM es un método preescrito de utilizar *ClearCase* para control de versiones y gestión de configuración. UCM es una capa sobre Base *ClearCase*. Por lo tanto, es posible trabajar eficientemente en UCM sin tener mayores conocimientos sobre *ClearCase*.

UCM ofrece una solución completa y Base *ClearCase* ofrece la flexibilidad de implementar virtualmente cualquier solución de gestión de configuración que se considere correcta para el ambiente de trabajo.

3.3.1 *Baselines*

ClearCase y UCM permiten crear *baselines*. UCM automatiza el proceso de creación y provee soporte adicional para trabajar con *baselines*. Una *baseline* identifica un conjunto de versiones de archivos que representan el proyecto en determinado punto. Por ejemplo, se puede crear una *baseline* llamada beta1 para identificar una copia de los inicios de los archivos fuentes de un proyecto.

Las *baselines* proveen dos beneficios principales:

- La habilidad de reproducir una determinada entrega del proyecto de *software*.
- La habilidad de agrupar el conjunto completo de archivos relacionados con un proyecto, como el código fuente, el documento de especificación de requerimientos, documentos de diseño y pruebas.

En Base *ClearCase* se puede crear una *baseline* creando una etiqueta de versión y aplicando esa etiqueta a un conjunto de versiones. En UCM, el soporte para *baselines* aparece en toda la interfaz de usuario debido a que UCM requiere que se trabaje con *baselines*. Cuando los desarrolladores se unen a un proyecto, deben primero crear sus áreas de trabajo con la *baseline* recomendada. Este método asegura que todos los miembros de equipo inician con el mismo conjunto de archivos compartidos.

3.3.2 Actividades

En Base *ClearCase* se trabaja a nivel de versiones y de archivos. UCM provee un nivel de abstracción superior: actividades. Una actividad es un objeto de *ClearCase* que se usa para grabar el trabajo requerido para completar una tarea de desarrollo.

Por ejemplo, una actividad puede ser cambiar una interfaz gráfica de usuario. Se necesitarán modificar varios archivos para realizar esos cambios. UCM graba el conjunto de versiones que se crean para completar la actividad de un conjunto de cambios. Se pueden realizar operaciones en un conjunto relacionado de versiones identificando las actividades en lugar de tener que identificar un gran número de versiones. Como las actividades corresponden a una tarea del proyecto, se puede observar el progreso del proyecto más fácilmente.

3.3.3 Políticas de desarrollo

Una parte importante de trabajar en la gestión de configuración de un proyecto de *software* es establecer y reforzar las políticas de desarrollo. En un ambiente de desarrollo en paralelo, es crucial establecer reglas que especifiquen cómo los miembros del equipo acceden y actualizan un conjunto compartido de archivos. Estas políticas son de ayuda en dos formas:

- Minimizan los problemas de la construcción de un proyecto, identificando los conflictos en los cambios hechos por varios desarrolladores.
- Establecen una mejor comunicación entre los miembros del equipo.

Estos son ejemplos de políticas de desarrollo comunes:

1. Los desarrolladores deben sincronizar sus áreas privadas de trabajo con la *baseline* del proyecto recomendada antes de integrar su trabajo al área compartida de trabajo del proyecto.
2. Los desarrolladores deben notificar a otros miembros del equipo por medio de e-mail cuando integren su trabajo al área compartida del proyecto.

3.4 Trabajar con UCM

En UCM, se trabaja en un ciclo que complementa el proceso de desarrollo iterativo. Los miembros de un equipo de proyecto trabajan en un *Proyecto de UCM*. Un proyecto es el objeto que contiene la información de configuración necesaria para administrar un desarrollo, como la construcción de una versión de un sistema. Un proyecto contiene un área de trabajo compartida y normalmente múltiples áreas de trabajo privadas.

Las áreas de trabajo privadas permiten a los desarrolladores trabajar en actividades en paralelo y sin conflictos.

El director de proyecto es responsable del mantenimiento al área compartida de trabajo. Trabaja en los avances del proyecto en la siguiente forma:

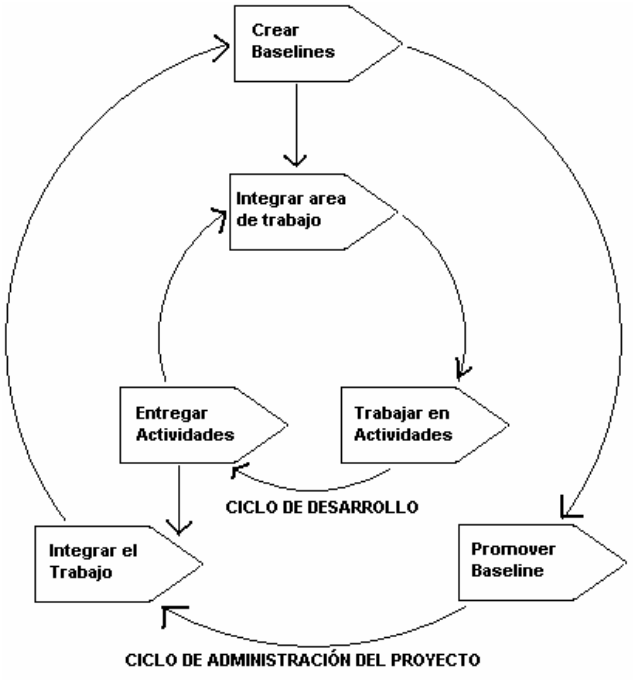
1. Crea un proyecto e identifica el conjunto inicial de *baselines* de uno o más componentes. Un componente es un grupo de directorios y archivos relacionados, que se trabajan, integran y liberan juntos. Una *baseline* es la versión de un componente.
2. Los desarrolladores se unen al proyecto creando su área privada de trabajo.

3. Los desarrolladores crean actividades y trabajan en una actividad a la vez. Una actividad registra el conjunto de archivos que un desarrollador crea o modifica para completar una tarea de desarrollo, como corregir un error.
4. Cuando los desarrolladores completan actividades y prueban su trabajo en las áreas privadas, comparten ese trabajo con el equipo del proyecto llevando a cabo operaciones de *entrega*. Una operación de entrega unifica el trabajo de las áreas privadas de los desarrolladores con el área compartida de trabajo.
5. En el área compartida de trabajo el director de proyecto integra el trabajo de los desarrolladores.
6. El director de proyecto realiza pruebas de validación rápidas para asegurarse de que todo funciona correctamente. Pruebas más extensas las realiza el equipo de calidad del *software*.
7. Los desarrolladores realizan operaciones de rebase para actualizar sus áreas privadas de trabajo con el conjunto de archivos que pertenecen a la nueva *baseline*.
8. Los desarrolladores continúan el ciclo de trabajar en actividades, entregar actividades completadas y actualizando sus áreas privadas de trabajo con las nuevas *baselines*.

3.4.1 Ciclo de vida del proyecto

La lista de tareas de UCM se pueden ver en dos ciclos: administración y desarrollo del proyecto.

FIGURA 2: Ciclos de tareas de UCM



Para crear e iniciar un proyecto se deben realizar las siguientes tareas:

1. Crear el repositorio para almacenar la información del proyecto
2. Crear los componentes que contienen el conjunto de archivos con los que los desarrolladores trabajan
3. Crear *baselines* que identifican las versiones de archivos con los que los desarrolladores inician su trabajo
4. Establecer las políticas de desarrollo

3.4.1.1 El PVOB

ClearCase almacena archivos, directorios y otros objetos en un repositorio llamado *versioned object base* -VOB-. En UCM, cada proyecto debe tener un VOB de proyecto -PVOB-. Un PVOB es un tipo especial de VOB que almacena objetos de UCM, como proyectos y actividades. El PVOB debe existir antes de que se cree un proyecto.

3.4.1.2 Componentes

Conforme el número de archivos y directorios en el sistema crece, se necesita una forma de reducir la complejidad de administrarlos. Los componentes es el mecanismo de UCM para simplificar la organización de los archivos y directorios. Los elementos que se agrupan en componentes típicamente implementan una pieza reusable de la arquitectura del sistema. Al organizar directorios y archivos relacionados en componentes, se puede ver el sistema como un pequeño conjunto de componentes identificables en lugar de un gran número de directorios y archivos.

3.4.1.3 Áreas de trabajo privadas y compartidas

Un área de trabajo consiste de una vista y un flujo. Una vista es un árbol de directorio que muestra una versión individual de cada archivo en el proyecto. Un flujo es un objeto de *ClearCase* que mantiene una lista de actividades y determina cuáles versiones de elementos deben aparecer en la vista.

Un proyecto contiene un flujo de integración, el cual registra las *baselines* del proyecto y permite el acceso a las versiones de los elementos compartidos del proyecto. El flujo de integración y la correspondiente vista de integración representan el área de trabajo compartida del proyecto.

Cada desarrollador en el proyecto tiene un área privada de trabajo, que consiste de un flujo de desarrollo y la correspondiente vista de desarrollo. El flujo de desarrollo mantiene una lista de las actividades del desarrollador y determina que versiones de los elementos deben aparecer en la vista del desarrollador.

3.4.1.4 Iniciar una *baseline*

Después de crear los componentes del proyecto se debe identificar la *baseline* o *baselines* que sirven como punto de inicio para el equipo de desarrollo. Una *baseline* identifica una versión de cada elemento visible en un componente. El flujo de integración del proyecto registra las *baselines*. Cuando los desarrolladores se integran al proyecto, ellos copian a su área de trabajo las versiones de directorios y archivos representados por las *baselines*. Esta práctica asegura que todos los miembros del proyecto empiezan con el mismo conjunto de archivos.

3.4.1.5 Establecer políticas

UCM incluye un grupo de políticas que se pueden establecer para reforzar las prácticas de desarrollo entre los miembros del equipo. Al establecer políticas se puede mejorar la comunicación entre los miembros del equipo y minimizar los problemas al integrar el trabajo.

3.4.2 Planeación del proyecto

Es esencial para desarrollar y dar mantenimiento a un *software* de alta calidad la definición de la Arquitectura del Sistema. El Proceso Unificado de *Rational* establece que definir y usar la arquitectura del sistema es una de las seis mejores prácticas para el desarrollo de *software*. La Arquitectura del Sistema es el nivel más alto de conceptualización de un sistema en su ambiente. El Proceso Unificado de *Rational* establece que la arquitectura del sistema es una guía para:

- Las decisiones significativas acerca de la organización de un sistema de *software*.
- La selección de los elementos estructurales y sus interfaces por las que el sistema está compuesto, juntos con el comportamiento como se especifica en las colaboraciones entre estos elementos.
- La composición progresiva de los elementos estructurales y de comportamiento en subsistemas más grandes.
- El estilo de arquitectura que guía la organización, los elementos y sus interfaces, las colaboraciones y composición.

Una arquitectura del sistema bien documentada mejora el proceso de desarrollo. Es el punto inicial ideal para definir la estructura del ambiente de gestión de configuración.

3.4.2.1 Mapeo de la arquitectura del sistema a componentes

Así como los diferentes tipos de planos representan diferentes aspectos de la arquitectura de un edificio, una buena arquitectura del sistema contiene diferentes vistas para representar sus diferentes aspectos.

El Proceso Unificado de Desarrollo define una vista de arquitectura como una descripción simplificada -una abstracción- de un sistema desde una perspectiva en particular, cubriente aspectos particulares y omitiendo entidades que no son relevantes para esa perspectiva.

Este proceso sugiere usar cinco vistas de arquitectura. De estas, la vista de implementación es la más importante para la gestión de configuración. La vista de implementación identifica los archivos físicos y directorios que implementan los paquetes de lógica del sistema, objetos o módulos.

De la vista de implementación se debe poder identificar el conjunto de componentes de UCM que se necesitarán en el sistema.

3.4.2.2 Decidir qué colocar bajo control de versiones

Al decidir qué colocar bajo control de versiones, no hay que limitarse a los archivos código fuente. El poder de la gestión de configuración es que se puede llevar n registro histórico del proyecto conforme va evolucionando para que se pueda recrear el proyecto fácil y rápidamente en cualquier punto del tiempo.

Para grabar una imagen completa del proyecto hay que incluir todos los archivos y directorios conectados con él. Un conjunto de archivos podrían ser:

1. Archivos de código fuente
2. Archivos de modelos
3. Librerías
4. Archivos ejecutables
5. Interfaces
6. Archivos para pruebas
7. Planes del proyecto
8. Documentación de usuario y del sistema
9. Documentos de especificación de requerimientos

3.4.2.3 Mapeo de Componentes a Proyectos

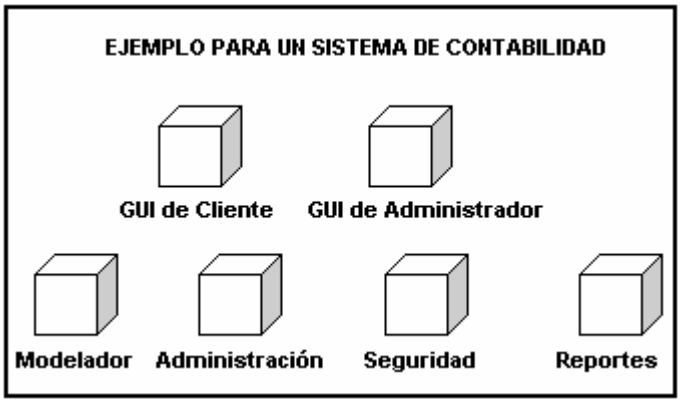
Después de hacer el mapeo de la arquitectura del sistema a un conjunto de componentes se necesita determinar si se usará un proyecto o múltiples proyectos, En general, hay que pensar en un proyecto como el ambiente de gestión de configuración de un equipo de proyecto. Para decidir cuántos proyectos usar, hay que considerar los siguientes factores:

- Tamaño del sistema
- Nivel de integración requerida
- Si se harán entregas de múltiples versiones del sistema simultáneamente

Para el tamaño del sistema hay que considerar el número de desarrolladores y el número de componentes del sistema. El nivel de integración determina las relaciones entre varios componentes. Componentes relacionados que requieren un alto nivel de integración pertenecen al mismo proyecto, con esto se pueden probar juntos frecuentemente.

Si se van a desarrollar diferentes versiones de un sistema en paralelo, hay que usar diferentes proyectos, uno para cada versión. Por ejemplo si hay que trabajar en un arreglo y en una nueva entrega al mismo tiempo.

Figura 3: Ejemplo de componentes



3.4.2.4 Organizar los componentes

Después del mapeo de la arquitectura del sistema a un conjunto inicial de componentes y determinar cuáles proyectos accesarán esos componentes, se debe refinar el plan llevando a cabo las siguientes tareas:

1. Asegurarse de que los componentes no excedan al tamaño de la VOB's
2. Identificar cualquier otro componente adicional
3. Definir la estructura de los directorios de componentes
4. Identificar componentes de solo lectura
5. Identificar componentes no unificables

3.4.2.4.1 Estructura de los directorios

Una estructura recomendada para iniciar el proyecto es la siguiente:

Tabla III: Estructura de directorios

Componente	Directorio	Contenido
Sistema	Planes	Planes del proyecto, etc.
	Requerimientos	Documentos de requerimientos
	Modelos	Documentos de la arquitectura
	Documentación	Documentación del sistema
Componente 1 a Componente N	Requerimientos	Requerimientos para los componentes
	Modelos	Modelos de los componentes
	Código fuente	
	Interfaces	Interfaces públicas
	Ejecutables	
	Librerías	Librerías usadas por el componente
	Pruebas	
Pruebas	Scripts	Datos usados y pasos para realizarlas
	Resultados	
	Documentación	
Instalación	Ejecutables	Archivos ejecutable instalados
	Librerías	Librerías instaladas
	Interfaces	
	Documentación	Documentación de usuario

Componente 1 a componente n se refiere a los componentes que hacen el mapeo del conjunto de paquetes lógicos en la arquitectura del sistema.

3.4.2.5 Especificar la estrategia para *baselines*

Después de organizar los componentes del proyecto, hay que determinar la estrategia para crear *baselines* para esos componentes. La estrategia para *baselines* debe definir lo siguiente:

3.4.2.5.1 Cuándo crear una *baseline*

Al inicio se crea una o más *baselines* como punto de inicio para el desarrollo. Luego se van creando más periódicamente.
La creación de *baselines* frecuentemente tiene como beneficios que los desarrolladores se mantienen sincronizados con el trabajo de los demás, la cantidad de tiempo requerida para unir versiones es minimizada y los problemas de integración son identificados pronto

3.4.2.5.2 Definir los nombres

Como las *baselines* son una herramienta importante en la administración del proyecto, hay definir un nombre significativo para ellas. Un nombre útil provee la siguiente información: nombre del proyecto, fase de la planificación del desarrollo y fecha de creación.

3.4.2.5.3 Identificar el nivel de promoción para reflejar el estado del desarrollo

Un nivel de promoción es un atributo de una básenle que se utiliza para indicar la calidad o estabilidad de la *baseline*.

ClearCase proporciona los siguientes niveles de promoción, pero se pueden definir otros que se adecuen al proyecto:

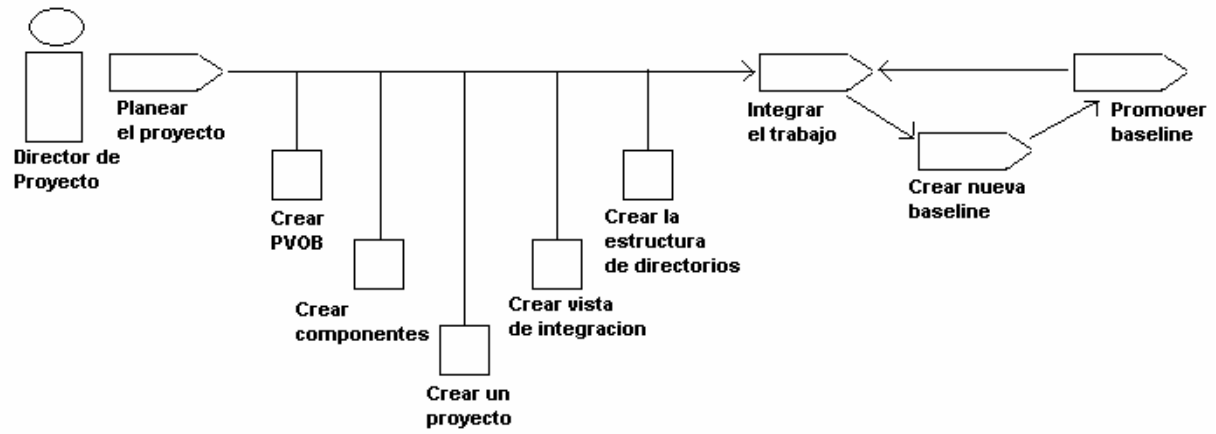
1. Rechazado
2. Inicial
3. Construido
4. Probado
5. Entregado

Se deben determinar los niveles de promoción para el proyecto y el criterio pasar a cada uno de los niveles.

3.4.3 Creación del proyecto

Esta parte describe cómo crear un proyecto desde cero para trabajar en un ambiente de UCM. Antes de crear un proyecto hay que llevar a cabo la planeación del proyecto.

Figura 4: Flujo de trabajo



3.4.3.1 Crear la VOB del proyecto

Para crear la PVOB hay que realizar los siguientes pasos:

1. Click en *Start>Programs>Rational ClearCase Administration>Create VOB*. Con esto se abre la ventana de creación de VOB.
2. En el primer paso se ingresa el nombre del PVOB, puede ingresarse un comentario. Seleccionar la opción de *datos de proyecto UCM*.
3. En el segundo paso especificar el directorio de almacenamiento para el PVOB, este sirve de repositorio para el contenido del PVOB.
4. En el tercer paso solicita el VOB de administración con el cual asociar el PVOB. Como el proyecto se está creando desde cero, no hay ningún VOB de administración, seleccionar la opción de *none*. Así, el PVOB asume el rol de VOB de administración.

3.4.3.2 Crear componentes

Para crear componentes se realizan los siguientes pasos:

1. Abrir la ventana de creación de VOB.
2. En el paso uno, ingresar el nombre del componente. Seleccionar la opción de *Crear VOB como un componente de UCM*.
3. En el paso dos, seleccionar donde almacenar el componente.
4. En el paso tres, se identifica el PVOB que almacenará la información del componente. Seleccionar de un listado el PVOB que se creó anteriormente.

ClearCase crea el componente con una *baseline* inicial que apunto a la primera versión del componente.

3.4.3.3 Crear el proyecto

El proyecto se puede crear desde el Explorador de Proyectos en la ventana de Nuevo Proyecto:

1. En el panel izquierdo del Explorador de *ClearCase*, click en UCM y luego en Explorador de Proyectos. El explorador de proyectos es una interfaz gráfica de usuario en la cual se puede crear, administrar y ver información de los proyectos.
2. El panel izquierdo del Explorador de Proyectos, lista los directorios raíz para todos los PVOB en el dominio local de *ClearCase*. Cada PVOB tiene su directorio raíz. *ClearCase* crea el directorio raíz con el nombre del PVOB. También se crea un directorio llamado Components, el cual contiene cada componente creado en el PVOB. Seleccionar el directorio raíz del PVOB que se desea usar para almacenar la información del proyecto.
3. Click *en File>New>Folder* para crear el directorio del proyecto.
4. En el panel izquierdo seleccionar el directorio del proyecto o directorio raíz. Click en *File>New>Project*. La ventana de nuevo proyecto aparece.
5. En el paso uno, ingresar un nombre significativo para el proyecto en el cuadro de *Título del Proyecto*.
6. En el paso dos, pregunta si se desea crear un proyecto en base a otro existente, seleccionar que *No*.

7. El paso tres pregunta por las *baselines* que el proyecto usará. Estas *baselines* son conocidas como *baselines* de fundación. Click *Add* para abrir el cuadro de Agregar *Baseline*. En la lista de Componentes, seleccionar uno de los componentes que se crearon previamente. La *baseline* inicial del componente aparece en la lista de *baselines*. Seleccionar la *baseline*. Hay que asegurarse que la opción de que el proyecto pueda modificar el componente esté seleccionada, de lo contrario el componente será de solo lectura. Click en *OK*. La *baseline* ahora aparece en la lista del paso tres.
8. El paso cuatro pregunta por las políticas de desarrollo para el proyecto.
9. El paso cinco pregunta si se configurará el proyecto para trabajar integradamente con *ClearQuest*.

3.4.3.4 Definir los niveles de promoción

ClearCase provee cinco niveles de promoción para *baselines*. Se pueden dejar algunos o todos y se pueden definir otros. Para definir otros niveles:

1. En el Explorador de Proyectos, seleccionar el directorio raíz del PVOB que contiene el proyecto, y luego click en *Tools>Define Promotion Level*. Todos los proyectos que usan el mismo PVOB tienen acceso a los mismos niveles de promoción.
2. El cuadro de *Define Promotion Levels* aparece. Aquí se pueden ordenar los niveles existentes.
3. Para agregar un nivel nuevo, click en *Add*. Ingresar el nombre del nuevo nivel y click en *OK*. El nuevo nivel aparece en el listado de niveles de promoción.
4. Cuando se termine de ordenar los niveles, se selecciona uno para ser el inicial para las nuevas *baselines*. Es el nivel asignado automáticamente al crear una *baseline*.

4. EL PROCESO DE GESTIÓN DE CONFIGURACIÓN DE *SOFTWARE* EN LAS EMPRESAS GUATEMALTECAS

En éste capítulo se evalúan los métodos de control de cambios y de gestión de configuración de las empresas de desarrollo de *software* en Guatemala.

La evaluación se realiza por medio de una encuesta, que es entregada a los jefes de proyecto de las empresas. Se hace una evaluación dividiendolas en empesas que desarrollan *software* para comercializar y empresas que tienen un departamento interno de desarrollo de sistemas.
También se cuestiona sobre la herramienta de GCS que utilizan, si están conformes con ésta o si piensan cambiarla.

4.1 Los métodos actuales de control de cambios en las empresas

Las empresas guatemaltecas de desarrollo tienen métodos de control de cambios similares entre ellas. Debido a que la mayoría de empresas de desarrollo no son grandes en cantidad de programadores y cantidad de locaciones de trabajo, tratan de llevar un control de cambios más rápido recibiéndolos del cliente y trasladándolos a los programadores para que empiecen a realizar los cambios. Los cambios son recibidos por el equipo de desarrollo, se hace una breve descripción que es la que se traslada a los programadores y se comienzan a ejecutar sin hacer un análisis muy exhaustivo del impacto que tienen sobre el trabajo ya realizado. Muchas veces, se introducen nuevos defectos al tratar de arreglar uno que haya sido reportado, extendiendo el tiempo que se tiene estimado para hacer el cambio.

No existe un grupo de personas, del equipo de desarrollo, que se encargue de manejar únicamente los cambios y esté a cargo de que cada modificación vaya de acuerdo a lo que se está solicitando y no altere el trabajo que ya se tiene realizado. Son los mismos analistas y diseñadores los que se encargan de recibir los cambios y autorizar su realización. Son ellos los que guardan los documentos de especificaciones del *software* y hacen las modificaciones conforme se reciben del cliente. Son pocas las empresas que cuentan con un medio de almacenamiento para todos los artefactos de un proyecto.

Hay empresas que tienen proyectos en los que solamente trasladan los sistemas a otro lenguaje de programación, bajo la misma funcionalidad que tiene el sistema actual, con mejoras en el rendimiento, capacidad de almacenamiento y/o tiempos de respuesta. En éstos proyectos, los cambios no son tan impactantes para la lógica del sistema, y la mayoría llegan a ser solamente de aspectos estéticos. Por esto, los cambios pueden ser poco detallados y a veces no ser documentados debidamente.

Otra forma para especificar los cambios por parte del cliente, es el uso de “e-mail”. Los clientes que no tienen mucho tiempo para reuniones extensas de verificación del sistema que se les está entregando, optan por hacer las verificaciones por su cuenta y luego enviar un e-mail al equipo de desarrollo con las observaciones que se hayan tenido de esa verificación. Con este medio de comunicación, se tiene al menos una copia escrita de los cambios que se solicitan y cuándo se solicitan.

4.2 Encuesta realizada

Para conocer los métodos de control de cambios y el proceso de Gestión de Configuración que existe en las empresas guatemaltecas, se realizó una encuesta con el objetivo de conocer sus procedimientos de control de cambios y sus conocimientos acerca de la gestión de configuración.

4.2.1 Descripción

El objetivo de la encuesta es conocer qué tanto conocen el proceso de Gestión de Configuración las empresas de desarrollo, si lo aplican y qué tan eficiente es su método de control de cambios. Cuáles de las actividades del proceso de GCS realiza la empresa, los medios de almacenamiento y de respaldos de los proyectos que realizan.

La encuesta se realizó a los jefes de proyecto de un grupo de 25 empresas de desarrollo de *software* en la ciudad capital de Guatemala, *software* comercial y *software* interno. Empresas que varían en tamaño y métodos de desarrollo. Independientemente de los métodos y lenguajes de desarrollo, cada empresa tiene sus métodos de control de cambios, que como se mencionó anteriormente, son similares para muchas de ellas. Algunas, a pesar de ser empresas pequeñas, tienen un mejor control de cambios que otras.

Las preguntas realizadas fueron preguntas con respuesta de selección múltiple, para que se pudiera medir qué tanto conocen del proceso de Gestión de Configuración. También se evaluó si las empresas cuentan con alguna herramienta de *software* dedicada al control de cambios y control de avance del proyecto.

4.2.2 Encuesta

1. ¿Desarrolla la empresa *software* para comercializar o es el departamento de informática de para desarrollo interno de una empresa?
   Software para comercializar ____ Desarrollo interno ____

2. ¿Conoce el proceso de Gestión de Configuración?
   Si ____ No ____

3. ¿Aplica la empresa un método de Gestión de Configuración?
   Si ____ No ____

4. ¿Se incluye el plan de Gestión de Configuración en la documentación de los proyectos?
   Si ____ No ____

5. ¿Cuentan con un medio de almacenamiento dedicado a los artefactos producidos para los proyectos que realizan?
   Si ____ No ____

6. ¿Tiene la empresa un equipo dedicado al control de versiones de los productos de *software*?
   Si ____ No ____

7. ¿Conoce la herramienta de control de cambios Rational Unified Change Management?
   Si ____ No ____

8. ¿Es efectivo el método de control de cambios con el que la empresa cuenta actualmente?
   Si ____ No ____

9. ¿Que herramienta de control de cambios utiliza actualmente?
   PVCS -Polytronic Version Control System- _____ CVS -Concurrent Versioning System- _____
   RCS -Revision Control System- _____ Rational UCM _____
   Otra: ____ No utiliza _____

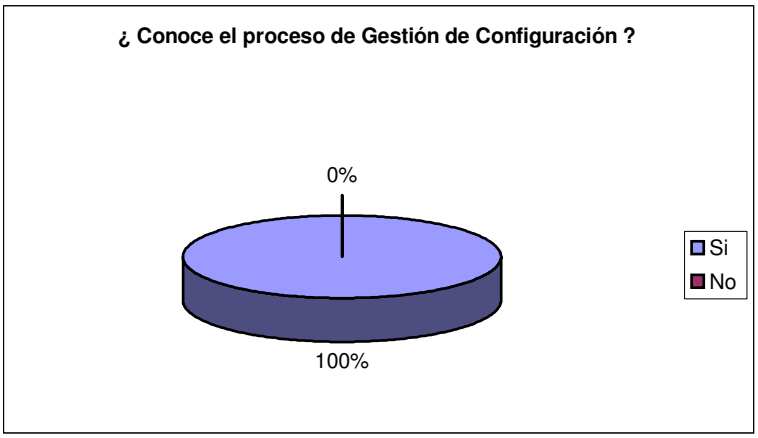
10. ¿Si adoptaran una herramienta de *software* para el control de cambios diferente a la que tiene actualmente, qué herramienta elegiría?
    PVCS -Polytronic Version Control System- _____ CVS -Concurrent Versioning System- _____
    RCS -Revision Control System- _____ Rational UCM _____
    Otra: _____

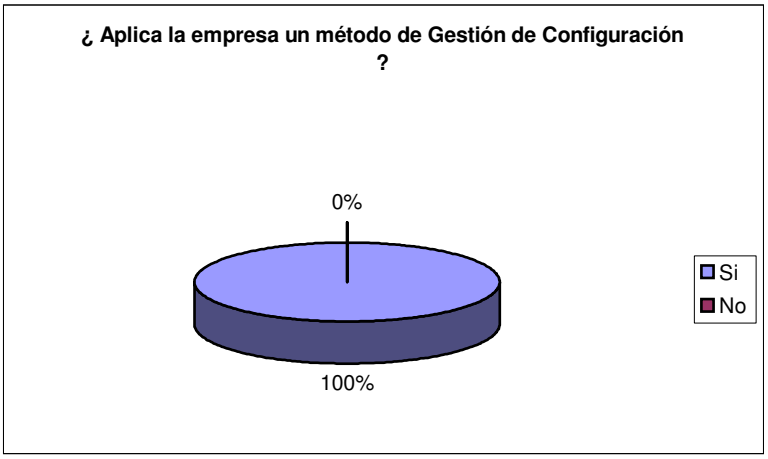
11. ¿Considera importante para su empresa un control de cambios muy detallado y con todas las actividades que forman la Gestión de Configuración?
    Si ____ No ____

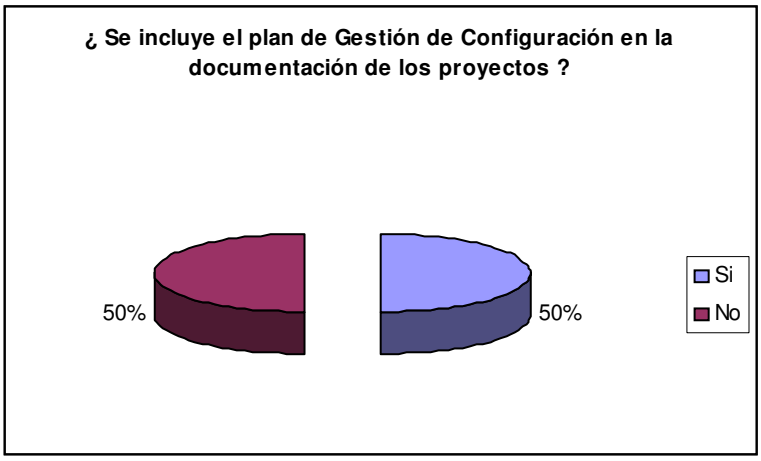
12. ¿Si decidiera implementar el proceso de Gestión de Configuración en su empresa, en qué período de tiempo comenzaría a implementarlo?
    1 año _____ 2 años _____ 3 años _____ Más de 3 años _____
    Ya está implementado _____ Está implementado pero se piensa cambiar la herramienta ____

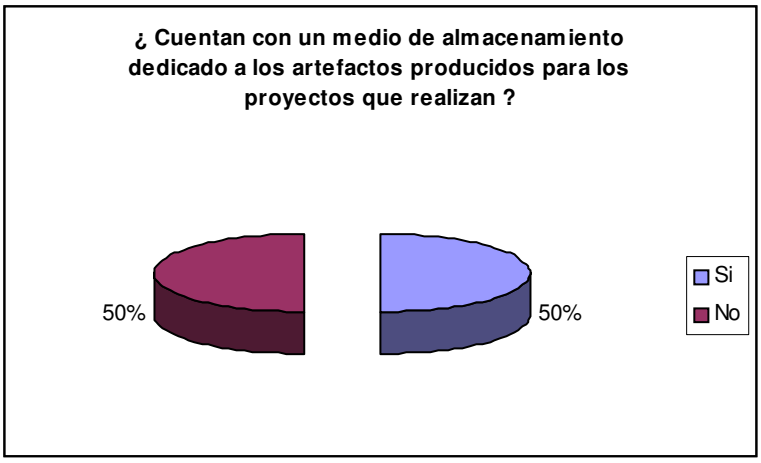
4.2.3 Resultados de la encuesta

Resultados para departamentos de desarrollo interno

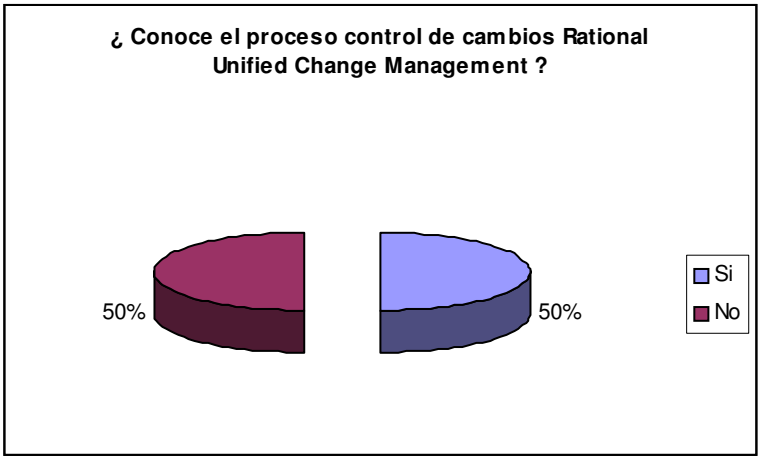


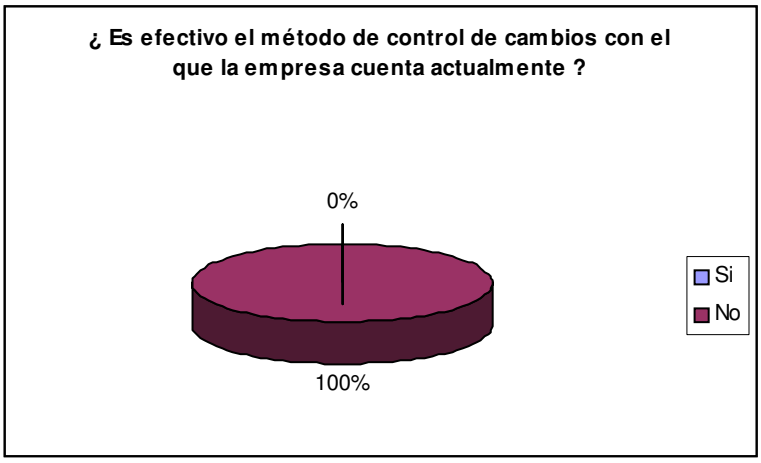


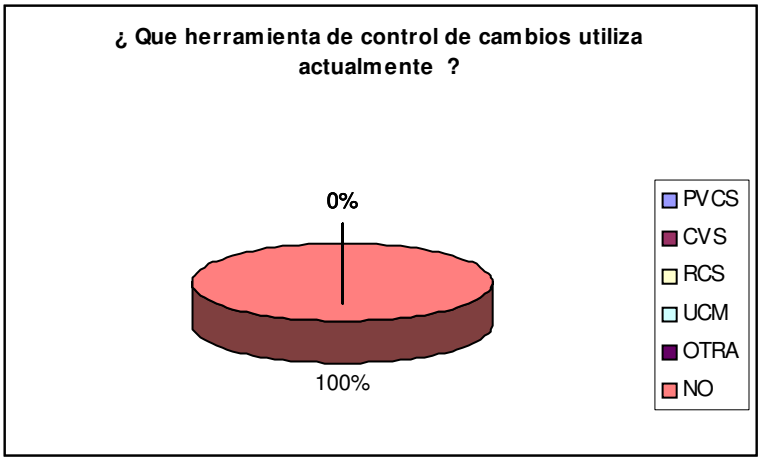


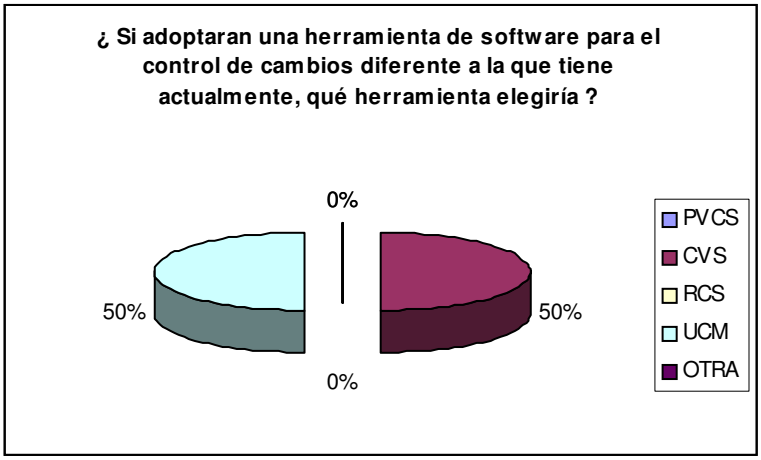

¿ Cuentan con un medio de almacenamiento dedicado a los artefactos producidos para los proyectos que realizan ?
50%
50%
Si
No

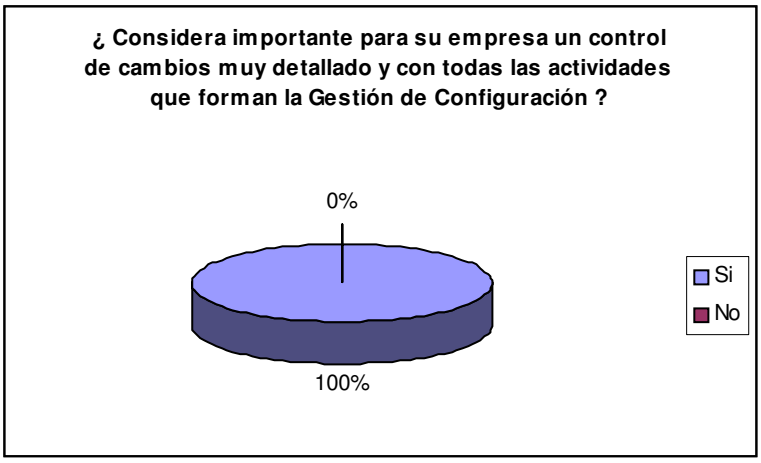

¿ Tiene la empresa un equipo dedicado al control de versiones de los productos de software ?
0%
100%
Si
No

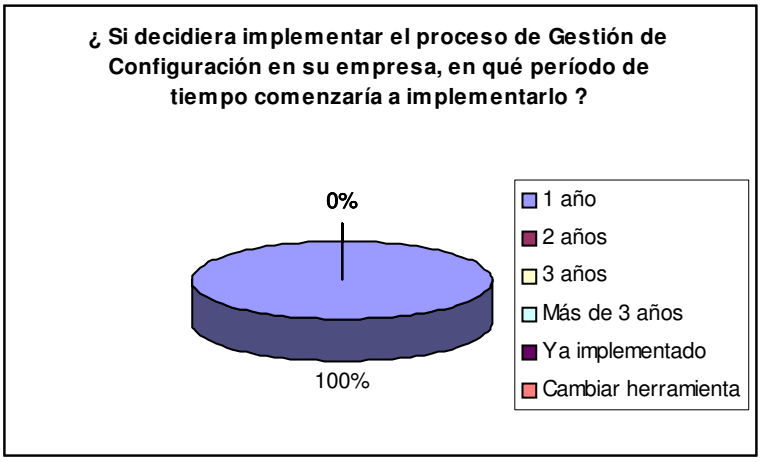

¿ Conoce el proceso control de cambios Rational Unified Change Management ?
50%
50%
Si
No


¿ Es efectivo el método de control de cambios con el que la empresa cuenta actualmente ?
0%
100%
Si
No

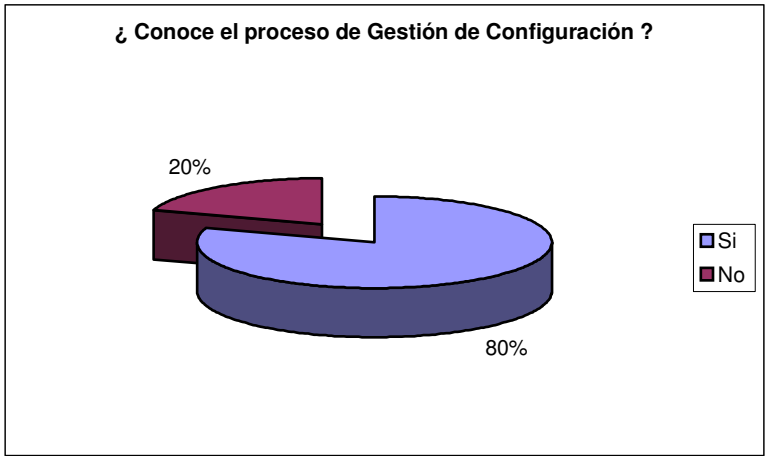

¿ Que herramienta de control de cambios utiliza actualmente ?
0%
100%
PVCS
CVS
RCS
UCM
OTRA
NO

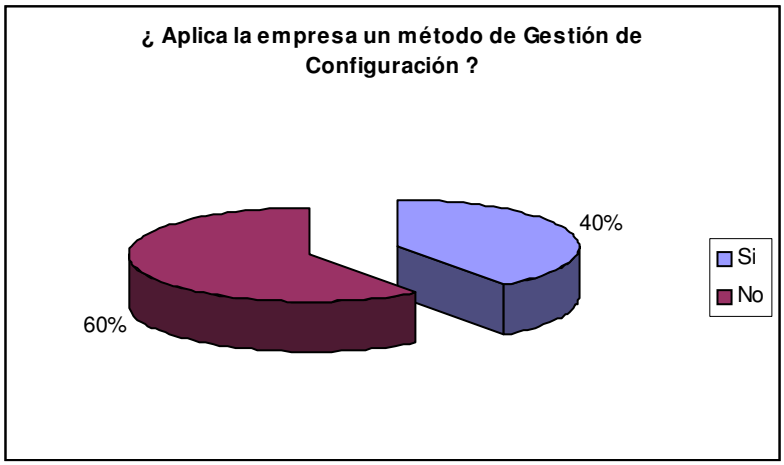

¿ Si adoptaran una herramienta de software para el control de cambios diferente a la que tiene actualmente, qué herramienta elegiría ?
0%
50%
50%
0%
PVCS
CVS
RCS
UCM
OTRA

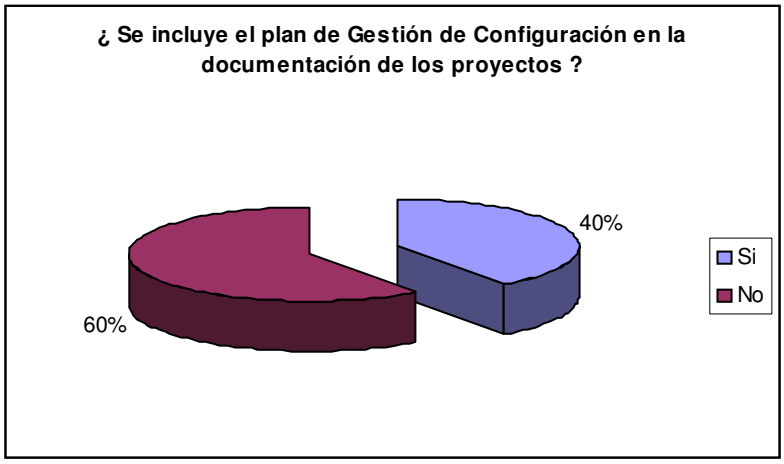


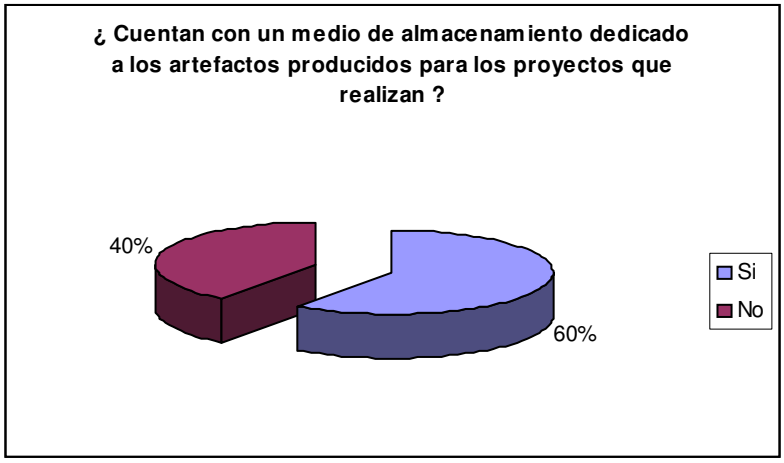


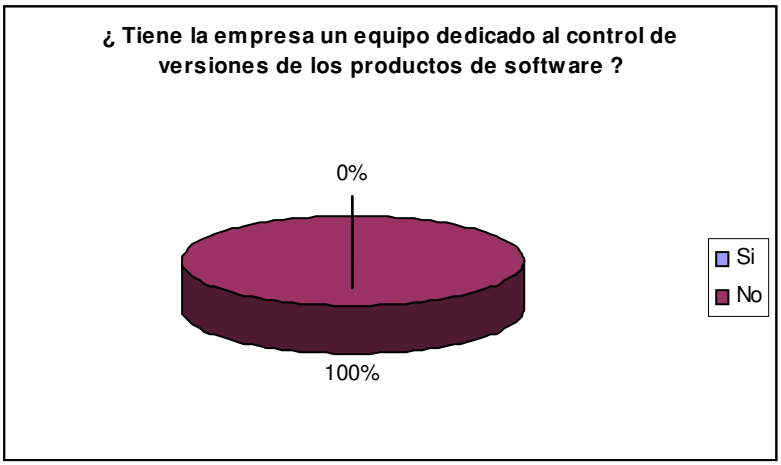
Resultados de empresas que desarrollan *software* comercial

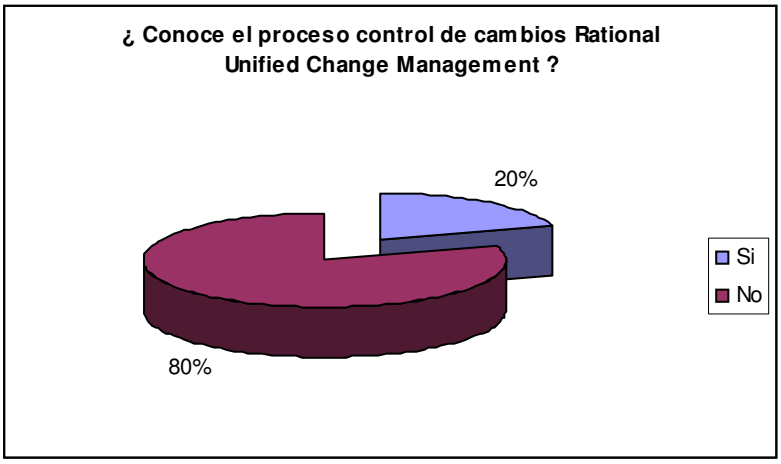


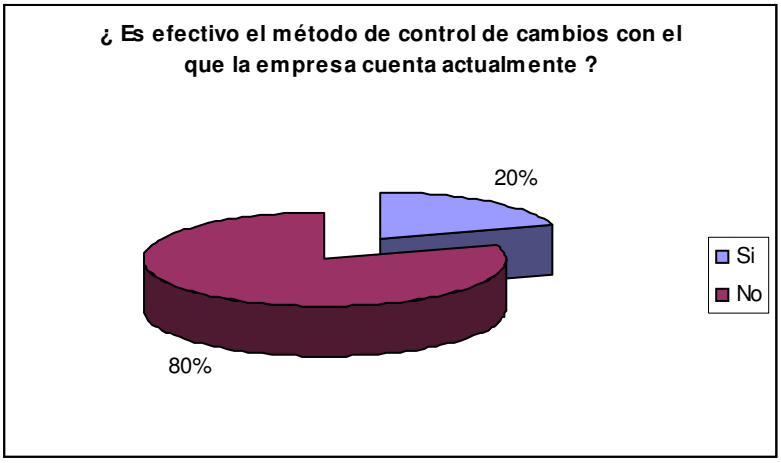

¿ Aplica la empresa un método de Gestión de Configuración ?
40%
60%
Si
No

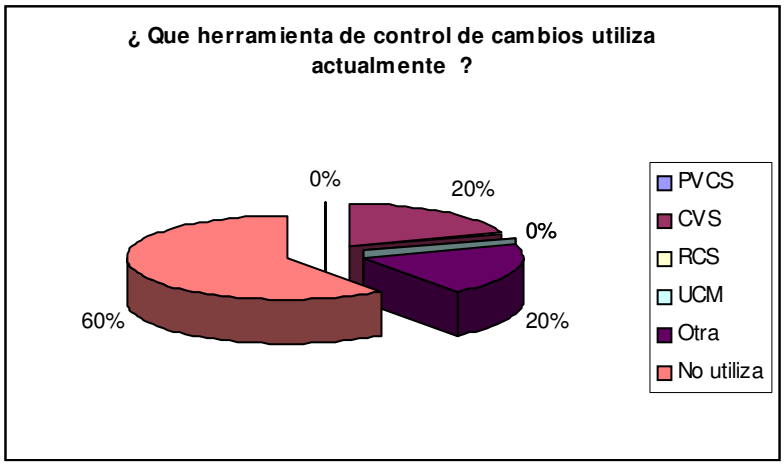

¿ Se incluye el plan de Gestión de Configuración en la documentación de los proyectos ?
40%
60%
Si
No

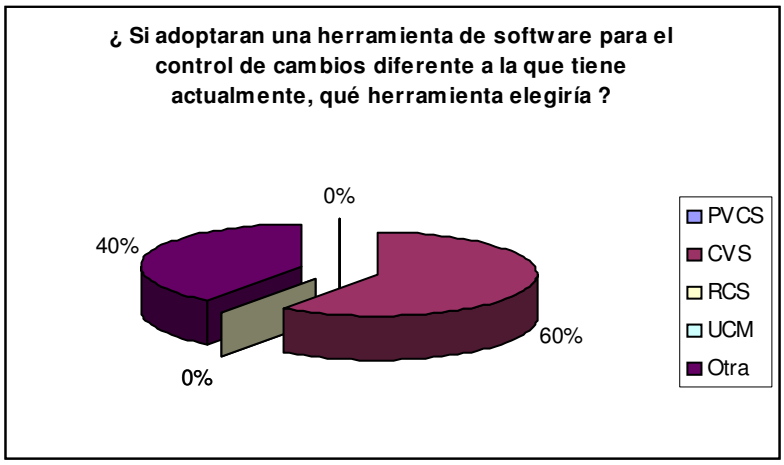

¿ Cuentan con un medio de almacenamiento dedicado a los artefactos producidos para los proyectos que realizan ?
40%
60%
Si
No

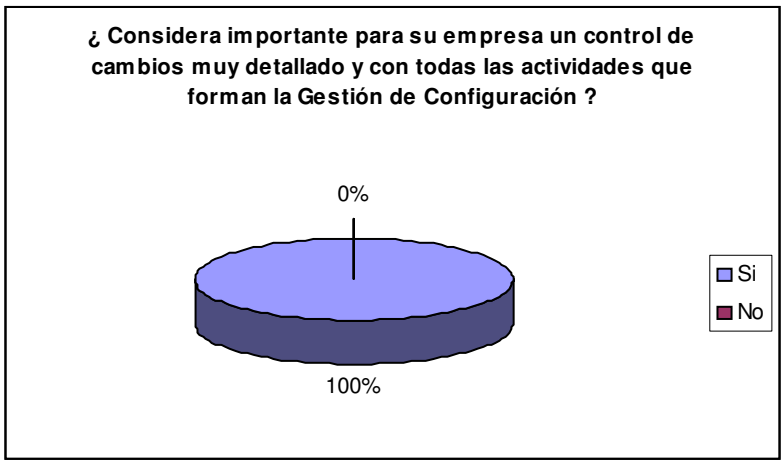

¿ Tiene la empresa un equipo dedicado al control de versiones de los productos de software ?
0%
100%
Si
No

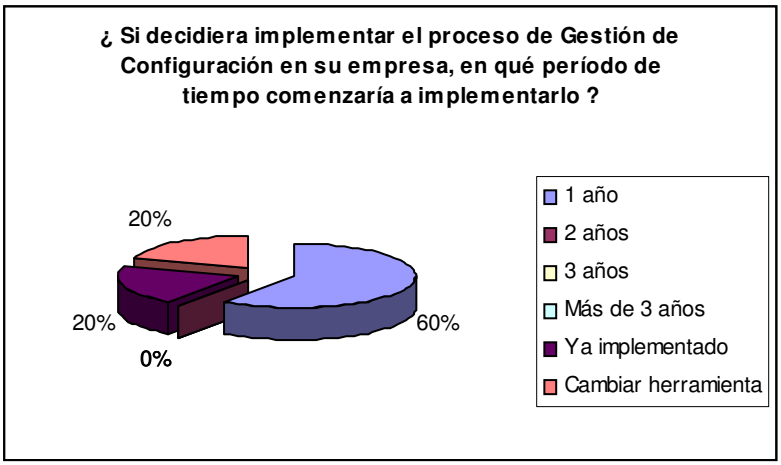

¿ Conoce el proceso control de cambios Rational Unified Change Management ?
20%
80%
Si
No


¿ Es efectivo el método de control de cambios con el que la empresa cuenta actualmente ?
20%
80%
Si
No


¿ Que herramienta de control de cambios utiliza actualmente ?
0%
20%
0%
60%
20%
PVCS
CVS
RCS
UCM
Otra
No utiliza


¿ Si adoptaran una herramienta de software para el control de cambios diferente a la que tiene actualmente, qué herramienta elegiría ?
0%
40%
60%
0%
PVCS
CVS
RCS
UCM
Otra


¿ Considera importante para su empresa un control de cambios muy detallado y con todas las actividades que forman la Gestión de Configuración ?
0%
100%
Si
No



4.3 Comparación de métodos de control de cambios

De los resultados de la encuesta se puede observar que son pocas las empresas de desarrollo que conocen el proceso de Gestión de Configuración, y todavía menos los que tratan de implementarlo. Esto depende también del tamaño de la empresa, ya que en las empresas pequeñas, los métodos simples de control de cambios son suficientes para controlar los proyectos que desarrollan. La necesidad de métodos más efectivos se ve en las empresas grandes, que manejan varios proyectos al mismo tiempo, con varios programadores por cada proyecto, donde sí es necesario un control más completo de los cambios y de todo el desarrollo de los proyectos.

En los departamentos de desarrollo se puede notar que si conocen el proceso de gestión de configuración y que realmente lo implementan en los sistemas que desarrollan para la empresa donde trabajan.

4.3.1 Beneficios de los métodos actuales

En las empresas que son pequeñas o con proyectos no muy complejos, es suficiente con que los cambios sean brevemente especificados. Dado que no son proyectos grandes, el impacto que tendrían los cambios no retrasaría el proyecto por un período de tiempo muy largo.

Como los cambios se reciben y autorizan por la misma persona, se pueden llevar a la implementación en un tiempo corto, y así, ir cumpliendo con cada cambio lo más rápido posible. Esto siempre, de acuerdo al tamaño y complejidad del proyecto.

4.3.2 Beneficios de adoptar el proceso de GCS

Al tener definido en la empresa un proceso de Gestión de Configuración, podrían fácilmente empezar a trabajar con proyectos grandes sin la necesidad de redefinir sus procesos actuales. Tendrían más control sobre los proyectos en desarrollo y éstos serían independientes para varios grupos de programadores o en paralelo para un mismo grupo de programadores.

También es necesario que las empresas tengan un medio de almacenamiento donde estén todas las versiones de *software* con una funcionalidad ya definida, no solamente en la computadora de cada programador o de cada analista, deben contar con un lugar donde esté toda la información referente a un proyecto, desde los documentos de especificación hasta los últimos cambios requeridos después de la entrega final.

Las empresas que no cuentan con una herramienta que les automatice la Gestión de Configuración, pueden tomar en cuenta la solución de Rational, ya que ésta solución provee métodos predefinidos que la empresa podría utilizar para iniciarse en el proceso de GCS, y ya cuando la empresa se familiarice y controle mejor el proceso, puede definir sus propios métodos de control, tienendo la flexibilidad de configurar la herramienta como mejor les convenga.

CONCLUSIONES

1. El proceso de Gestión de Configuración debería ser un proceso que todos los equipos de desarrollo deberían implementar, no importando el tamaño del proyecto o del equipo. Siempre es necesario el control de cambios.

2. La solución de Racional, UCM, a través de sus herramientas *ClearCase* y *ClearQuest* provee una forma muy eficiente de implementar un proceso de Gestión de Configuración, automatizando la mayoría de trabajo de control y ajustándose a las necesidades del equipo de desarrollo por medio de sus procesos predefinidos personalizables.

3. En Guatemala, la mayoría de las empresas de desarrollo de software no conoce el proceso de Gestión de Configuración y de las que lo conocen, son pocas las que lo implementan.

4. Para elevar el nivel de calidad en los productos de las empresas guatemaltecas y la puntualidad en los proyectos, es necesario que implementen un proceso de Gestión de Configuración.

RECOMENDACIONES

1. Los equipos de desarrollo que trabajen en empresas comerciales o en departamentos internos, deben considerar la gestión de configuración como uno de sus procesos fundamentales para asegurar el éxito de los proyectos.

2. Todos los equipos de desarrollo que implementen la gestión de configuración deberían contar con herramientas que les ayuden a automatizar las actividades que requiere éste proceso, como por ejemplo: Rational Unified Change Management.

3. El control de versiones es un área del proceso de gestión de configuración que no se trató a profundidad en ésta investigación, al momento de implementar éste proceso en una empresa, se debe ahondar más en el control de versiones y definir, detalladamente, la forma en que se va a llevar de acuerdo a las políticas que trabaje la empresa.

BIBLIOGRAFÍA

1. H. Ronald Berlack. **Software Configuration Management**. John Wiley & Sons, Inc. 1992

2. Rational Software Corporation. **Managing Software Projects whith Clear Case**. 2001

3. Rational Software Whitepaper. **Unified Change Management from Rational Software**. 2002

4. http://www.rational.com/products/whitepapers/scmseg.jsp

5. http://www.rational.net/main/cata.html?USE_CASE=viewcontent&SERVICE_NAME=Vignette&SERVICE_SPECIFIC_ID=2018&RESOURCE_ID=191958

6. http://scmlabs.com/htdocs/html/knowledge_release_based.htm

7. http://www.perforce.com/perforce/bestpractices.html