03063

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

5
24

UNIDAD ACADEMICA DE LOS CICLOS PROFESIONALES Y DE POSGRADO

INSTITUTO DE INVESTIGACIONES EN MATEMATICAS APLICADAS Y EN SISTEMAS

"ADMINISTRACION DE LA CONFIGURACION DEL SOFTWARE"

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS DE LA COMPUTACION

P R E S E N T A :

ANIBAL RIOS CAMPOS

DIRECTORA:

M. EN C. MARIA GUADALUPE ELENA IBARGÜENGOITIA GONZALEZ.

260934

MEXICO, D. F. MAYO 1998.

TESIS CON
FALLA DE ORIGEN

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

AGRADECIMIENTO.

A mi directora de tesis M. en C. María Guadalupe Elena Ibargüengoitia González por haberme tenido mucha paciencia al conducir este trabajo, por darme estímulos para llegar a la conclusión de la misma y por brindarme parte de su tiempo.

A mis sinodales por la revisión de este trabajo y sus comentarios para mejorarlo:

Ing. Mario Rodríguez Manzanera.
M. En C. Gustavo Arturo Marquez Flores.
M. En C. Francisco Edgar Castillo Barrera.
Dra. Hanna Oktaba.

Al CONACyT por los dos años de crédito que me otorgaron para estudiar la maestría.

Al Lic. Eddy Hugo Aparicio Hernández compañero de la maestría por su participación y comentarios en el desarrollo del "Sistema administrador de configuraciones" (SAVE).

A Lulú, Juanita, Violeta y Mardonio, todos ellos personal de la maestría, por el servicio atento y desinteresado que brindaron.

A mi amigo Armando Náñes Martínez por sus comentarios y apoyo moral.

A mis familiares, amigos y maestros que me han guiado en la trayectoria de mi vida.

Y a todos aquellos que haya olvidado...

Reciban todos ustedes mis más sinceras gracias:

Anibal Rios Campos.

CONTENIDO

INTRODUCCIÓN.

Durante el desarrollo y mantenimiento de un producto de software hay varias actividades que deben llevarse a cabo constantemente. Una de ellas es la administración de la configuración. La configuración de un sistema está sometida a cambios. Con una adecuada administración de la configuración se logrará que el caos no domine al introducir cambios y que se logre configurar exactamente la versión deseada.

La tesis consta de dos partes: en la primera, se hace una revisión bibliográfica de los conceptos fundamentales de la administración de la configuración del software; en la segunda, se describe la construcción de un prototipo de herramienta de apoyo para llevar a cabo dicha administración. La primera, consta de los capítulos uno al nueve; la segunda del diez al once.

Los conceptos recopilados aclaran como poder establecer una adecuada administración de la configuración del software, también muestran como implementar el registro de la configuración de un sistema de software por medio de una base de datos.

El capítulo 1 se enfoca a la introducción de la administración de la configuración del software (SCM). SCM es una palabra formada de las iniciales del título en inglés "software configuration management" y es un término que se utilizará con frecuencia en este trabajo. Aqui se define el concepto de SCM, así como sus actividades principales.

El capítulo 2 describe la importancia de la identificación del software en todo el ciclo de vida del sistema. Presenta los conceptos de línea de base que son fundamentales para aplicar SCM y lo que es un elemento de software y su jerarquía .

El capítulo 3 explica como controlar la configuración. El alterar la configuración comienza con una solicitud de cambio a un elemento de software que al implementarla genera una versión, la cual altera la configuración del producto dando como resultado una versión diferente del sistema (nueva versión).

El capítulo 4 expone como llevar a cabo el reporte del estado de la configuración del software al que se está desarrollando o al que se está dando mantenimiento. La forma de escoger los datos que servirán para llevar un registro de los elementos. Esto se reflejará en reportes que apoyen al administrador del proyecto para valorar los efectos de los cambios.

El capítulo 5 define las auditorías que se deben llevar a cabo cuando se alcanza la línea de base de producto. Estas auditorías que desempeña SCM son: auditoría de la configuración física y auditoría de la configuración funcional.

El capítulo 6 indica como administrar las bibliotecas de software que se van

construyendo a lo largo del ciclo de desarrollo. También trata de las liberaciones que se llevan a cabo para los clientes.

El capítulo 7 muestra como construir un sistema que cumpla la configuración requerida. También aclara como definir configuraciones cuando no se posee una herramienta de simulación.

El capítulo 8 expone brevemente el contenido de un plan para SCM con los errores que se pueden cometer al definirlo.

El capítulo 9 trata de las características que deben tener las herramientas de ayuda a las actividades de SCM y describe algunas libres y comerciales.

En la segunda parte.

El capítulo 10 presenta el análisis del prototipo de un sistema administrador de configuraciones (SAVE), se establecen sus requerimientos, el modelo de objeto y el modelo dinámico con algunos diagramas de escenarios.

El capítulo 11 describe el diseño del prototipo. Los puntos que se tocan son: como se hace la persistencia de los datos, la arquitectura y algunas pantallas de SAVE.

Finalmente se presentan algunas conclusiones y mejoras al prototipo.

1. INTRODUCCIÓN A LA ADMINISTRACIÓN DE LA CONFIGURACIÓN DEL SOFTWARE.

La administración de la configuración del software (SCM) es de vital importancia para el desarrollo y mantenimiento de productos de software de calidad. Su origen comienza a finales de la década de los cincuenta y principios de la década de los sesenta con la administración de la configuración del hardware, la que da origen a la especialización del software. Es la industria militar la que observa que es necesario crear una técnica de administración y una disciplina para resolver los problemas de calidad pobre, ensambles erróneos y partes no reparadas, esto con el fin de garantizar la reproducción de sus naves espaciales. Con el paso del tiempo surgió la especialización para el software.

Las primeras computadoras tenían poca memoria y como resultado el programador tenía que hacer que su programa cupiera en el número reducido de celdas o pedazos de memoria. Al principio de la década de los setenta surge un estándar el MIL STD 483, donde ya se le da importancia a la administración de los programas de computadora, de hecho fue el primer estándar que reconoció la administración de hardware y software. La consolidación de SCM se da con el estándar MIL STD 973, ya que aquí el hardware y software se separan y cada uno recibe su propia forma de administración.

1.1 ¿Qué es la administración de la configuración del software?

La administración de la configuración del software en inglés "software configuration management" (SCM), tiene como objetivo administrar la evolución de un proyecto (sistema) de software, desde su desarrollo hasta su mantenimiento.

Con SCM se trata de dirigir de forma ordenada la construcción de un sistema así como el mantenimiento que se llevará a cabo cuando ya esté operando. Veamos algunas definiciones de ciertos autores respecto a SCM.

* SCM se define como la disciplina de identificar la configuración de un sistema en puntos discretos del tiempo, con el propósito de controlar (administrar) sistemáticamente los cambios a esta configuración, mantener la integridad y seguimiento de esta configuración a través del ciclo de vida del sistema [Ber 80].

* SCM es un proceso de identificar y definir los elementos en el sistema, controlando los cambios de esos elementos, las solicitudes de cambios, y verificando la completes y correctez de los elementos [IEEE- 729 83].

* SCM es el arte de identificar, organizar y controlar modificaciones al software construido por un equipo de programadores. El objetivo de SCM es maximizar la productividad minimizando los errores [Bab 86]

* SCM es una colección de técnicas las cuales identifican y controlan cambios al

software desarrollado por un equipo de programadores [Dav 91].

* SCM está relacionada con el desarrollo de procedimientos y estándares, para manejar la evolución de un producto de software. En esencia está relacionada con el control de los cambios, como manejar sistemas sujetos a cambios y como liberar estos cambios del sistema a los clientes (usuarios) [Som 92].

* SCM es una actividad "protectora" que se aplica a lo largo del proceso de ingeniería del software. Como el cambio se puede producir en cualquier momento, las actividades de SCM sirven para (1) identificarle; (2) controlarlo y (3) garantizar que el cambio se implemente adecuadamente; (4) finalmente informando del cambio a todos aquellos a los que les afecte [Rog 93].

* SCM es un conjunto de procedimientos de ingeniería para seguir y documentar software a través de su ciclo de vida, para asegurar que todos los cambios sean registrados y el estado actual del software sea conocido y reproducido [StGu 94].

* SCM es el arte de mantener un seguimiento de lo que ha cambiado y como son combinados los componentes de un sistema [Jac 94].

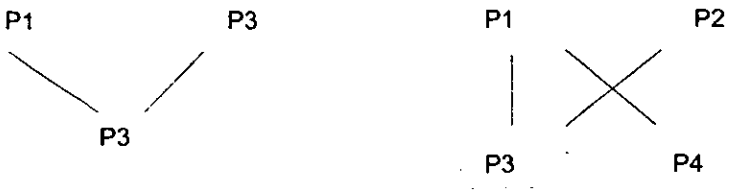
Las definiciones mencionadas sin duda nos enfatizan diferentes facetas que SCM desempeña en la elaboración de un producto de software, y uno que resalta es el control de los cambios al producto software.

En la administración del proyecto de software la SCM marca el camino que hay que seguir en cuanto a los procedimientos que se deben llevar en la producción de éste.

Cuando los productos de software están terminados; éstos sufren evoluciones al paso del tiempo. SCM nos ayuda a que la evolución del producto terminado sea más manejable.

Si deseamos reconstruir un sistema en su versión original, SCM nos dice que elementos de software deben ensamblarse a ese sistema, ya que mantiene la bitácora de cada instancia de ese sistema.

La cuestión de coordinar al personal que labora en el proyecto del software (sistema), también la lleva a cabo SCM, Según [Bab 86], la comunicación se incrementa más rápido que el número de personas, por lo cual si no tenemos un control de ella, la productividad del personal que labora en un proyecto será pobre. Ver figura 1.0.



Canales de comunicación en un equipo de tres personas

Canales de comunicación en un equipo de cuatro personas

Figura 1.0

Podemos notar de la figura 1.0, que para tres personas hay tres canales de comunicación, pero al aumentar al equipo una persona más nos da seis canales de comunicación. Estos canales harán que las personas no sean tan productivas, ya que se gastará más tiempo en coordinar su trabajo. Es obvio que los canales de comunicación los utilizan, para coordinarse. De lo mencionado SCM intervendrá en como organizar a las personas del equipo de desarrollo de software para que se obtenga una mayor productividad en el proyecto (menor pérdida de tiempo en comunicación).

Los problemas típicos según [Bab 86] por los cuales se necesita SCM son:

1. El doble mantenimiento.
2. La compartición de datos.
3. La actualización simultánea.

* El objetivo de usar SCM es asegurar la integridad de un producto y hacer su evolución más manejable. Aunque hay un precio elevado envuelto en usar SCM, se está generalmente de acuerdo que las consecuencias de no usar SCM pueden dirigir a muchos problemas e ineficiencias. Los costos de usar SCM se relacionan con el tiempo, recursos y los efectos en otros aspectos del ciclo de vida del software.

1.1. Actividades principales de SCM.

Las actividades que desempeña SCM en un producto de software (sea en la etapa de desarrollo o funcionamiento de éste), deben poder responder a las siguientes preguntas:

* ¿Cómo debe ser estructurado el sistema para que diferentes versiones puedan ser construidos con requerimientos de distintos usuarios?

* ¿Cómo debe ser guardada una versión vieja del sistema para investigar una falla más tarde?

* ¿Cómo puede una versión del sistema ser construido de manera que contenga ciertas modificaciones, pero no otras?

* ¿Cómo dos programadores trabajarán un elemento de software al mismo tiempo?

* ¿Cómo almacenar de manera eficiente los elementos de muchas versiones?

* ¿Es el mismo sistema en el cuál la falla ha sido reportada?

* ¿Qué ha cambiado en el sistema?

*¿Por qué el comportamiento de este elemento de software no es consistente con la lista de elementos de software del sistema?

* ¿Cuál es el estado de algún elemento de software?

* ¿Cuál ha sido la evolución de ciertos elementos de software?

* ¿Qué código fuente y opciones de compilador deben usarse para construir el sistema?

* ¿Qué partes se deben recompilar cuando un elemento de software ha cambiado?

* ¿Habia errores de compilación o advertencias cuando el sistema fue construido?

* ¿Qué pasos se deben seguir en la construcción del sistema?

* ¿Cuáles son las implicaciones de instalar una nueva versión del compilador?

* ¿Cuál es el procedimiento correcto para evaluación e implementación de un cambio?

* ¿Cuáles fallas están arregladas en una determinada versión del sistema y cuáles son sobresalientes?

* ¿ Fue el cambio probado correctamente?

* ¿Hay cambios planeados para este elemento de software?

* ¿Cuándo deben hacerse los respaldos de los elementos de software para asegurarlos contra perdidas o corrupción?

* ¿Cómo prevenir de accesos no autorizados a los elementos de software?

* ¿Cómo restablecer el sistema a través de un respaldo?

* ¿Cuándo los elementos de software deben ser guardados en almacenamiento secundario?

* ¿Cuándo y cómo debe liberarse el sistema a los usuarios?
[Dav 93]

A las preguntas anteriores les da solución SCM mediante sus cuatro actividades principales que son: identificación de la configuración, control (administración) de la configuración, informe del estado de la configuración y auditoría de la configuración. El control de la configuración en el presente trabajo se divide en dos secciones que son: control de cambios y control de versiones. Se agregan otros temas que están inmersos en SCM y son: plan de la administración de la configuración, la construcción del sistema, la liberación del sistema y la administración de la biblioteca del software.

Hay dos actividades más según [Ron 92] que son: el control de contratos y el control de interfaces. Estas tienen básicamente que ver con agentes externos al desarrollo del sistema o el mantenimiento. Control de contratos, administra los contratos con los proveedores que suministren el software de apoyo al desarrollo del producto. El control de interfaces administra los acoplamientos que se lleven a cabo entre el producto y otros componentes o sistemas de otras empresas. SCM debe asegurar que se respeten los parámetros de las interfaces, tiempos de respuesta y retornos ya establecidos con otros componente o sistemas.

A continuación se explica detalladamente lo que realiza cada actividad de SCM, pero el éxito de una buena SCM se alcanza cubriéndolas todas.

2. IDENTIFICACIÓN DE LA CONFIGURACIÓN.

SCM debe por medio de su administrador, empezar a identificar y ubicar a cada línea de base que se establezca, así como los elementos de la configuración de software para esas líneas de base. Estos elementos de software deberán ser nombrados de manera única y ubicados en una jerarquía de software. Esta forma de nombrar de manera única a los elementos de software, redituará en un buen control de la configuración (cambios y versiones). Veamos algunas definiciones de lo que significa la identificación de la configuración:

* Es la actividad de documentar un esquema de identificación que refleje la estructura del producto [IEEE-1042 87].

* La selección de documentos que identifican y definen la característica de los elementos en cada línea de base de la configuración [MIL STD 973].

* Es la documentación técnica actual o condicionalmente aprobada de un elemento de configuración correspondiendo a las especificaciones, diseño, código u otros documentos [MIL STD 480].

La actividad de la identificación de la configuración es darle un nombre adecuado a cada documento que se produzca en el desarrollo, ya que al hacer el mantenimiento se deberán rastrear estos documentos.

Según [GECC 86], la identificación de SCM tiene los siguientes objetivos:

(1) Definir una estructura de documentación organizada de manera comprensible y previsible.
(2) Suministrar métodos para acomodar revisiones y seguir aquellos cambios que ocurran.
(3) Una correlación de cambios en cuanto a quién, qué, cuándo, por qué, y cómo suceden para facilitar el control.

2.1. Línea de base.

La línea de base dice [IEEE-729 83] que es una especificación o producto que ha sido revisado, que más tarde servirá de base en el desarrollo y mantenimiento, y que puede ser cambiado sólo a través del procedimiento de control de cambios formal.
Estas líneas de base permiten a varias personas trabajar juntas al mismo tiempo. Las disciplinas de SCM giran en torno a la definición y mantenimiento de las líneas de bases.

Una línea de base se refiere a un conjunto de elementos de software que se producen en una fase del ciclo de vida del proyecto. [Ron 92] Menciona que hay cinco líneas de bases generales y son:

(1) Línea de base de la definición. Cuando se tiene la definición de los requerimientos del software.
(2) Línea de base de diseño. Es el punto en el cual se ha determinado cómo construir o codificar para satisfacer los requerimientos del sistema.
(3) Línea de base de código/unidades de pruebas. Es el punto en cual los programas codificados en un lenguaje han sido terminados y son probados, de tal forma que satisfacen los requerimientos de diseño, estos requerimientos a su vez satisfacen la especificación de requerimientos de software.
(4) Línea de base de prueba. Es el punto en el cual las pruebas se han efectuado para demostrar que el producto reúne los requerimientos y esta listo para entregarse al cliente o usuario.
(5) Línea de base de mantenimiento. Es el punto en el cual la forma, adaptación y función del producto entregable puede ser modificado, corregido sus errores y mejorado.

Para SCM las líneas de bases formales usadas son: línea de base funcional, línea de base reservada/asignada (allocated) y línea de base de producto [IEEE-729 83].

Línea de base funcional. Es un producto de la etapa de formulación de concepto del sistema. Es típicamente un documento de requerimientos, una lista de cosas que el sistema llevará a cabo. Este bosquejo se establece en términos que tanto el cliente y el desarrollador del sistema pueden entender, y aprobar en conjunto [Ber 80].

Línea de base reservada/asignada. Documenta precisamente qué funciones serán desempeñadas por los componentes de hardware y software [Ber 80]. También [Ron 92] menciona que es la documentación en la que se rige el diseño e implementación del sistema.

La línea de base de producto. Es el código fuente y documentación relacionada del análisis, diseño y pruebas que definen a los elementos de configuración durante la producción, operación, mantenimiento y fases de su ciclo de vida. La definición incluye las características físicas y funcionales designadas para las pruebas de aceptación y pruebas necesarias para el soporte de los elementos de configuración. Incluye toda la configuración del sistema de software.

Hay también otro término que se encuentra entre las líneas de base reservada y de producto, y es la configuración de desarrollo. Este término es utilizado para evaluar la evolución de la configuración de los elementos de configuración durante su desarrollo. Su control queda sujeto a los desarrolladores. Al final de la auditoría

física y funcional, la configuración de desarrollo pasa ha establecerse como línea de base de producto.

La tabla de la figura 2.1 nos muestra como se integran las líneas de bases generales y las formales de SCM.

General	Formal
Definición.	Funcional y Reservada
Diseño.	Reservada y Configuración de desarrollo
Código/Unidad de prueba.	Configuración de desarrollo.
Pruebas.	Configuración de desarrollo.
Mantenimiento.	Producto.

Figura 2.1 Integración de líneas de bases

Las líneas de base funcional y reservada de SCM comprenden la definición del problema, por lo cual quedan integradas en línea de base de la definición.

La línea de base reservada y configuración de desarrollo describen la línea de base de diseño general. La reservada de SCM se da cuando los desarrolladores analizan los requerimientos de la línea de base funcional y de esta manera empiezan a participar en el diseño porque comprenden las actividades que el sistema hará. La configuración de desarrollo abarca completamente la parte del diseño conceptual del sistema.

La configuración de desarrollo integra las líneas de base de código/unidad de prueba y pruebas. Ya que todas estas fases se dan en la configuración de desarrollo.

La línea de base de producto integra a la línea de base de mantenimiento, porque al establecerse la línea de base del producto en ella se llevan a cabo el mantenimiento del sistema.

2.2. Elemento de configuración.

Los elementos de configuración (hardware/software) son elementos que giran en torno al proyecto, el presente trabajo se enfoca a los elementos de configuración de software (elementos de software). Un conjunto de elementos de configuración definen una línea de base, las siguientes definiciones ayudarán a definir su concepto.

* Es información creada como parte del proceso de ingeniería de software [Rog 93].

* Una colección de elementos de software/hardware tratados como una unidad para el propósito de la administración de la configuración [IEEE-729 83]

Los siguientes elementos de software están sujetos a SCM y constituyen las líneas de base:

(1) Planes del proyecto.
(2) Especificación de requerimientos de software.
(3) Manual de usuario preliminar.
(4) Especificación de diseño.
    a. Descripción del diseño de datos.
    b. Descripción del diseño arquitectónico.
    c. Descripción del diseño de los módulos.
    d. Descripción del diseño de las interfaces.
    e. Descripción de los objetos (si se utilizan técnicas orientadas a objeto)
(5) Código fuente.
(6) a. Procedimiento de prueba
    b. Casos de prueba y resultados registrados.
(7) Manuales de instalación y de operación.
(8) Programas ejecutables
    a. Módulos, código ejecutable.
    b. Módulos enlazados.
(9) Descripción de la base de datos.
    a. Esquema y estructura de archivos.
    b. Contenido inicial.
(10) Manual de usuario final.
(11) Documentos de mantenimientos.
    a. Informes de problemas del software.
    b. Solicitudes
(12) Estándares y procedimientos de la ingeniería de software.

2.3. Jerarquía de software.

La manera de estructurar el proyecto y clasificar sus elementos comienza al establecer la jerarquía de software. Esto es importante para SCM, ya que suministrará la primera vista estructural del sistema y sus elementos. Según [Dav 91] una estructura pobre del sistema hará que un pequeño cambio funcional requiera corregir a varios elementos en varias partes de éste.

2.3.1. ¿Por qué descomponer el software?

Con la jerarquía de software se divide el proyecto para empezar a preasignar nombres a los documentos, mantener el seguimiento de cada bloque de como éste madura, y asignar personal y recursos que llevarán el proyecto. [Dav 91] nos dice que también se alcanzan cuatro objetivos con la descomposición del proyecto y son:

* Manejar la complejidad.
* Dividir las cargas de trabajo. Asignar tareas a las personas
* Producir un sistema mantenible. Cuando aparezcan los cambios el llevarlos a cabo no será complicado.
* Elementos de software de reuso. Tener elementos de software para reutilizar

La figura 2.2 muestra la jerarquía de estructura de árbol de un proyecto.

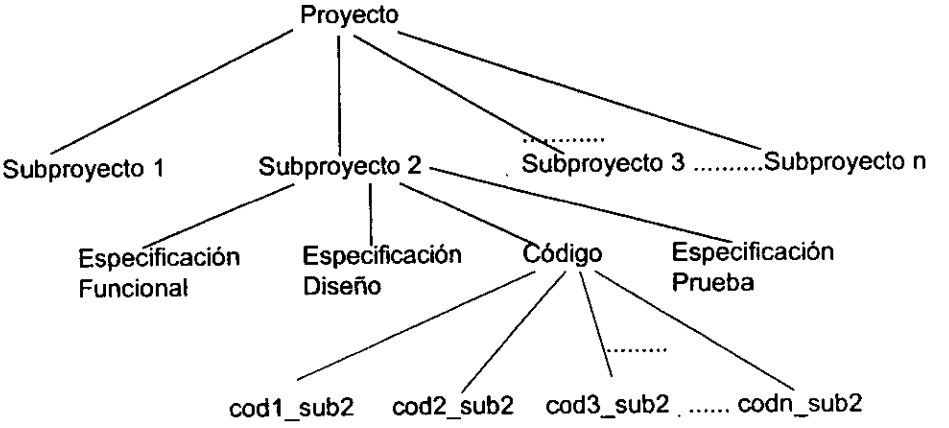


Figura 2.2 Un esquema de jerarquía del software.

2.4. Identificación de los elementos de software.

La identificación de SCM tiene los siguientes objetivos:

(1) Definir una estructura de documentación organizada de una manera comprensible y previsible.
(2) Suministra métodos para acomodar revisiones y ayudar en seguir los cambios que ocurran.
(3) Una correlación de cambios en cuanto a "quién, qué, cuándo, por qué y cómo" para facilitar el control.

La tarea de identificación comienza con la definición de los elementos de configuración de software, que representarán los productos entregables requeridos en cada línea de base establecida. Estos elementos tendrán un nombre que los identifique de manera única. La forma de asignar los nombres es por medio de un esquema de nombre jerárquico o un esquema numérico. El propósito de nombrar de manera adecuada a los elementos de software es para facilitar el control de los cambios. La configuración del software resultante se mantendrá por toda la vida del sistema.

2.4.1 Esquema de nombre jerárquico.

El nombre de un elemento se asigna basándose en la jerarquía establecida, en el ejemplo de la figura 2.2:

proyecto/subproyecto2/codigo/cod1_sub2

El nombre del elemento cod1_sub2 estaría constituido por cuatro partes, que indica de que proyecto es, a que subproyecto pertenece y a que código. La última parte del nombre se le llama nombre relativo.

El nombre relativo se escoge conciso y descriptivo. Esta forma de nombrar ayuda a los programadores, en rastrear los documentos cuando se desea darles mantenimiento.

Un problema que se puede presentar son las colisiones con los nombres en el proyecto, la manera de esquivar ésto es mantener una lista de los nombres que se van generando y al detectar una colisión dar otro nombre. El nombre de un elemento se iguala a un archivo, pero si queremos agrupar varios elementos (clases, funciones o procedimientos) no hay inconveniente siempre y cuando el módulo /unidad/paquete, tenga un nombre adecuado que se refleje en el archivo.

2.4.2. Esquema numérico.

Se asigna un número único a cada documento de la siguiente manera:
(1) Un identificador de proyecto único.
(2) Un identificador para cada elemento de configuración.
(3) Un número para cada de nivel de revisión.
(4) Un código de atributo.

Cada documento que se genere tiene un nombre con una lógica en su construcción la cual el administrador de SCM almacena por medio de una tabla, llevando los consecutivos de cada documento que se produzca. En la figura 2.3, muestra una tabla de como asignar el identificador (nombre) a cada documento.

Algunos ejemplos que se generan con esta tabla.

SIC-0001-P-0-3/93	Es un plan del sistema integral de calzado. El documento es la versión original. Fue colocado bajo el control de cambio en marzo de 1993.
SIC-0001-P-1-5/95	Es la revisión uno para el plan. Este fue colocado bajo el control de cambio en mayo de 1995.
SIP-0020-R-3-8/96	Es el requerimiento 20 del sistema integral de producción, el documento es la revisión tres y fue colocado bajo el control de cambios en agosto de 1996.

Este esquema no dice nada en cuanto a la posición del elemento de software en la jerarquía del software. En cuanto a que tipo de elemento es, sólo con la tabla en mano una persona podría determinar de que tipo de documento se trata.

Número de referencia de documento
XXXX-YYYY-Z-RL-NNN

XXXX-YYYY	Es un identificador común para cada proyecto de software. XXXX.- Es un identificador (prefijo) de proyecto. YYYY.- Es un identificador de componente (consecutivo)
Z	Un carácter que define el tipo de documento. P.- Plan R.- Especificación de requerimiento. D.- Documento de diseño. S.- Código fuente. T.- Documentación de prueba. U.- Manual de usuario. I.- Guía de instalación. M.- Manual de mantenimiento.
RL	Es el nivel de revisión.
NNN	Es un código de atributo (tal como fecha) definida por el desarrollador para reflejar atributos importantes del elemento de configuración.

2.3 Tabla para un esquema numérico.

3. CONTROL DE LA CONFIGURACIÓN.

Mediante el control de la configuración, SCM somete a un control a los elementos de software que han sido o están siendo producidos. La configuración del software se mantiene regulada por medio de esta actividad de SCM. Con el control de la configuración se establecen los procedimientos necesarios para proponer, evaluar, revisar, aprobar e implementar los cambios a una línea de base. Sin estos procedimientos los cambios en lugar de dar solución causarían problema, por lo cual todo cambio a un elemento de software debe ser controlado mediante un procedimiento formal para obtener la autorización para hacer el cambio. El organismo que autoriza el cambio es usualmente conocido como la mesa de control de configuración (CCB). La mesa tiene la facultad de evaluar y autorizar la implementación del cambio al sistema.
Veamos algunas definiciones de control de la configuración.

* Es el control de cambios a la configuración y sus documentos de identificación [MIL STD 973].

* Es la evaluación sistemática, coordinación, aprobación e implementación de todos los cambios aprobados en la configuración de un elemento de configuración después de un establecimiento formal de su identificación [MIL STD 480].

Esta actividad la dividimos en dos para detallarla un poco más, ya que así se aclararan los temas que se vean en esta sección: control de cambios y control de versiones. A continuación se tratan estos dos temas:

3.1. Control de cambios.

Desde que se inicia un proyecto de software, aparecen cambios en éste, ya sea en la fase de desarrollo o durante su mantenimiento. La primera ley de [Ber 80] nos dice: " Sin importar en que momento del ciclo de vida del sistema nos encontremos, el sistema (proyecto) cambiará, y el deseo de cambiarlo persistirá a lo largo de todo el ciclo de vida". Como vemos no podemos librarnos de este asunto de control de cambios en cualquier fase del proyecto. Los procedimientos mediante los cuales controlamos los cambios nos garantizan que en un sistema sean hechos de una manera controlada, de forma que sus efectos en el sistema puedan ser predichos [Som 92].

3.1.1. ¿Por qué el cambio?

Esta respuesta la apoyamos totalmente en [Rog 93] que dice: "El cambio es un hecho vital en el desarrollo de software. Los clientes desean modificar los requisitos. El equipo de desarrollo desea modificar el enfoque técnico. Los administradores desean modificar el enfoque del proyecto. ¿Por qué todas estas modificaciones? La respuesta es realmente muy simple. A medida que pasa el tiempo, todo el mundo sabe más (sobre lo que necesita, sobre el mejor enfoque al problema y sobre como hacerlo ganando dinero). Este conocimiento adicional es

la fuerza motriz de la mayoría de los cambios y nos lleva a sentenciar algo que difícilmente aceptan la mayoría de los ingenieros de software: ¡La mayoría de los cambios están justificados!".
La cita de Presman menciona que todos los involucrados en el proyecto cada vez saben más del proyecto, por lo cual desean hacer modificaciones, para que funcione cada vez mejor.

3.1.2. ¿Por qué controlar los cambios?

Una empresa desarrolladora de software libera un sistema y logra colocar N ventas de éste, por lo cual tenemos N copias del sistema distribuido a los clientes; a su debido tiempo un cliente reporta un error y la fábrica responde enviando la corrección de la queja; otros clientes reportan errores diferentes; la empresa también descubre y establece errores. Muy lentamente empiezan a surgir las variantes en los elementos de software debido a los cambios y estas no serán compatibles en las versiones de algunos clientes. El caos entonces surge debido a las variantes ya que no se sabrá a que cliente entregar su sistema, y esto debido a que el personal de la empresa de software se perderá con tantos cambios. De esto concluimos que la empresa de software debe tener un procedimiento que le facilite controlar los cambios (como manejar los cambios). Además para predecir como se afecta el sistema con un cambio y que se impacta.

3.1.3. Análisis del impacto del cambio.

La mesa de control de configuración/cambios debe juzgar una solicitud de cambio antes de aprobarla o rechazarla. Según [Dav 91], la mesa debe hacerse cuatro preguntas, para dictar su juicio y son:

* ¿Hasta donde se extiende la petición del cambio? Que otros elementos de software salen afectados al realizar esa modificación.

* ¿ Cuál es el costo del cambio? Los que empleará ese cambio (máquina/hombre), además la merma que puede existir en cuanto a presupuesto y plan del proyecto. Todo al final se reflejará en tiempo que se llevará este cambio en su implementación.

* ¿Cuáles son las ventajas y desventajas del cambio? Esta pregunta involucra al ingeniero de software y a los usuarios, los cuales entablan una negociación en cuanto al cambio. Por un lado el ingeniero desea mantener en control el sistema; por otro lado, el usuario desea que el sistema le facilite su trabajo de operación.

* ¿ Qué tan importante es el cambio? Decidir la importancia y urgencia del cambio.

3.1.4. Las bases para el control de cambios.

En el control de cambios, es necesario tipificar los cambios a los que un sistema se someterá, siendo estos: Informal, a nivel de proyecto y formal [Rog 93].

El informal.- Se da cuando todavía no hay una línea de base establecida. Este cambio se hace justificado por el proyecto y los requisitos técnicos (siempre que no impacte en otros requisitos del sistema más amplios que queden fuera del ámbito de trabajo del encargado del desarrollo.).

A nivel de proyecto.- En este cambio el administrador del proyecto debe decidir si el cambio es local o de la mesa de control de configuración si impacta en otros elementos de software. En algunos casos, se dispensa de generar formalmente las solicitudes de cambios, los informes de cambios y la orden de cada cambio; pero hay que evaluar la realización de cada cambio y seguirle la pista.

Formal.- Se presenta cuando ya el sistema está distribuido a los clientes, entonces todo cambio debe realizarse de manera formal siguiendo un protocolo de cambio.

3.1.4.1 El ciclo de vida del cambio.

En el control de cambios, los niveles y formalidad de control requerido varía considerablemente. En proyectos grandes debe existir seguridad estricta en cuanto a examinar los cambios a los elementos de software. El control de cambio debe suministrar siempre lo siguiente:

* Un medio por el cual puede ser solicitado un cambio.

* Un mecanismo, para evaluar el cambio, el cual tome en cuenta el costo y efectos en otros elementos de software.

* Una autoridad responsable, para aprobar o rechazar el cambio.

* Un método de seguimiento al cambio desde su petición hasta su implementación.

La forma de efectuar un cambio está dado por la figura 3.1.

```
PROC CicloDeCambio()
COMIENZA
  Llenar una petición de cambio.
  Analizar petición de cambio
  SI cambio es válido ENTONCES
    Valorar como implementar el cambio.
    Valorar costo del cambio.
    Someter petición a la mesa de control de cambio (CCB)
    SI cambio es aceptado ENTONCES
      REPITE
          Hacer cambio al software
          Someter cambio de software, para aprobar su calidad
      HASTA que calidad de software sea cumplida
      Crear nueva versión del sistema
    OTRO
      Rechazar petición de cambio
    FINSI
  OTRO
     Rechazar petición del cambio
  FINSI.
TERMINA. /* Termina  CicloDeCambio */
```

Figura 3.1 El algoritmo de ciclo de cambio

3.1.4.2. Documento para realizar una petición de cambio.

En general al tipo de cambio que nos estamos refiriendo es el formal, por lo cual al iniciar un cambio necesitamos donde registrar esa petición de cambio. El documento donde solicitamos un cambio se llama **forma de solicitud de cambio.** Ver figura 3.2. Los campos que tiene esta forma se van llenando de acuerdo a como progrese el cambio; aunque si es rechazada la solicitud por la mesa de control de la configuración los campos de implementador y garantía de calidad de software no se llenarán.

La forma de la figura 3.2, puede variar de acuerdo a las necesidades de cada organización (tamaño, forma, datos requeridos, etc.). Se observan cuatro áreas; las cuales se dividen: en el origen de quien pide el cambio, la mesa de control de la configuración que evalúa el cambio, el implementador y el personal de garantía de calidad de software (personal que hace las pruebas a los elementos de software) que autoriza.

La primera área indica a que proyecto o sistema se hará el cambio en su respectiva versión; cuáles elementos de software se desean cambiar; descripción del cambio (que debe de hacer el elemento de software); razones del cambio (si hago ésto, cuales son los beneficios que se obtienen); quien origina la petición del cambio (nombre de la persona) y fecha de la petición del cambio.
La segunda área es para ser llenada por la mesa de control de cambios. En esta área se evalúa si hay razón para el cambio. El cambio se puede aprobar o rechazar; el nombre de la autoridad que valoró el cambio; la fecha en que se recibió y decidió el cambio; valoración del cambio en cuanto a como repercutirá el cambio; prioridad del cambio; costo estimado del cambio (cuántas horas hombres serán utilizadas en el cambio) y comentarios respecto al cambio.
La tercera área la llenará(n) la (s) persona(s) que implante (n) el cambio en el sistema. Contiene el nombre de los implementadores; los nombres de los

elementos de software y sus versiones respectivas que serán modificadas con la petición; la fecha de inicio en que los implementadores iniciaron el trabajo y la fecha de término de la petición; y comentarios respecto al cambio.
La cuarta área indica cuando el departamento de garantía de calidad de software comenzó a revisar los elementos de software, para ver si se ajusta a las especificaciones de la petición del cambio y a los estándares de software. Contiene la autoridad que hizo las pruebas de calidad; fecha de recepción y término de las pruebas; comentarios y la fecha de cuando se entregaron estos elementos de software a SCM.
Realmente la forma de solicitud contiene suficiente información para tener un control adecuado de los cambios y poder hacer el seguimiento de cada solicitud de cambio.

Solicitud de Cambio	Identificador de la solicitud de cambio:
Ser llenado por el solicitante	
Sistema/Proyecto: _____	Versión: _____
Elemento a ser cambiado: _____	Versión: _____
Descripción del Cambio: _____	
Razones para el cambio: _____	
Originador: _____	Fecha: _____
Ser llenado por la mesa de control de configuración	
Solicitud Aprobada/Rechazada	Autoridad: _____ Fecha de Recibo: _____
Valoración del Cambio: _____	Fecha de Decisión: _____
Prioridad del Cambio: Crítico/Muy importante/Importante/Inoportuno/Interesante Costo Estimado del Cambio: _____	
Comentarios:	
Ser llenado por el implementador	
Implementador: _____	
Cambios implementados en elementos: _____	Versiones: _____
Comentarios: _____	Fecha de Inicio: _____ Fecha de Término: _____
Ser llenado por garantía de calidad	
Autoridad : _____	
Comentarios: _____	
Fecha de entrega a SCM: _____	Fecha sometida a garantía de calidad: _____ Fecha de Término:_____

Figura 3.2 Forma de solicitud de cambio

3.1.5. Personas involucradas en una forma de solicitud.

Las personas involucradas en una forma de solicitud de cambio, observando la figura 3.2 se destacan cuatro grupos y son:
* Usuarios (clientes, desarrolladores y vendedores).
* La mesa de control de configuración/cambios.
* Administrador del proyecto.
* Departamento de control de calidad.

Los usuarios encuentran errores y cambios evolutivos al sistema, los cuales se reflejan en una solicitud de cambio.
La mesa evalúa la solicitud de cambio para determinar si es costeable la modificación al sistema.
El administrador del proyecto es la persona encargada del sistema, el cual conoce la infraestructura del software y podrá evaluar el tiempo de implementación de la modificación.
El departamento de control de garantía de calidad de software. Una vez que la modificación fue hecha al sistema, se pasa a este departamento, que somete a prueba los elementos de software que se modificaron, si hay errores o no se ajusta a los requerimientos, se rechaza, regresándolo(s) a los implementadores; y si están bien son pasados a la custodia de SCM.

3.1.6. Reporte de falla.

Hay en la forma de solicitud de cambio mejoras al sistema o errores que se producen en la operación de éste, algunas de estas peticiones son urgentes y otras pueden esperar. Según [Dav 91], la importancia de poder diferenciar entre una mejora y una falla, se logra mediante un reporte de fallas. Este documento se utilizará exclusivamente para reportar fallas del sistema y servirá de mucho en la fase de mantenimiento del sistema.

3.1.6.1. Estructura de un reporte de fallas.

¿Qué debe contener un reporte de fallas?. En primer lugar debe contener la información necesaria para identificar el elemento de software que está provocando la falla (error). En segundo lugar decir los efectos de la falla (que ocasionó). En ningún momento debe sugerir como arreglar la falla [Dav 91]. El reporte de falla debe describir lo siguiente:

* La identificación completa del elemento de software que presenta la falla. probablemente ya esté solucionada ésta en otra versión, o la falla no se ha podido arreglar.

* ¿Por qué está ocurriendo en circunstancias diferentes?

* La naturaleza de la falla. Confrontar los datos que arroja el sistema con la documentación de éste, la cual describe como debe comportarse el elemento de

software.

* La circunstancia en la cual la falla ocurrió. Descubrir de manera precisa y completa las entradas a las cuales el elemento de software respondió incorrectamente.

* El ambiente en el cual la falla ocurrió. Mencionar el tipo de máquina, el sistema operativo, fecha y hora, y otras especificaciones que el implementador considere adecuado.

* Información de diagnóstico. Si el sistema maneja una bitácora de errores, entonces el error que presente un elemento de software nos dirá la bitácora porque fue ese error.

* El efecto de la falla. ¿Por qué ocurrió la falla?, ¿Qué operacion produjo la falla?

3.1.6.2. Procedimientos para el manejo de reporte de fallas.

El procedimiento es importante en la fase de mantenimiento del proyecto [Dav 91]. En esta fase ya es más claro distinguir entre un reporte de falla y una petición de mejora. El equipo de desarrollo de software se divide por lo tanto en dos grupos: uno que repare las fallas y otro que se encargue de las mejoras en otra liberación del proyecto.
La figura 3.3 nos muestra una forma de reporte de fallas que consta de cuatro áreas que serán llenadas por: el solicitante, el investigador, la mesa de control de configuracion y el implementador.

Solicitante.- La llena directamente la persona que detectó la falla en el sistema. Contiene la información de lo que ocurrió.

Investigador.- Se completa con información de la persona que identifica la causa de la falla, propone un arreglo de la falla y hace un costo estimado de reparación de la falla y hace un costo estimado de reparación de ésta. Puede cancelar la falla si el solicitante no logró interpretar correctamente los datos o el estado en el que queda el sistema.

La mesa de control de configuración.- Evalúa la falla y determina su aprobación o rechazo.

Implementador.- Son datos que el implementador pondrá, cuando actúe con los fuentes de cada elemento de software.

Reporte de fallas	Identificador de reporte de falla:
	Será llenado por el solicitante
Sistema/Proyecto : _____	Versión: _____
Elemento de software: _____	Versión: _____
Descripción de la falla (incluir un diagnóstico disponible):_____	
Circunstancia y ambiente de la falla: _____	
Efecto de la falla: _____	
Prioridad: _____ Solicitante: _____	Fecha: _____
	Será llenado por el investigador
Causa de la falla: _____	
Proponer arreglo: _____	
Estimar costo de arreglo (días de trabajo): _____	
Prioridad: _____ Investigador: _____	Fecha: _____
	Será llenado por la mesa de control de configuración
Autoridad: _____	
Reporte de la falla: aprobada/rechazada	prioridad: _____ Fecha: _____
Comentario: _____	
	Será llenado por el implementador
Implementador: _____	
Elementos de software cambiados: _____	Versión: _____
Comentarios: _____	

Figura 3.3

3.1.6.3. Asignación de prioridad a fallas.

Es necesario determinar la prioridad de la falla, ya que así se asignarán los recursos adecuadamente (máquinas y hombres). También nos ayudará a definir los tiempos de respuesta de la mesa de control de configuración al reporte de falla; así como un indicador de la calidad del elemento de software [Dav 91].

Existen varias maneras de establecer la prioridad de una falla, por ejemplo alta, media o baja, o en una escala del uno al diez. Aunque esta escala cumple de una manera general en la asignación de prioridades, no refleja el síntoma de la falla, ya que queda al gusto del que reporta la prioridad. La forma más correcta en asignar prioridad a la falla es en base a las consecuencias que le puede ocasionar al sistema, si no se le atiende a su debido tiempo. Por ejemplo, una falla puede ser

clasificada en base así:

* Detiene al sistema entero.
* Detiene ciertas funciones del sistema.
* Se continua trabajando, pero no fácilmente.
* No tiene efecto significativo sobre el desempeño del sistema.

La prioridad de la falla, la valora el solicitante de la forma de reporte falla, pero la mesa de control de configuración y el investigador dirán la última palabra en cuanto a la misma.

3.2. Control de versiones.

En el desarrollo de un sistema (proyecto) de software, sus elementos van sufriendo transformaciones, los cuales empiezan a diferir en pequeña porciones de código o diferencias ya considerables respecto de su ancestro. Estos mismos elementos de software al sufrir transformaciones generan nuevas instancias del sistema. Este subtema de SCM trata de como manejar estas instancias del sistema. Hay que mencionar que estas transformaciones que sufren los elementos de software se dan cuando ya se está administrando la configuración.

3.2.1. ¿Qué son las versiones?

Utilizando una pregunta del inicio del capítulo 2 de [Bab 86] que dice ¿Cuál es el propósito de un proyecto de desarrollo de software? Su propósito es construir una familia de programas en donde cada miembro de la familia será una versión.

Algunas definiciones de versión:

* Una versión es una instancia de un elemento con sus variantes y revisiones [Dav 91].

* Una versión de sistema es una instancia de un sistema el cual difiere, en alguna manera, de otras instancias. Esta puede incluir una nueva funcionalidad o puede operar en una configuración de hardware diferente [Som 92].

* Una versión, por definición representa una perturbación pequeña a una configuración de software [Ber 80].

3.2.2. ¿Por qué el control de versiones?.

El control de versiones es la actividad de manejar la historia de un componente conforme va sufriendo transformaciones [ThMc 93].

El control de versiones (administración de versiones) combina procedimientos y herramientas para administrar las versiones de los objetos de configuración creados durante el proceso de ingeniería del software [Rog 93].

En el desarrollo de un sistema los elementos de software se pueden modificar, adicionar y borrar sus contenidos. Este proceso continúa mientras no se libere el sistema, el cual utiliza algún elemento de software de los ya producidos. En el momento de la liberación los elementos de software que utiliza el sistema quedan inalterables y no se podrán modificar a menos que se genere una nueva versión. Dado que los elementos de software evolucionan al paso del tiempo de acuerdo a las necesidades de los clientes; si no tenemos un control adecuado de que instancias de elementos de software estamos ensamblando en un sistema, el error que se presentará al funcionar éste, lo que se entregue al cliente funcionará de forma diferente. También debemos saber las transformaciones que ha sufrido un elemento de software y si es posible la línea de base donde se transformó.

Con lo mencionado nos damos cuenta que los tipos de problemas en cuanto a que versión y las transformaciones (historia) de los elementos de software, deben ser controlados mediante algún procedimiento. Esta tarea la lleva a cabo SCM mediante su actividad de control de versiones (administración de versiones).

3.2.3. Revisiones.

Según [Bab 86], dos versiones de un elemento de software pueden diferir porque una es una revisión de la otra. Una revisión es una nueva versión pensada para sustituir una antigua.

El propósito de una revisión se da cuando realizamos un cambio, ya que los elementos de software que se están controlando por SCM, solo se pueden modificar por medio de una revisión. A cada revisión le antecede otra, excepto cuando estamos hablando del primer elemento (la clase inicial que dio origen a las demás). Las revisiones reflejan mejoras en la funcionalidad o un mejor desempeño.

¿Cómo guardamos las revisiones? Se almacenan en un depósito o biblioteca en donde se observa la evolución de estos elementos ver figura 3.4

Cada revisión debe de alguna manera ser mejor que la revisión anterior. Observando la figura 3.4 el ElementoX(2) es mejor que el ElementoX(1).

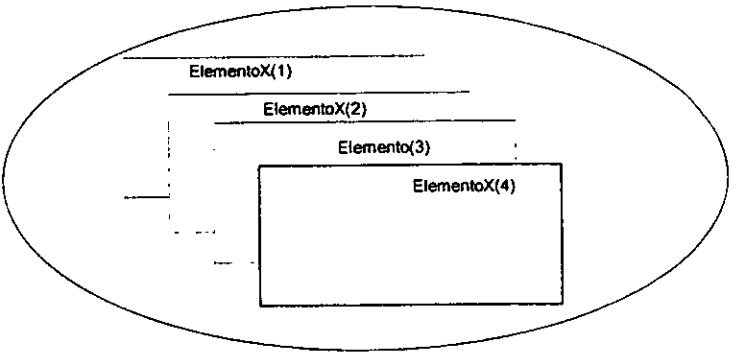


Figura 3.4

Cada versión de un elemento de software puede ser un archivo almacenado en un módulo, paquete o unidad; donde éstas representarán un archivo, el cual es presentado en una biblioteca como un conjunto de versiones de ese módulo y la forma de guardar estas versiones son por medio de: (1) archivos separados, (2) un archivo que almacene la primera versión y las demás versiones se guardan sólo sus diferencias, (3) o en un sólo archivo con compilación condicional. Con los archivos separados, cada versión se almacena en diferentes archivos, pero se presenta el problema del doble mantenimiento.

¿Por qué conservar las revisiones? Existen varias razones para conservarlas y son:

* Si una revisión antigua es parte de un sistema que está en uso; una falla de operación en ésta da origen a hacer las investigaciones de los elementos de software que el sistema tenga, lo cual incluye a versiones(revisiones) anteriores.

* Si se instala una nueva revisión (versión) de un elemento de software en un sistema y por algún motivo salta un error, podemos recurrir a la anterior revisión y reconstruir el sistema de los elementos anteriores.

* Los usuarios se resisten a veces a nuevas versiones, las cuales ellos consideran que son más inestables, y mientras no cambien a ésta (las revisiones anteriores o versiones) es necesario mantenerlas para darles mantenimiento a esos sistemas instalados.

El procedimiento para generar una revisión se muestra en la figura 3.5

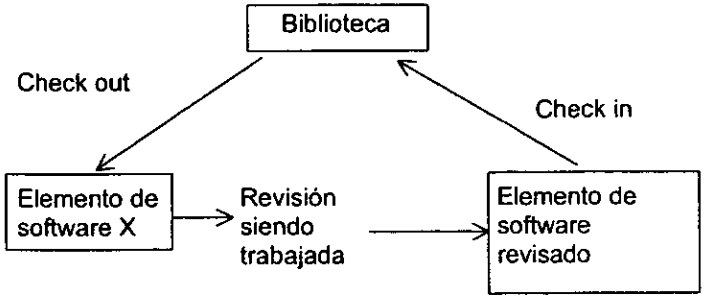


Figura 3.5 Procedimiento para generar revisiones

La figura 3.5 dice lo siguiente:

Paso 1 Se toma de la biblioteca de software, un elemento de software X a revisar.
Paso 2 se trabaja este elemento X (modifica, borra y agrega líneas).
Paso 3 Una vez que el elemento está revisado por el control de calidad, se mete a la biblioteca de software generándose una nueva versión del elemento de software X.

3.2.4. Variantes.

La variante de un elemento de software lleva a cabo la misma función para situaciones ligeramente diferentes y es por tanto pensada para ser alternativa de parte intercambiable. Una configuración típicamente incluye sólo una variante de cualquier elemento de software [Bab 86]. Distinta a las revisiones, una nueva variante no sustituye a una antigua. Múltiples variantes coexisten como alternativas iguales. También una variante de un elemento de software evoluciona y por lo cual existirán varias revisiones de una variante.

Una instancia de un elemento es identificado en base a que variante es y cual revisión es. El conjunto de todos los elementos de software que son variantes o revisiones de cada uno de otro constituyen todas las versiones del elemento [Dav 91]. Ver figura 3.6

De la figura 3.6, podemos decir que el módulo impresión de salida, tiene variantes de destino para las impresoras; Epson, Hp, Star micronics e IBM proprinter; estas serían variantes que dependen del dispositivo para imprimir.

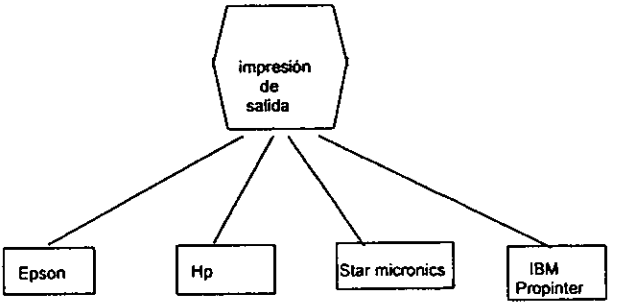


Figura 3.6 Variantes

3.2.4.1. Representación de variantes.

Hay dos formas de representar a las variantes y son:
(1) Segregación. Mantener una copia separada del elemento de software para cada variante.
(2) Fuente simple. Mantener un objeto fuente simple para todas las variantes que son extraídas como se necesiten.

En la segregación hay dos formas de representarla. Una sería a través de directorios del sistema operativo. La otra manera sería almacenarla en un archivo de versiones. El directorio nos presenta problemas de consumo de espacio en disco y doble mantenimiento.

Fuente simple. Todo su código se encuentra en un sólo código fuente, del cual se toma la porción deseada en base a la compilación condicional. Este esquema tiene dos ventajas:
(a) La redundancia de variantes diferentes de un elemento de software puede ser casi completamente evitado.
(b) El uso de espacio del disco es optimizado.
Aunque la forma de rastrear el código de este elemento no sea fácil debido a las macros de compilación que se reflejan por todo el código fuente.

Las variantes se pueden clasificar en temporales o permanentes.

3.2.4.2. Variantes temporales.

Es una variante que más tarde será mezclada con otras variantes del elemento de software. Estas variantes resultan debido a fallas no esperadas, ver figura 3.7.

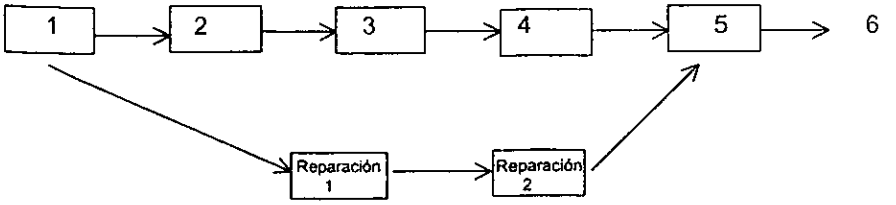


Figura 3.7 Esquema de una variante temporal.

Se está trabajando en la versión 3 del elemento de software de pronto hay una falla de funcionamiento, y a vapor se corrige llevando a cabo una Reparación 1, la cual se le suministra al cliente. Más tarde ya se tiene la versión 3 y aparece la versión 4, aunque también la Reparación 1 evolucionó generando la Reparación 2. El fantasma del doble mantenimiento está presente y se debe quitar, ¿Como? Mezclando rápidamente Reparación 2 en la versión 5. De aquí que sean variantes temporales estos elementos de software.

3.2.4.3. Variantes permanentes.

Pocos sistemas son pensados para un sólo cliente y en un sólo ambiente. La mayoría de los sistemas son utilizados por varios clientes, quienes definen diferentes requerimientos y desean que opere en ambientes diferentes [Dav 91]. Los elementos de software que se produzcan estarán pensados en los recursos y necesidades de diferentes grupos de clientes (por ejemplo sistemas operativos), de aquí que estos elementos al paso del tiempo no pueden ser mezclados, si no, que se dejan como variantes permanentes. Este tipo de variantes se clasifican en:

* Requerimiento variable de usuario.- Cada cliente define un requerimiento común para el sistema que le vende la empresa de software. La empresa con estos requerimientos elabora elementos de software que varía según las necesidades del cliente.

* Plataforma variable.- La empresa de software debe estar al tanto en cuanto a las plataformas que tengan sus posibles clientes. Por ejemplo adecuar el sistema para que funcione en diferentes sistemas operativos.

* Variantes para pruebas y depuraciones.- A los elementos de software para que tengan una mejor calidad, se les ponen precondiciones y postcondiciones, las cuales servirán para hacer las pruebas y depuraciones. El sistema entregable ya no tendrá en el código esas precondiciones y postcondiciones. Estas variantes se deben conservar junto con las otras variantes.

3.2.5. Identificando versiones de elementos de software.

Los elementos de software evolucionan al paso del tiempo debido a cambios

(mejoras, adecuaciones y fallas), por lo cual se debe tener un identificador único de cada elemento de software, su nombre será el mismo, pero el identificador es el que nos hará la distinción entre ellos.

3.2.5.1. Identificando versiones.

La primera revisión de un sistema es simplemente llamado 1.0, subsecuentes versiones son 1.1, 1.2 etc. Con el tiempo se decidió crear la versión 2.0 y el proceso comienza otra vez en revisión 2.1, 2.2, etc. [Som 92]. Ver figura 3.8

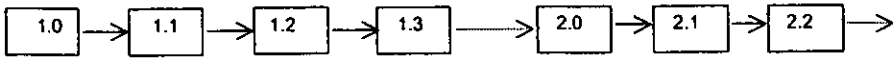


Figura 3.8

3.2.5.2. Identificando variantes permanentes.

Un elemento de software puede existir como un conjunto de variantes para cada una de las varias dimensiones de "discrepancia". Por ejemplo dos discrepancias pueden ser: el sistema operativo y el país donde será usado el sistema. Por lo cual hay dos tipos que describirán las discrepancias de cada elemento en el sistema y son:
País: USA, México, Francia y Alemania.
Sistema Operativo: UNIX, VMS, OS/2.

De aquí que cada elemento existirá como un conjunto de 12= 4X3 variantes. Una por cada combinación de país y sistema operativo.

Identificar estas variantes se llevaría a cabo de la siguiente manera: primero el nombre del elemento por ejemplo, ElementoX, luego agregarle el país (ElementoX.Alemania) y por último el sistema operativo (ElementoX.Alemania.VMS).

Si tenemos la variante ElementoX.Alemania.VMS, sabremos que este elemento correrá para el país de Alemania y en sistema operativo VMS. Por otra parte si hay un ElementoZ independiente del país, pero dependiente de sistema operativo su identificación será ElementoZ.UNIX

3.2.5.3. Identificando variantes temporales.

Según [Dav 91] es conveniente identificar estas variantes agregando un subfijo al nombre del elemento por ejemplo ver figura 3.9

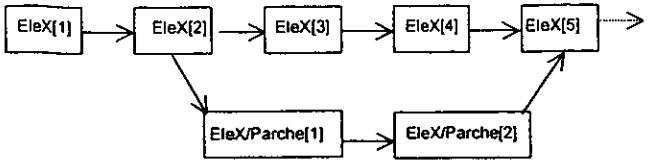


Figura 3.9 Variantes temporales

Aunque surge el inconveniente de que ya no se puede reusar el nombre, al menos en las variantes temporales.

3.2.6. Versión de elementos compuestos.

Los principios de control de versiones son extendidos a los elementos de software compuestos. Los elementos compuestos de software tienen revisiones, variantes temporales y permanentes; y versiones activas. Las versiones están en función de las versiones de cada uno de sus componentes. Ejemplo ver figura 3.10

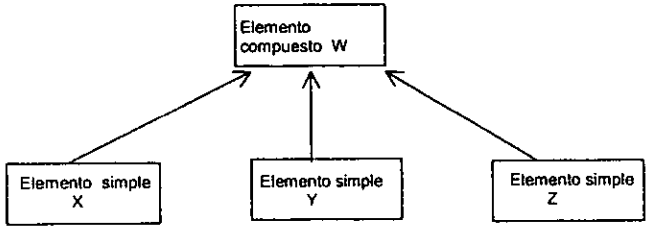


Figura 3.10

El elemento de software compuesto es determinado por los elementos simples X, Y y Z. Estos elementos tienen sus propias versiones, las cuales en conjunto determinarán al elemento W.

Las versiones de elementos compuestos se identifica usando una forma donde se registran los componentes del elemento. La figura 3.11 tomada de [Dav 91] es un ejemplo de tal forma.

Documento de descripción de versiones		
Elemento: Genera Factura		
Parte de: Sistema de facturación		
Versión: UNIX 4		
Estado: Aprobado	Por: CAR	Fecha:: 1/enero/1997
Implementación de solicitud: 77. 81		
Descripción: Primera variante de UNIX de Genera factura		

Componentes	Versión
Pantalla de captura	2.0
Función consecutivo de folio	1.2
Impresión de salida	2.5. HP

Figura 3.11

A la forma se le llama Documento de descripción de versión (Version Description Document) o VDD. Esta documento de descripción de versión contiene la lista de componentes que contiene un elemento compuesto; en este caso pantalla de captura, función de consecutivo de folio e impresión de salida son los componentes del elemento compuesto Genera Factura. El documento de descripción de versión también es utilizado para describir el sistema entero (proyecto), muestra la configuración para cada sistema que se entregue a un cliente.

3.2.7. Derivaciones.

Las derivaciones según [Bab 86], son historias de los programas - como fueron creados. Se pretende dar un panorama en cuanto al registro de la historia de los programas y el uso de estas historias para detectar errores en el programa indicado al depurarlo.

Los códigos fuentes cambian en un tiempo, lo cual nos conduce a una bitácora de cambios de cada fuente. Esta bitácora dice cómo y por qué difiere una revisión de sus predecesoras. Al menos la bitácora tendrá la fecha en que fue revisado el elemento de software, quién fue el responsable por la revisión y por qué el elemento de software fue cambiado. De lo mencionado vemos que el propósito de la derivación es seguir la huella de cada elemento de software en su proceso evolutivo.

La derivación queda determinada por:

* La identificación completa, es decir el nombre y la versión, de cada uno de los elementos fuentes de los cuales el elemento es construido directamente.

* La derivación de cada uno de los elementos de los cuales es directamente construido.

* La identificación completa de la herramienta usada para construirle.

* Las opciones utilizadas que se pasan a la herramienta para construir éste.

* La fecha y hora cuando se uso la herramienta para construir el elemento de software.

Si mantenemos el registro de la derivación de los elementos de software podemos responder a preguntas como:

* ¿Qué ha cambiado en este elemento de software?

* ¿Cuál versión del programa es éste?

* ¿Este listado corresponde a este programa?

La forma adecuada para el registro es colocar su historia en el elemento de software mismo. Es usualmente posible describir un texto dentro del elemento (comentarios). Sería como la cabecera del elemento que nos describiría al elemento de software y su historia. En un elemento derivado describir la cabecera dentro del mismo es más complicado, ya que no dependen de los elementos fuentes, sino, también de las herramientas utilizadas en la construcción (parámetros que se mandan a éstas). Ejemplo del registro de derivación en un elemento fuente:

/* La historia de cambios de la Impresión de Factura */

4.0 RCA 6/May/1997 18:40
Corregir a hoja tamaño carta (Solicitud 450)

3.0 CAS 30/Dic/1996 9:20
Corregir el IVA del 10 a 15 % (Solicitud 200)

2.0 RCA 1/Jun/1996 10:00
Corregir el intercambio de columna entre pieza y precio (Solicitud 150)

1.0 RCA 1/Ene/1996 12:00
Primera versión de subprograma para impresión de facturas. La solicitud surgió debido al proyecto factura.

/* Fin de la historia del cambio */

RCA y CAS son iniciales de las personas que desarrollaron o modificaron el elemento de software.

La historia del elemento fuente comienza desde el cambio más reciente hasta su origen principal, que es la primera versión.

4. INFORME DEL ESTADO DE LA CONFIGURACIÓN.

En el área de SCM hay una actividad que se encarga de reportar el estado del sistema/proyecto que se está desarrollando o dando mantenimiento. El estado de cada elemento de software del sistema es muy subjetivo; sin embargo mediante registros cuidadosos de estos elementos de software en la computadora se conseguirá saber cual es el estado de éstos, pero veamos que es el informe del estado de la configuración.

4.1. ¿Qué es el informe del estado de la configuración y cuáles son sus funciones?

Algunas definiciones:

* Es el registro y reporte de la información que es necesaria para manejar una configuración efectiva, incluyendo una lista de la identificación aprobada de la configuración, el estado de cambios propuestos a la configuración, y el estado de la implementación de cambios aprobados [MIL STD 480A].

* Es la actividad de registrar y seguir sobre los resultados de las actividades de la administración de la configuración como son la identificación de la configuración, el control de la configuración y auditoría de la configuración [Ron 92].

* Es el proceso de mantener registros de las otras actividades para uso en el proceso de SCM [StGu 94].

De las definiciones anteriores se observa que la tarea del informe del estado de la configuración es mantener un registro del sistema/Proyecto en cuanto al desarrollo de éste. [Ron 92] menciona que "básicamente se refiere a las funciones de registro y mantenimiento inherente de las otras actividades de SCM y al sistema de información de administración especializada que debe existir para suministrar toda la información técnica acerca de la configuración del software". Los datos que se registren deben estar en una forma que permita rastrear desde el origen hasta el destino y del destino hasta el origen el software en desarrollo, ya en operación, o en el mercado abierto.

¿Cuáles serían las funciones del informe del estado de la configuración? Citando [StGu 94] encontramos que es "el vehículo por el cual el administrador del proyecto valora los efectos de SCM". Los reportes que suministre incluirán información como cambios propuestos, cambios aprobados y reportes de fallas, ordenados por prioridades. En sistemas grandes definitivamente se tendrá que recurrir a una base de datos computarizada de la información de la configuración del sistema y hacer reportes lógicos del informe del estado.

4.2. La fuente de datos para el informe del estado de la configuración.

Para efectuar el informe del estado de la configuración se necesita determinar la materia prima que genere el producto terminado (reportes o consultas), es decir aquellos datos que se capturarán para mantener el registro del sistema, ¿Pero de dónde se obtendrán estos datos?. Los datos se deben obtener del archivo de desarrollo de software. Dentro de este archivo de registro está contenido toda y cualquier necesidad por saber acerca de una unidad dada o una entidad de nivel alto. La información puede incluir referencias o texto actual de requerimientos, especificaciones, órdenes de cambio y mandatos de cambios. El contenido principal del archivo incluye lo siguiente:

- El primer documento de especificación de requerimientos de software.
- El diseño resultante del requerimiento.
- La lista de código para ir a buscarlo.
- El plan o descripción de pruebas y procedimientos para el módulo, copias de los cambios desde el tiempo que fueron registrados en la biblioteca de software al tiempo que fueron incorporados.
- Todos los datos y reportes de pruebas, junto con registros del diseño y código, otros datos de revisiones y auditorías.
- Referencia o copia de la especificación del producto y documento de descripción de la versión, referencias al manual de mantenimiento y copia del documento [Ron 92].

Los elementos básicos necesarios que deben ser seleccionados, definidos y recomendados en el informe del estado de la configuración o en una base de datos de archivo de desarrollo de software son las siguientes según [Ron 92]:

- Número de especificación.
- Letra de revisión de la especificación.
- Número de identificación del software.
- Letra de revisión del software.
- Número del código del software.
- Número de versión del código del software.
- Número de elemento/equipo.
- Número de identificación del cliente/número de contrato.
- Como construir la letra/número de versión de la revisión.
- Indicador de mantenimiento.
- Indicador de modificación.
- Cambios internos:
    - Fecha de registro.
    - Número de cambio.
    - Descripción.
    - Código o nombre del que pide el cambio.
    - Identificación de otros elementos afectados.

    - Fecha de revisión por la mesa de revisión de software.
    - Mandato y fecha.
    - Nueva letra de la revisión o nuevo número de versión de la revisión.
    - Fecha de la incorporación.
- Cambios propuestos al cliente:
    - Número.
    - Referencia de número de cambios interno.
    - Fecha de anotación.
    - Fecha de revisión por la mesa de control de configuración/cambio.
    - Disposición y fecha.
    - Fecha sometida al cliente.
    - Fecha de retorno de la disposición.
    - Fecha de retrabajo.
    - Fecha de representación (volver a someterlo).
    - Fecha de mandato.
    - Fecha de incorporación.
    - Fecha de liquidación.

- Manual técnico de identificación y revisión.
- Conseguir la revisión o identificación del software de:
    - La identificación de los proveedores/vendedores.
    - Fecha de obtención, pruebas o demos, aceptada.
    - Elemento de software usados.
    - Datos de los derechos de las disposiciones o restricciones.
- Fechas de la expedición de las licencias, si se aplica.
- Identificación y revisión de las herramientas de soporte.

4.3. Determinando la Base de Datos para el informe del estado de la configuración.

La base de datos para el informe del estado puede llevarse en un folder de escritorio como lo hacen las secretarias con su archivero, donde cada folder representará un subconjunto de la información relacionada al sistema de tal manera que un conjunto de folder y de un cajón comprenderá toda la información del sistema; pero como el costo de computadoras está más accesible a toda compañía pequeña, entonces podemos llevar a cabo un sistema apoyándonos en las bases de datos relacionales. El sistema del informe del estado puede estar en una PC aislada o integrada con el sistema de control y administración de versiones el cual almacena los documentos del proyecto/sistema [Som 96]. Lo mejor sería integrar la base de datos en el sistema de control y administración de versiones, pero la mayoría no son compatibles con sus repositorios y ésto limita la integración del sistema del informe del estado.

[Dav 91] menciona que las preguntas a las que debe responder el informe del estado de la configuración son las siguientes:

* ¿Cuál es el estado de un elemento de configuración de software? Mucha gente necesitará saber el estado de diferentes elementos de software (unidad, módulo, programa, elemento de software). Un programador deseará saber si una especificación ha sido completamente aprobada o si un subsistema que se ha ensamblado ha sido aprobado. El administrador del proyecto deseará seguir el progreso conforme los elementos son desarrollados e integrados.

* ¿Ha sido aprobada una solicitud de cambio o rechazada por la mesa de control de configuración? El solicitante de un cambio deseará saber si la mesa ha aprobado su solicitud. Por ejemplo, si una solicitud (petición de cambio) para una mejora a un elemento de software de la biblioteca ha sido rechazada por costosa, entonces el originador de la solicitud debe decidir como trabajar con la rutina como está.

* ¿Cuál versión de un elemento de software implementó una solicitud aprobada? Una vez que una solicitud de mejora a un elemento de software de la biblioteca es implementada, el solicitante del cambio deseará saber cuál versión del elemento de software incluye la mejora.

* ¿Cuál es la diferencia de una nueva versión de un sistema? Una nueva versión de un sistema de software debe estar acompañada por un documento listando los cambios de las versiones previas. Esta lista debe incluir mejoras y reparaciones a fallas. La información es parte del documento de descripción de versión. Cualquier falla que no haya sido arreglada debe también ser mencionada.

* ¿Cuántas fallas son detectadas cada mes y cuántas son arregladas? Las fallas están continuamente siendo detectadas durante el uso operacional de un sistema; las fallas son también reparadas por el equipo responsable de mantener el sistema funcionando. Comparando el número detectado de fallas contra el número reparado, se valorará la eficacia del equipo de mantenimiento y decidirá cuando debe ser hecha una nueva liberación del sistema.

* ¿Cuál es la causa de las solicitudes de fallas? Las solicitudes de fallas pueden ser clasificadas por su causa; por ejemplo, violaciones de estándares de programación, interfaces inadecuadas de usuario, desempeño inadecuado o mal entendimiento de los requerimientos de usuario. Si el administrador del proyecto sabe que muchas fallas tienen una causa similar, se puede tomar alguna acción para detener tales recurrencias de fallas. Por ejemplo, si muchas solicitudes de fallas surgen a causa de mal empleo de áreas de datos globales entonces el administrador del proyecto puede restringir el uso de datos globales.

Agregando algunas de [Som 92]:

* ¿Cuáles clientes han recibido una versión particular del sistema? Los clientes adquirirán el sistema y es necesario saber que versiones tienen para poder mandarles alguna actualización o corregirle un error reportado.

* ¿Cuál es la configuración de hardware o sistema operativo requerido para correr una versión del sistema? El sistema que se embarque forzosamente se debe saber en que plataforma montarlo para su operación.

* ¿Cuántas versiones de un sistema han sido creadas y cuál fue su fecha de creación?

* ¿Qué versiones de un sistema podrían ser afectadas si un componente particular se cambia? Al dar mantenimiento a un elemento de software y si la modificación es sobresaliente, se deberá tener cuidado en cuanto a que otros elementos podrían salir afectados al hacer la mejora o corrección del elemento indicado, porque de lo contrario se propagarían errores a los otros elementos.

Agregando otras observaciones de [MIL STD 973]:

* Identificar la documentación de configuración aprobada actual y el número de identificación asociada con cada elemento de configuración.

* Registrar y reportar el estado de cambios de ingeniería propuesto desde el inicio a la implementación aprobada o contractual final.

* Registrar y reportar los resultados de las auditorías de configuración para incluir el estado y disposición final de las diferencias identificadas.

* Registrar y reportar el estado de todas las solicitudes críticas, las principales desviaciones o pasadas por alto (olvidadas), las cuales afectan la configuración de un elemento de configuración.

* Reportar el estado de la instalación y la efectividad de los cambios de la configuración a todos los elementos de configuración en todas sus localizaciones.

5. AUDITORÍA DE LA CONFIGURACIÓN.

Es necesario que el patrocinador, cliente o usuario sepa la condición en la cual se encuentra el sistema (que se está desarrollando, entregando o en uso operacional), por lo cual se necesitará de alguna manera obtener información respecto al producto de software. En este capitulo se muestra como mediante la auditoría se conocerá el estado que mantienen los elementos de software que componen un producto final de software, en la etapa de desarrollo o de pruebas.

5.1. ¿Qué es una Auditoría?

Una auditoría en cualquier aspecto de la vida es un examen o evaluación aplicada por especialistas ajenos (externos) a una actividad. Y el fin que se persigue es evaluar la situación. Diremos en una auditoría de software, simple y sencillamente que se aplica de igual manera el concepto. Veamos algunas definiciones.

* Las auditorías son un medio por el cual una organización puede garantizar que los desarrolladores han hecho todo su trabajo de tal manera que satisface cualquier revisión externa [IEEE-1042 87].

* La auditoría de configuración es el proceso de verificar y validar el hecho que una configuración propuesta sea completa y consistente [StGu 94].

* El objetivo de la auditoría del software es proveer una evaluación objetiva de productos y procesos para confirmar obediencias a estándares, directrices, especificaciones y procedimientos [IEEE-1028 88].

* La auditoría de configuración de un producto de software desarrollado suministra garantía de que lo que era requerido ha sido construido, como se evidencia por el reporte de pruebas del software, documentación y medio de comunicación [Ron 92].

* La auditoría de la configuración es el proceso de verificar que todos los elementos de la configuración requerida han sido producidos, que la versión actual esté de acuerdo con los requerimientos especificados, que la documentación técnica describa completa y acertadamente los elementos de la configuración, y que todos los cambios pedidos han sido resueltos [IEEE-729 83].

De las definiciones anteriores se percibe que la auditoría de la configuración del software evalúa el trabajo que se está llevando o se ha llevado a cabo en un producto de software. La aprobación que se obtenga por el equipo de auditoría certificará que el producto que se está produciendo, entregando o en uso, cumple con las necesidades, para lo cual se está desarrollando o fue elaborado. Si no se obtiene la aprobación del equipo de auditoría, éste proporcionará una lista de anomalías encontradas y recomendaciones.

El grupo que desempeña la SCM debe preparar lo siguiente para la auditoría:

* Preparar un salón de reunión.

* Preparar mesas y sillas a los participantes en la auditoría para que revisen documentos o sean testigos de demostraciones.

* Preparar recursos humanos de oficinas que estén dispuestos a colaborar a la auditoría, en las reuniones que haya una persona que tome las minutas en una PC (Computadora Personal).

* Preparar una agenda, incluyendo los elementos de configuración a ser auditado en la reunión y un paquete de datos de auditoría como podría ser requerido por lista de requerimientos de datos del auditor en el tiempo prescrito. El paquete deberá contener cambios sobresalientes, desviaciones y olvidos. En la agenda, la comida, los descansos y reuniones deben llevarse a sus tiempos y de manera ordenada.

* Preparar todos los documentos, listas, cintas, disquete, etc. que será requerido por el equipo auditor, de tal forma que se encuentren a la mano.

5.2. Tipos de Auditorías.

El análisis que se lleve a cabo al producto de software por medio de la auditoría verificará que las descripciones de los elementos de software sean los que tiene el producto (código ejecutable) en las especificaciones y documentos, que el paquete revisado esté completo. ¿Pero de qué manera haremos ésto?, ésto se logra mediante dos tipos [IEEE-1042 87] que son: la auditoría de la configuración física y auditoría de la configuración funcional.

5.2.1. Auditoría de la configuración física.

La tarea que realiza la auditoría física consiste en determinar que todos los elementos identificados como parte existente de la configuración estén presentes en la línea de base del producto [IEEE-1042 87]. También debe de establecerse que cada elemento de software contenga la versión y revisión correcta que le corresponde a la línea de base del producto. Estos elementos de software deben corresponder a información contenida en el reporte del estado de configuración de la línea de base. [Ron 92] menciona que auditoría física determina si las especificaciones de diseño y producto, y documentos referenciados representan al producto de software que fue codificado y probado para un elemento de configuración.

De alguna manera la auditoría física garantiza que la documentación entregada con el software, represente el contenido del producto software. Si se hablara de un producto final, el cual empezará a dar resultados con un cliente, la auditoría física estaría garantizando que la configuración del producto que se lleva es la correcta y por lo cual podrá realizar sus tareas con este producto.

La figura 5.1 nos muestra una lista para cotejar la certificación. La primera señala la documentación a ser auditada y la documentación que será requerida para soporte de la auditoria. La segunda parte son las tareas que se desempeñarán durante la auditoría. Algunos de los documentos referenciados y tareas se refieren a elementos de hardware y software juntos.

¿Cómo se lleva a cabo el trabajo de auditoría física? Comienza cuando el desarrollador explica los cambios sobresalientes, desviaciones y olvidos; describe el programa de prueba y cualquier prueba o verificación de prueba sobresaliente; también señala cualquier cambio en la documentación que no ha sido incorporado o aprobado en el tiempo de la auditoría. Los participantes de esta auditoría son los mismos de la auditoría funcional [Mar 94].

Las principales tareas de la auditoría física incluyen:

(1) Para cada elemento de configuración, se hace una revisión completa de la especificación del producto para comprobar integridad y consistencia con la especificación de requerimiento de software, la especificación de requerimiento de interfaces y el documento de diseño de especificación.

(2) Revisar la descripción del diseño para entradas propias, símbolos, etiquetas, referencias de rótulos y descripción de datos.

(3) Comparar las descripciones de diseño de componente de alto nivel para descripciones de diseño de unidad de bajo nivel.

(4) Comparar la descripción de la unidad con la lista de código fuente relacionado para comprobar integridad y exactitud.

(5) Revisar el manual de software para inspeccionar formato, integridad y conformidad con las descripciones de elementos de datos o instrucciones de formato similar.

(6) Inspeccionar el medio de software, disquete, cintas, compact disk, etc. en conformidad a los requerimientos, cotejándolo con el equipo y su relación.

(7) Revisar listas de código fuente para comprobar que cumpla con estándares de codificación establecidos en el arranque del proyecto.

(8) Asegurar que el código fuente presentado produzca el código ejecutable aceptado por la auditoría funcional.

Una muestra de lista para una auditoría física

La siguiente documentación de hardware y software de computadora tienen que estar disponibles y las siguientes tareas tienen que ser realizadas en la auditoría física.

Hardware: Software de computadora: Documentación:	Si	No
(1) Borrador final aprobado de la especificación.	—	—
(2) Una lista describiendo tanto cambios sobresalientes y aprobados contra el elemento de configuración	—	—
(3) Lista de lo que falte totalmente.	—	—
(4) Aceptación de datos de pruebas asociados y procedimientos de prueba.	—	—
(5) Índice del diseño de ingeniería.	—	—
(6) Manuales de averías de partes de operación, mantenimiento y explicación.	—	—
(7) Lista de acciones de la mesa de revisión de material aprobado pasados por alto (olvidados).	—	—
(8) Documento de diseño propuesto forma 250. "Inspección de material y reporte de recepción"	—	—
(9) Aprobada nomenclatura y letreros/placas con nombre	—	—
(10) Copia manuscrita de todos los manuales elemento de configuración de software.	—	—
(11) Documento de descripción de versión de software de computadora.	—	—
(12) Actual conjunto de listas y actualizaciones de descripción de diseño u otros medios de describir el diseño para cada elemento de configuración de software.	—	—
(13) Minutas de la auditoría física para cada elemento de configuración.	—	—
(14) Lista de selección de partes de programas (PPSL).	—	—
Tareas:		
(1) Definir línea de base del producto.	—	—
(2) Validación y revisión de especificaciones.	—	—
(3) Revisión del plan.	—	—
(4) Revisión de resultados y procedimientos de pruebas aceptados.	—	—
(5) Revisión de cambios de diseño no incorporados y faltantes.	—	—
(6) Revisión de desviaciones/olvidos.	—	—
(7) Examinar el DD (documento de diseño) 250.	—	—
(8) Revisar sistema de control de cambios y liberaciones de ingeniería del contratista.	—	—
(9) Revisar el sistema de asignación de documento.	—	—
(10) Revisar manuales de usuario de software, manuales de programador del software, manual de operador del sistema de computadora, manual de soporte de firmware.	—	—
(11) Revisar elementos de configuración de software para lo siguiente: (a) Descripción de diseño preliminar y detallada del componente del software. (b) Requerimiento de interfaces de software preliminar y detallada. (c) Características de base de datos, esquemas de asignación de almacenamiento y características de secuencias y sincronización.		
(12) Revisar el paquete de plan y requerimientos.	—	—
(13) Revisar estado de rectitud en los datos.	—	—
(14) Asegurar que todos los elementos apropiados instalados en el hardware entregable, que tienen que haber sido procesados a través de el PCP (configuración física del producto), están identificados en el PPSL o que la documentación aprobada necesaria esté disponible y que el hardware no contenga elementos que tengan que haber sido procesados a través de el PCP pero no fueron.	—	—

5.1 Un formato de certificación - Muestra de una auditoría física, lista de cotejo

A medida que se realizan las tareas de la auditoría física, aparecen hojas de certificación para cada elemento de configuración, el número de hojas crece cuando el sistema/proyecto es grande; para ver a detalle estas hojas ver [MIL STD 1521A] y [Ron 92]. En proyectos pequeños el equipo de auditoría toma la decisión de no usar las formas, simplemente firmar en un memorándum de conformidad de que la auditoría fue exitosa, todas las tareas fueron completadas, que la línea base del producto ha sido establecida y que se llevaron cualesquiera tipo de acciones correctivas a problemas encontrados.

5.2.2. Auditoría de la configuración funcional.

[Ron 92] menciona que el objetivo de la auditoría funcional es verificar que un actual elemento de configuración se desempeñe de acuerdo con sus requerimientos de software como lo establecido en su especificación de requerimientos de software y especificación de requerimiento de interfaces. Esta auditoría es llevada a cabo antes de la entrega del producto del software y consiste en determinar que alguien acepte haber inspeccionado o probado cada elemento para determinar que éste satisface las funciones definidas en las especificaciones o contratos para el cual este fue desarrollado [IEEE-1028 88]. De hecho antes de hacer la auditoría de configuración física, se debe de realizar la auditoría funcional.

El comienzo de la auditoría funcional es revisar como las pruebas del elemento de software fueron realizadas, como fueron conducidas, y como los reportes fueron preparados y presenciados. También el equipo auditor preguntará respecto a problemas asociados con el programa de prueba para que sea explicado y cualquier prueba incompleta o resultado de prueba rechazado tiene que ser identificable en su inspección. Aquellos documentos de pruebas incompletos o desaprobados se escribirán en un reporte de deficiencias, el cual será la conclusión de la auditoría, este reporte es entregado a los desarrolladores para que proyecten la acción correctiva y de esta manera permitir la aprobación de auditoría funcional.

El administrador de la configuración del software tiene que preparar una área de trabajo en disco donde colocar el producto que será auditado, los auditores ahí verán las demostraciones de pruebas. Las fallas en el software son imprevistas, por lo tanto, los desarrolladores en la auditoría deben estar listos para suministrar soluciones o aceptar un requerimiento de diseño para la corrección del problema. El administrador deberá estar atento que al corregir fallas a los elementos de software se le generen nuevas versiones.

La figura 5.2 y 5.3 son formas de auditoría funcional. La figura 5.2 es una lista de inspección o verificación y la figura 5.3 es un reporte de la auditoría con la lista de resumen de deficiencias encontradas.

Muestra de una hoja de inspección o verificación para una auditoría funcional.

Nomenclatura:

Número de elemento de configuración (CI): _____ Fecha: _____

Requerimientos del contrato	Si	No
1. Lista preparada de desviaciones u olvidos	—	—
2. Procedimientos de pruebas de calificación sometidas	—	—
3. Pruebas de requisitos completada.	—	—
4. Resultados de pruebas de calificaciones compilados y disponibles	—	—
5. Facilidades para conducir la auditoría funcional	—	—
6. Procedimientos de pruebas de evaluación revisados y aprobados	—	—
7. Pruebas de evaluación presenciados.	—	—
8. Pruebas de evaluación de datos y resultados revisados y aprobados	—	—

Comentario: _____
_____
_____
_____

Definiciones:

Comentarios: Una nota explicando, ilustrando o criticando el significado de un escrito. Elementos de esta naturaleza deben ser averiguados por el contratista y/o el gobierno, pero la acción correctiva no es necesaria para llevar exitosamente una auditoría física.

Deficiencia: Las definiciones consisten en dos tipos: (1) condiciones o características en cualquier hardware/software los cuales no están de acuerdo con la configuración especificada; o (2) inadecuada (o errónea) configuración, identificación la cual ha resultado, o puede resultar en elementos de configuración que no cumplen requerimientos operacionales aprobados.

Figura 5.2 Muestra de un formato de certificación para auditoría funcional.

Lista de resumen de deficiencias de la auditoria funcional.

Número de elemento de configuración	Reporte de referencia	Descripción	Acción Correctiva	Lugar de Inspección	Inspeccionado por

Figura 5.3 Reporte de auditoria de software. Varios tipos de reportes son llamados "Software Audit Report". Esta forma es similar a una encontrada en MIL STD 1521A (UASF) y se usa para listar las deficiencias del software encontradas durante una auditoria funcional.

En sistemas complejos o delicados la auditoria funcional debe llevarse de manera incremental, es decir, conforme se completa un elemento de configuración de software éste pasa por el proceso de auditoria funcional. De esta manera se ahorrará grandes recursos, ya que toda la información está fresca o reciente por lo cual es fácil localizarla y cuando se llega al último elemento de software, la auditoria habrá terminado.

El líder del equipo de auditoria deberá realizar las siguientes tareas:

(1) Seleccionar al personal que participará en la auditoria funcional, incluyendo a la persona que ha sido responsable de la prueba de software y la generación de los reportes de pruebas.

(2) Identificar y describir él o los elementos de configuración a ser auditados en una sección dada, así como el estado de cualquier herramienta de prueba de software que pueda ser usada para demostrar el cumplimiento de los elementos de software.

(3) Identificar desviaciones y olvidos pedidos, pero no aprobados, y compilar una lista de cualquier petición de cambio sobresaliente llevado a cabo, pero no aprobado todavía.

(4) Preparar una agenda, y distribuirla al personal de su equipo y clientes (los que pidieron la auditoria) para acuerdos.

(5) Tener un cuaderno de trabajo por cada participante conteniendo una copia de la agenda, las especificaciones de requerimientos de software y las especificaciones de requerimientos de interfaces a ser auditada, las especificaciones de producto de software y la lista de código fuente (si es muy grande incluir la referencia donde localizarlo).

6. ADMINISTRACIÓN DE LA BIBLIOTECA.

Una biblioteca de software se define [IEEE-729 83] como una colección controlada de software y documentos de diseño relacionados que ayudan en el desarrollo de software, su uso o mantenimiento. Los tipos incluídos son biblioteca de desarrollo de software, biblioteca maestra, biblioteca de producción, repositorio de software, biblioteca de programación y biblioteca del sistema. Apoyándonos de [Ron 92] para el desempeño de SCM se utilizan la biblioteca de producción, biblioteca maestra y el repositorio de software. ¿Por qué? La elección es debido a que la biblioteca maestra contiene versiones de la liberación formal de software y su documentación; la biblioteca de producción contiene software aprobado para su uso en operación actual; y el repositorio de software proporciona permanentemente el almacenamiento del software y su documentación relacionada [IEEE-729 83].

[Ron 92] dice que la biblioteca de producción puede ser considerada como una entidad simple para el desarrollo del código de software y como un sitio para retención provisional de tal código durante la codificación y prueba del elemento de configuración de software. En base a la experiencia se menciona que esta biblioteca se divide en dos: una de trabajo y una de soporte del proyecto, para de esta manera poder tener un mejor control e informe del software que esta siendo producido. En la biblioteca de trabajo el programador escribe el código fuente y en la biblioteca de soporte del proyecto se retiene este código que aparentemente está listo para las pruebas del componente (elemento de software).

La biblioteca maestra contiene en calidad de inalterable el código aprobado, liberado y los documentos que se han aprobado y liberado, que serán entregados a clientes o distribuídos al mercado.

El repositorio de software guarda en un lugar físico (bóveda, cuarto, archivo muerto) la información del software y documentación relacionada a éste cuando se termina el proyecto. Se hace para que cuando ocurre un desastre se pueda recuperar el sistema (instalarlo nuevamente).

6.1. Relación entre bibliotecas.

La biblioteca de trabajo y biblioteca de soporte del proyecto están contenidas en la biblioteca de producción. ¿Pero cómo se desenvuelven o relacionan? La biblioteca de trabajo siempre alimentará a la biblioteca de soporte del proyecto. A continuación se describe como se lleva a cabo.

6.1.1. Biblioteca de trabajo.

Es un área de trabajo del programador que se establece al inicio del proyecto para permitir al programador montar sus áreas al desarrollar el código fuente asignado

por el líder del proyecto. El programador codifica el procedimiento (clase o función), lo prueba y corrige en su área. Si sus pruebas son aceptadas, el administrador de SCM transfiere el código de la biblioteca de trabajo a la biblioteca de soporte del proyecto para realizar más pruebas que hará algún miembro del equipo de control de calidad. El programador podrá seguir trabajando en el código producido, ya que le es permitido que lo tenga en su área de trabajo. Hay una observación que se debe tomar en cuenta al meter el código producido a la biblioteca de soporte del proyecto, se hace un bloqueo de manera que el programador no podrá meter alguna mejora, al menos que haya una solicitud de cambio y ésta se apruebe. Al aprobarse la solicitud de cambio se genera una nueva versión, remplazando así a la anterior. Cualquier copia de algún elemento de software se obtiene de la biblioteca de soporte del proyecto y debería anotarse en algún lugar (bloqueando para escritura aquel que lo obtuvo primero y a los restantes se los damos para lectura)

6.1.2. Biblioteca de soporte del proyecto.

En esta biblioteca se encuentra el código producido por la biblioteca de trabajo junto con la documentación detallada usada para producirla. Esta biblioteca está bajo control de SCM, ya que aquí el equipo de control de calidad hará pruebas exhaustivas a estos elementos de configuración de software, así que necesita una solicitud de cambio para generar una nueva versión del elemento de software y ésta se debe de notificar al equipo de control de calidad para que hagan las pruebas sobre la nueva versión.

En la biblioteca de soporte del proyecto se pueden empezar a hacer escenarios de la integración de los elementos de software del sistema, de esta manera el sistema empieza a operar con los elementos de software disponibles ya interconectados (comunicándose). Después se procederá a liberar estos elementos en la biblioteca maestra.

La importancia de tener la biblioteca de producción dividida en dos y su función sea provisional o de paso, ayuda a no llenar la biblioteca maestra con elementos inconsistentes debido a que no cumplen con la calidad requerida, esto no sucede ya que las bibliotecas de trabajo y bibliotecas de soporte del proyecto son un filtro y garantizan que los elementos pasarán por un proceso previo de verificación.

6.1.3. Biblioteca maestra.

[Ron 92] menciona, "que la biblioteca maestra es el punto final de retención (congelación) de todos las unidades, componentes, elementos de configuración y documentos asociados que han sido probados, revisados por una debida mesa de revisión y han sido liberados a la biblioteca maestra bajo el control de configuración".

El control es muy estricto para la biblioteca maestra. Sólo el encargado de custodiar esta biblioteca tendrá acceso de la lectura y escritura, pero también el líder del proyecto o el líder de ingeniero de software puede servir de asistente a esta tarea. Si el producto que se está elaborando o que ya se elaboró es confidencial, entonces no se permitirá a cualquier persona el acceso de lectura o escritura a esta biblioteca.

6.1.4. Repositorio de software.

En el repositorio de software se guarda el producto final (proyecto o sistema) y su documentación asociada, el fin que se persigue al pasar al repositorio la información es porque el proyecto ya no será alterado por un tiempo razonable (más de seis meses). Al archivar el producto de software debe existir una seguridad de que podrá ser recuperado para uso y que lo que está archivado proviene directamente de la biblioteca maestra en la configuración deseada del producto.

[Ron 92] menciona que hay dos problemas que se presentan en el repositorio de software, la recuperación frecuente y el paso del tiempo en el repositorio.

La recuperación frecuente del repositorio, se da cuando la empresa desea reutilizar algunos elementos de software de un proyecto o cuando no se ha estabilizado el sistema y se producen por lo tanto errores seguidos. En ambos casos hay lentitud en la recuperación del repositorio, ya que el volumen de éste es grande y posicionarse en la información indicada consumirá mucho tiempo. Por lo tanto, se necesita un trabajo dirigido que garantice que tales elementos que se coloquen en el repositorio no sean tocados dentro de seis meses o un año.

El paso del tiempo en el repositorio, ocurre cuando las computadoras y sus medios de almacenamiento se van actualizando, por lo cual el repositorio entrará en conflicto, ya que obtener información de él será imposible (si en esa ciudad no hay forma de leer el dispositivo que representa el repositorio de software). La forma de solucionar este problema es que el grupo de SCM; conforme pasen los años, debe ir escalando el repositorio a los nuevos dispositivos para así no enfrentarse al problema.

6.2. Respaldo de bibliotecas.

Los respaldos de bibliotecas contienen duplicados de las bibliotecas activas [Ron 92]. La frecuencia de los respaldos puede ser cada noche, cada fin de semana, o cada mes; la elección queda en función de que tan delicado sea el proyecto e incluso de que el ambiente de desarrollo no sea muy confiable (el ambiente de entorno sería la computadora, la red, el sistema operativo, suministro de energía, herramientas de soporte, etc.), y se pueda de esta manera dañar o perder los

archivos de software. Con el respaldo se puede librar la pérdida o daños que hayan sufrido los archivos.

Los respaldos deben distribuirse en localidades ajenas a las instalaciones donde se esté desarrollando el proyecto, para que en caso de un suceso mayor como incendio del local, se pueda recurrir a estos respaldos y restablecer el software.

6.3. Responsabilidades.

La biblioteca contiene el trabajo intelectual del equipo de desarrollo de software y es el corazón de cualquier empresa, por lo cual debe de ser protegida por personas autorizadas y responsables. La persona que se encarga de efectuar la administración de las bibliotecas recibe nombre de bibliotecario, especialista de soporte de software o especialista de administración de configuración.

Sin importar que nombre se le dé a la persona que administre la biblioteca SCM debe velar porque se cumplan las responsabilidades de la figura 6.1. Otras obligaciones de manera global [StGu 94] las menciona y son:

* Insertar archivos en la estructura del directorio.

* Mantener un control y registro de los respaldos.

* Registrar la estructura del archivo de configuración.

* Mantener una clasificación y catálogos de copias hechas.

Las herramientas que típicamente SCM usa en la administración son las siguientes:

* Control de versiones.

* Un sistema del informe del estado de la configuración.

Descripción del trabajo del bibliotecario.

Responsabilidad	Aplica a (tipo de biblioteca)
1. Establecer y mantener archivos de documentación de proyecto (especificaciones, diseños, planes, listados de computadoras, etc.) en acorde con procedimientos especificados.	Todas
2. Establecer y mantener archivos de computadora de información relacionada y software (computadoras operando) de acuerdo con procedimientos especificados, usando entradas suministradas por programadores y máquinas perforadoras o terminales.	BSP, BM (Si esta en disco)
3. Identificar y obtener suministro y facilidades necesarias para el mantenimiento de archivo (incluye cintas en blanco, etc., cuando aplique a BSP)	Todas
4. Informar a usuarios de procedimientos para someter cambios a archivos.	Todas
5. Cumplir las solicitudes de cambio en una manera ordenada y a tiempo.	BSP, BM
6. Asignar o registrar la identificación para archivos, documentos, reportes de problemas, órdenes de cambio.	Todas
7. Distribuir reportes de problemas y órdenes de cambio.	BM
8. Preparar y emitir reportes sobre el estado del cambio y otras actividades en la cual la biblioteca esté envuelta.	BSP, BM
9. Mantener y distribuir índice de módulos de software de computadora disponible para uso de múltiples proyectos.	RS (Repositorio de software)
10. Mantener planes de bitácoras y retención para archivos del proyecto.	Todas
11. Obtener asistencia a personal administrativo adicional (es decir, soporte con la máquina perforadora) tal como la requieran.	Todas
12. Emitir copias de software solicitado y documentación en acorde con los procedimientos especificados.	Todas
13. Desempeñar otros deberes de soporte que les sean asignados (es decir, las tareas secretariales)	Todas

Figura 6.1 Prescripción de trabajo del bibliotecario del software [EIA CSL]

Biblioteca Maestra –BM.
Biblioteca de soporte de proyecto –BSP
Repositorio de software - RS

La administración de la biblioteca debe dar seguimiento a los siguientes datos:

* Un catálogo de archivos, número de versiones, localización en disco y fecha.

* Copias de todos los documentos de desarrollo y revisiones incluyendo una lista con fechas de ellos.

* Un catálogo de todos los materiales copiados y la fecha en que fueron metidos en la biblioteca del sistema.

6.4. Administración de liberaciones.

El término liberación se denomina a la distribución de software fuera de la organización de desarrollo, por ejemplo a usuarios [Dav 91]. [StGu 94] dice que "es una versión de un CI que es distribuido fuera de la organización de desarrollo". Es la entrega del producto final de un cliente, por lo cual SCM se muestra interesada en que la configuración que se arme de ese producto de software sea la correcta. [StGu 94] menciona también que hay dos tipos de liberaciones: "Liberaciones para desarrollo, las cuales son generadas internamente a una organización de desarrollo; y liberaciones formales las cuales son distribuidas al exterior". Hablaremos aquí de liberaciones formales.

6.4.1 Notas (apuntes) de la liberación.

Las notas describirán información referente al producto que se esta entregando. A grandes rasgos son [Dav 91]:

- Información acerca de un sistema y lo que requiere del ambiente.
- Como leer el medio magnético, como instalar el producto y como correr las pruebas, las cuales verifican que el sistema ha sido instalado correctamente.
- Una lista de fallas conocidas y limitaciones de esta versión del sistema.
- Un resumen de las diferencias entre las liberaciones del sistema anterior.
- La forma en que el cliente debe comunicarse con el proveedor para reportar fallas, solicitudes de mejora y obtener ayudas con los problemas.

La figura 6.2 muestra una solicitud de liberación de versiones. Esta forma es un resumen de todos los cambios de elementos incorporados y nombres de documentos de cambios de programas acompañados a ellos. Es de gran utilidad para informar a la mesa de revisión de cambios de software o a otra organización responsable de los contenidos exactos de la nueva liberación.

Solicitud de liberación de versión	Prefijo del sistema	Fecha de elaboración	Número de versión

Autoridad de aprobación	Firma:

Solicitud de modificación de programa (lista de todas las solicitudes de modificación a ser cargadas en esta versión.)	Documento de cambios del programa (lista de todas los documentos de cambios a ser cargados a esta versión.)

Figura 6. 2 Solicitud de liberación de versión.

7. CONSTRUCCIÓN DEL SISTEMA.

[Som 92] menciona que la construcción del sistema o construcción de la configuración del sistema es "el proceso de combinar los componentes de un sistema en un programa, el cual se ejecuta en una configuración para un objetivo específico". En este proceso está incluída la compilación y ligado de componentes. El compilado traslada los programas fuentes que están en un lenguaje de alto nivel a código objeto [IEEE-729 83]. El ligado es un proceso mediante el cual los códigos objetos de los programas son agrupados en un archivo ejecutable (EXE). Los componentes pueden ser simples o compuestos.

SCM debe poner mucho cuidado en la construcción de la configuración del sistema, ya que las partes que se están ensamblando producirán un ejecutable del sistema diferente de acuerdo a las variantes o las versiones de los elementos de software que se ensamblen.

7.1. Proceso de construcción.

[Dav 91] propone para construir una configuración que se tengan:

* Los elementos fuentes. El código fuente con el que se construye la configuración.

* El modelo del sistema. Una descripción de la relación de los elementos que se necesitarán para construir la configuración requerida.

* Selección de versión. Una selección de las versiones de cada uno de los elementos que serán usados para construir la configuración dada.

* Elementos derivados. Un depósito de elementos derivados, que ya han sido construídos y por tanto no necesitan ser construídos nuevamente, sino deberán ser incluídos.

La figura 7.1 ilustra los procesos que se siguen en la construcción de una configuración deseada (sistema).

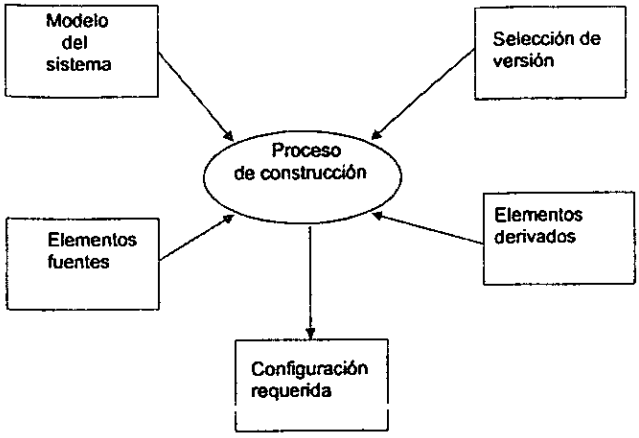


Figura 7.1 Proceso de construcción.

7.2. Construcción incremental.

Un sistema grande tiene miles de elementos de código fuente y si un elemento cambia, habría que compilar todos éstos y entonces hacer el ligado para dar el nuevo programa ejecutable. La solución sería compilar sólo uno o algunos que hubieran cambiado, a esta forma se le llamaría construcción incremental.

Una falla en la construcción incremental, en cuanto a que se olvide recompilar los elementos deseados (pasados por alto), daría por resultado un producto final erróneo. ¿Pero de qué manera se construirá el sistema de forma que se aminoren los errores de construcción? Esto se puede lograr auxiliándose del sistema operativo (SO) en el que se esté desarrollando y con Make. Esto es útil cuando la empresa de desarrollo no tiene suficientes recursos para encontrar una buena herramienta de SCM.

7.3. Usando facilidades del sistema operativo.

El sistema operativo (SO) permite construir archivos de comandos y una estructura de directorio jerárquico. Estos dos apoyos se pueden usar para crear configuraciones que más tarde puedan ser recuperadas en la construcción del sistema.

7.3.1. Archivo de comando de construcción.

En la primera versión del proyecto el archivo tendrá una estructura sencilla, pero a medida que se generan nuevas versiones, la complejidad de la estructura se complicará a tal grado de tener varios archivos para diferentes versiones, la

estructura se va perdiendo y su mantenimiento se hace muy complicado. Por lo cual como todos los programas de desarrollo, este archivo deberá ser cuidadosamente diseñado. Un archivo de comando de construcción es un archivo que contiene comandos del sistema operativo, los cuales construyen elementos derivados.

Hay tres principios para escribir y estructurar el archivo de comandos de construcción: que sea parametrizable, automatizado y que maneje errores.

Parametrizable. El archivo de comando debe permitir al usuario mandar uno o varios parámetros desde el teclado. Estos parámetros ayudarán a elegir la configuración deseada del sistema. Algunos lenguajes de alto nivel aceptan compilación condicional, la cual ayuda a discriminar líneas del código del fuente.

Automatizado. El procesamiento que haga el archivo debe requerir lo menos posible de la intervención humana y donde sea necesario recibir datos externos (aquéllos que capturen) debe validar estos parámetros. No se debe permitir que solo un miembro conozca el mantenimiento del archivo de comando, si no que varios miembros del equipo (pudieran ser todos) deben saberle dar mantenimiento o saber utilizarlo en la construcción.

Manejo de Errores. El archivo de comando debe poder recuperarse de los errores que le ocasionen los parámetros recibidos, mala compilación de elementos, y problemas de ambiente (disco lleno, no se encuentra el elemento). Al ocurrir el error en la construcción el archivo debe de:

* Detectar el error y responder a éste. Aquí podría abandonar la construcción o continuar verificando sin generar el ejecutable.

* Todos los errores se registren.

* Restablecer la versión anterior de tal forma que no se pierda la anterior configuración que tenía la biblioteca.

El siguiente ejemplo es un archivo de comando para un programa calculadora:

```
COMPILE        Multiplicar.c
COMPILE        Dividir.c
COMPILE        Restar.c
COMPILE        Sumar.c
LINK      Multiplicar.obj, Dividir.obj, Restar.obj, Sumar.obj EXE = Calculadora
```

7.3.2. Usando directorios que contengan la configuración del sistema.

La figura 7.2 muestra una estructura simple de un directorio el cual es adecuado para retener configuraciones de un sistema pequeño.

El directorio DES contiene las últimas revisiones de todos los elementos en el sistema. PRU contiene las últimas revisiones de cada elemento que ha sido aprobado. Los directorios 1.1, 1.2, ...2.1... Contienen las versiones congeladas del sistema.

Con esta estructura de directorio el archivo de comando puede ir por los elementos a trabajar o tener un sólo archivo de comando en cada directorio y de esta manera se obtiene la construcción adecuada del sistema.

La trayectoria de búsqueda es muy útil al programador, para realizar pruebas en su directorio de trabajo. Se coloca la trayectoria de búsqueda donde se encuentren las versiones congeladas del sistema, de esta manera al buscar los componentes restantes, el archivo de comando sabrá donde buscarlo.

Para proyectos complejos se descompone en subsistema haciendo más manejable el sistema, ver figura 7.3 .

El archivo de comando debe ser parametrizado de manera que la elección de una versión del subsistema se haga sólo una vez. La asignación de las versiones que serán tomadas de cada subsistema en los parámetros en el archivo, se deben asignar antes de cualquier instrucción del archivo de comando.

Lo interesante es que se puede jugar con las combinaciones de versiones del subsistema. Y se puede reutilizar el código de alguna versión del subsisistema en otro proyecto.

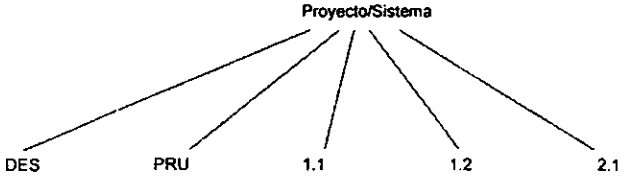


Figura 7.2 Una estructura adecuada para un pequeño sistema.

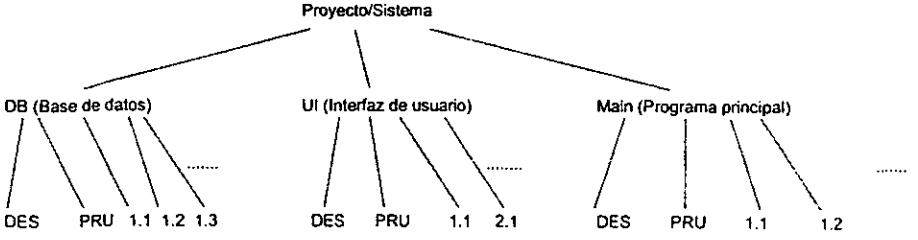


Figura 7.3 Una estructura directorio para subsistemas.

7.4. Herramienta Make.

Make es una herramienta pionera de apoyo en SCM. ¿Cuál fue su contribución en SCM? Menciona [Fed 79] "Make fue originalmente escrita porque su autor olvidó compilar parte de programas antes de integrarlos".

Make reconstruye sólo los elementos que hayan cambiado. La forma de construir de manera incremental se basa en la comparación de la fecha y hora de los elementos que contiene gracias a que los SO incrustan en los archivos la hora y fecha, la cual sirve a Make para realizar sus comparaciones.

El ejemplo siguiente es un archivo make (Makefile) del archivo de comando calculdora:

```
Calculadora: Multiplicar.obj Dividir.obj Sumar.obj Restar.obj
        Link Multiplicar.obj, Dividir.obj, Sumar.obj, Restar.obj EXE = Calculadora

Multiplicar.obj: Multiplicar.c  Constante.h
        Compile  Multiplicar.c

Dividir.obj      : Dividir.c  Constante.h
        Compile  Dividir.c

Sumar.obj        : Sumar.c
        Compile  Sumar.c

Restar.obj       : Restar.c
        Compile  Restar.c
```

El Makefile expresa, las dependencias correctas y proporciona al ligador y al compilador los comandos necesarios para verificar dichas dependencias. Un comando Link enlaza todos los objetos "obj" y el resultado del ligado será un "EXE". La línea del COMPILE menciona que compila los elementos si en su comparación cambiaron de fecha y hora.

El Makefile es creado una vez por el administrador de configuraciones, la estructura del módulo se diseña y no necesita ser recreado en cada invocación de Make. El Makefile sólo cambiará cuando las estructuras de dependencias de algún módulo cambie. [Bab 86] menciona que "el uso de Make es más eficiente cuando el software se diseña con una buena estructura de interfaz. Los archivos de interfaces deben ser pequeños de manera que los cambios afecten a pocos módulos". El archivo interfaz contiene constantes o estructuras que utilizan cada módulo. En el ejemplo de Makefile "Constante.h" representa un archivo de interfaz.

8. PLAN DE LA ADMINISTRACIÓN DE LA CONFIGURACIÓN DEL SOFTWARE.

El éxito de un buen control de SCM, radica en que tenga un buen plan para la administración de la configuración del software [Som 92]. Por lo cual es necesario hacer un buen plan de SCM antes y durante el desarrollo del proyecto. Las siguientes citas nos ayudarán a definir una idea lo que debe tener el plan.

* Es un documento que una organización usa para definir los métodos y tiempos en que llevarán las actividades de SCM [StGu 94].

* Documenta por medio de un escrito los métodos a ser usados para identificar los elementos de software, controlar e implementar los cambios de éstos, y registrar y reportar el estado de la implementación de los cambios [IEEE-828 90].

* Es el documento que describe como un proyecto administrará las configuraciones. Define el quién, el qué y el cuándo de SCM. El plan de SCM puede ser parte del plan de calidad del proyecto y si es grande el proyecto el documento se maneja por separado [Dav 91].

* La tarea principal de los procesos de SCMP es decidir exactamente cuales elementos (o clase de elementos) serán controlados, por medio de un documento [SOM 92].

De las citas anteriores se percibe, que el plan apoya en cuanto a definir el trabajo de un software; distribuir las cargas de actividades en el proyecto, en base a que papel desempeñará cada persona involucrada en el proyecto. Nos ayuda también a ir seleccionando los documentos más relevantes del proyecto, los cuales deberán ser puestos en custodia de SCM. en este punto se define la jerarquía que habrá de los documentos respecto al proyecto.

Para estructurar el documento del plan de SCM nos apoyaremos en el estándar de la IEEE número 828 de 1983 y los temas que debe tener son:

* Introducción.
* Administración.
* Actividades de SCM.
* Herramientas, Técnicas y Metodología.
* Control de proveedores.
* Registro de conservación y colección.

8.1. Introducción.

Esta sección debe proveer un bosquejo del plan resumiendo los siguientes puntos:

8.1.1. Propósito.- Definir de manera específica el propósito del plan.

8.1.2. Alcance.- Debe identificar los elementos del software a ser producidos y usados, las organizaciones, las actividades y las fases del ciclo de vida del software que el plan aplicará.

8.1.3 Definiciones y acrónimos.- Se debe definir o proveer, en esta subsección, una referencia de definiciones de todos los términos y acrónimos requeridos para interpretar de una manera adecuada el SCMP.

8.1.4. Referencias.- (1) Suministrar una lista completa de todos los documentos relacionados de otra parte en el SCMP; (2) identificar cada documento por título y número de reporte; (3) especificar las fuentes en que los documentos relacionados pueden ser obtenidos.

8.2. Administración.

Se describe en esta subsección la organización y las responsabilidades asociadas.

8.2.1. Organización.- Básicamente detectar la jerarquía de la institución y como quedan en un diagrama las personas involucradas en el desarrollo del sistema (proyecto) y el (los) cliente (s) que intervendrán en el trabajo del proyecto. Aquí también se asigna a las personas que estarán involucradas con SCM. De esta manera se distribuye las tareas que tendrá cada persona involucrada en el proyecto.

8.2.2. Políticas, Seguimiento y Procedimientos.- En esta subsección como su nombre lo indica se establecen las políticas, seguimiento y procedimientos que debe tener el plan. Por ejemplo: Identificar los niveles del software en una jerarquía de árbol; nombres convencionales, para módulos y programas; designación de los niveles de versiones; métodos de identificación de los productos de software; procesos de liberación de documentos; liberaciones de productos de software a un depósito (biblioteca del software); estructura y operación de las mesas de control de configuraciones; liberaciones y aceptaciones de los productos de software; la manera de hacer auditorías por parte de SCM y niveles de garantía de la calidad.

8.2.3. Control de interfaces.- Definir la manera de como llevar a cabo las interfaces de tal manera que su costo sea mínimo. El sistema que se desarrolle interactuará con usuarios y otros sistemas, por lo cual hay que hacer buenas interfaces que se deberán cumplir con un cierto estándar que se define aquí.

8.2.4. Implementación del plan.- Lo que se pretende hacer en esta subsección es establecer los principales objetivos que se deben de llevar a cabo para implementar el plan. Los objetivos incluyen lo siguiente:

(1) La mesa de control de configuraciones.
(2) La configuración de cada línea de base.
(3) Programas y procedimientos, para revisiones y auditorías de SCM.

8.3. Actividades de SCM.

En esta sección se describe como se deberán cumplir los requerimientos establecidos por SCM cumplidos. Las actividades serán:
(1) Identificación de la configuración.
(2) Control de la configuración.
(3) Informes del estado de la configuración.
(4) Revisiones y auditorías de la configuración.

8.3.1. Identificación de la configuración.- Esta subsección identificará las líneas de base del sistema y las relaciones entre ellas; bosqueja los nombres que llevarán cada elemento de software que se produzca.

8.3.1.1. Los elementos de software que formarán una línea de base.

8.3.1.2. Los sucesos de aprobación, revisión y los criterios de aceptación asociados con establecer cada línea de base.

8.3.1.3. Participación de usuarios y desarrolladores en establecer las líneas de base.

8.3.2. Control de la configuración.- Esta subsección describe los niveles de autoridad, métodos, mesas de control de configuraciones y procedimientos en cuanto a los cambios que sufrirán los elementos del software y por ende el producto final.

8.3.2.1. Describir los niveles de autoridad, para aprobar los cambios a ser usados en cada fase del ciclo de vida del software.

8.3.2.2. Definir los métodos a ser usados, para procesar los cambios propuestos, para establecer configuraciones. Esto se logra haciendo:
(1) Identificando la ruta - procedimientos que deben de seguir- de los cambios propuestos de cada una de las fases del ciclo de vida.
(2) Describir los métodos de implementación de cambios aprobados propuestos.
(3) Describir los procedimientos para el control del depósito (biblioteca de software).
(4) Si los parches deben ser utilizados para cambiar el código objeto, describir los métodos para identificar y controlar éstos.

8.3.2.3. La mesa de control de configuración/cambios u otro cuerpo de administración de cambios debe de hacer lo siguiente:
(1) Definir el papel de cada uno
(2) Especificar sus autoridades y responsabilidades.
(3) Establecer la resolución de los desarrolladores y los usuarios a la mesa de control de configuración.

8.3.2.4. Establecer los métodos que serán utilizados en el control de interfaces con programas y proyectos, mas allá del alcance del plan.

8.3.2.5. Establecer los procedimientos de control para productos de software especiales asociados al proyecto.

8.3.3. Informe del estado de la configuración.- Este subtema se enfoca a saber en que etapa los elementos de software se encuentran en la producción. Esto se lleva a cabo de la siguiente manera:
(1) Establecer un orden de como la información en los estados de los elementos de configuración (elementos de software) serán colectados, verificados, almacenados, procesados y reportados.
(2) Identificar los reportes periódicos, que serán proporcionados y su distribución
(3) Establecer que capacidades de búsqueda dinámica serán suministradas.
(4) Describir los medios que serán utilizados para implementar algún requerimiento del informe del estado especial requerido por el usuario.

8.3.4. Revisiones y auditorías de la configuración.- Esta subsección se encarga de manera especial en saber como está la producción del software si se está produciendo de acuerdo a lo establecido. Para implementar este punto en el plan se realiza lo siguiente:
(1) Definir el papel de SCM en auditorías y revisiones que serán realizadas, en puntos específicos en el ciclo de vida del software.
(2) Identificar los elementos de configuraciones que serán cubiertos en cada una de estas auditorías y revisiones.
(3) Establecer los procedimientos que serán usados, para la identificación y resolución de problemas que ocurran durante estas auditorías y revisiones.

8.4. Herramientas, Técnicas y Metodología.

Esta sección debe identificar, establecer los propósitos y describir el uso de las herramientas de software específica. Las técnicas y metodología que se utilizarán para soportar SCM en el proyecto específico.

8.5. Control de proveedores.

Establecer en esta sección los acuerdos que se llevarán con los proveedores y desarrolladores, de tal manera que nos aseguren los requerimientos para el proyecto. Los puntos que se deben de considerar son:
(1) Métodos, para controlar subcontratos y proveedores y en base a quienes pueden repercutir en el plan.
(2) Explicar los métodos a ser usados.

8.6. Registro de conservación y colección.

Se encarga de saber que documentación de la SCM debe ser conservada. También determina los métodos y facilidades en cuanto a reunir, respaldar y mantener esta documentación. Y designar el periodo por el cual se conservará esta documentación.

8.7. Errores comunes en un plan.

Según [Ron 92] hay errores comunes que se llevan al elaborar un plan y son:

(1) Una adecuación del plan de otro proyecto.
Resultado. - Los requerimientos del proyecto actual no se adecuarán a éste.
(2) Plan escrito para propuesta, pero no actualizado en concesiones de contrato.
Resultado.- No refleja los requerimientos nuevos/modificados que se negociaron con el cliente. También ha sido escrito más para vender SCM en vez de suministrar la dirección necesaria al proyecto en implementar SCM.
(3) Las tareas/programas de SCM no mantienen una fecha con los cambios del actual proyecto.
Resultado.- Suministra información engañosa al proyecto en preparar y responder a programa de revisión, fechas de auditorías y sometimiento a éstas.
(4) Árbol de documentos incorrecto y/o jerarquía de software.
Resultado.- Puede ser que los documentos más o menos sean nombrados adecuadamente.
(5) Flujo de procedimiento de proceso de cambios de software no está completo o es vago o no está estricto.
Resultado.- El proceso de cambio es costoso y pobre, lentitud en llevarlo a cabo o evasión del proceso porque es muy estricto. Causa pérdida de tiempo, mala voluntad dentro del proyecto.
(6) Los requerimientos para establecer el informe del estado de la configuración, el mantenimiento y los reportes son vagos o inconclusos.
Resultado.- El registro de los elementos de datos variará y los formatos de reportes no son consistentes. Los registros no se guardan con el fin de asegurar una exactitud en las consultas de la información del proyecto.
(7) No suministra control a proveedores cuando éstos son conocidos o ya estaban contratados antes de iniciar el proyecto.

Resultado.- Los proveedores son dejados a sus propios métodos, los cuales pueden no ser compatibles con el plan de SCM del proyecto. La documentación del software y su código será recibido sin control y causará muchos problemas de integración.
(8) No se previó una forma de archivar o almacenar el proyecto finalizando sus etapas.
Resultado.- Pérdida de información o pérdidas de tiempo en localizar ésta. El reuso se ve afectado, problemas para reconstruir una demostración o para reutilizar parte del código.

9. HERRAMIENTA DE AYUDA EN LAS ACTIVIDADES DE SCM.

La herramienta que se escoja para implementar los procesos de SCM en un proyecto debe describirse en el plan de SCM y necesita ser compatible con el ambiente en que se hace el desarrollo o mantenimiento. También se necesita saber si soportará el control de los fuentes en el lenguaje de programación con que se esté desarrollando.

[StGu 94] menciona que "la mayoría de herramientas enfatizan el control de configuración (control de versión) en vez de las principales actividades de SCM". Aunque se tenga la mejor herramienta, no se garantiza automáticamente que será llevada a cabo eficazmente SCM. El encargado de SCM no debe caer en el error al pensar que las herramientas mismas constituyen la SCM. Más bien los conceptos fundamentales de las herramientas deben ser dominados.

9.1. Clasificación de herramientas.

Hay varias maneras de clasificar las herramientas para SCM, pero nos apoyaremos en [IEEE-1042 87] para mencionar la forma de como agruparlas, de hecho se hace cita textual de este estándar. La clasificación se da en base a como las herramientas se integran en el ambiente de ingeniería de software del proyecto. El actual conjunto de herramientas de SCM se clasifican en términos del nivel de automatización que suministra al ambiente de programación del proyecto. El agrupamiento es en: básicas, avanzadas, en línea e integradas.

9.1.1. Básicas.

Estas herramientas controlan los programas de la biblioteca de desarrollo que se encuentra en el disco duro. Hacen distinción entre cuales elementos de software (unidades o componentes) están o no están controladas. Las herramientas de este tipo simplifican y minimizan la complejidad, el tiempo y los métodos necesarios para generar una línea de base.
Esta herramienta debe ser compatible con un ambiente de programación que no sea complejo.

Este grupo cuenta con las siguientes características:

(1) Sistema de administración de base de datos básicos.
(2) Generador de reportes.
(3) Medios para mantener separada la biblioteca maestra y la biblioteca de área de trabajo del programador.
(4) Un sistema de archivos para manejar los Check-in y Check-out de las unidades para controlar compilaciones y capturar los resultados de los productos.

9.1.2. Avanzadas.

Este grupo está capacitado para apoyar a un equipo de SCM en sus actividades, las cuales se llevan a cabo en proyectos grandes y complejos. Se asume que se dispone de un ambiente de programación donde hay suficientes recursos de computación.

Incluye las siguientes características:

(1) Las características de las herramientas básicas.
(2) Programas de control de código fuente que mantendrán la historia de revisiones y versiones.
(3) Programas de comparación para identificar (y ayuda de verificación) cambios.
(4) Incluye herramientas para construir o generar código ejecutable.
(5) Un sistema de documentacion (procesador de palabra) para introducir y mantener las especificaciones y archivos de documentación de usuario asociado.
(6) Un sistema de solicitud de cambios al software, un sistema de seguimiento de autorizaciones que hace órdenes para cambios fáciles de leer en la máquina.

9.1.3. En línea.

Estas herramientas interactuan constantemente con el proyecto, su acceso es en cualquier instante a la biblioteca del proyecto; por lo cual es necesario contar con recursos óptimos de cómputo para que no se formen cuellos de botella en las consultas, actualizaciones y creaciones.

Incluye las siguientes características:

(1) Herramientas genéricas de las herramientas avanzadas integradas, así ellas trabajan desde una base de datos común.
(2) Un sistema de control y seguimiento de solicitud de cambio de software/autorización de cambio de software que trae la creación, revisión y aprobación de cambios en línea.
(3) Generadores de reportes de trabajo en línea con la base de datos común, y un sistema de solicitud de cambio de software/autorización de cambio de software que permite al grupo de SCM generar respuestas a consultas en línea.

9.1.4. Integradas.

Debido a que estas herramientas integran las funciones de SCM con el ambiente de ingeniería de software, las actividades de SCM son trasparentes al ingeniero y solo se percatará de SCM cuando intente desempeñar una función u operación que no ha sido autorizada.

Incluye las siguientes caracteristicas:

(1) Cubren las características de las de en línea.
(2) Una base de datos de ingeniería integrada con comandos de construcción de SCM, los comandos de ingeniería en línea son comúnmente usados en programas de diseño y desarrollo.
(3) La integración de los comandos de SCM con comandos de administración en línea para construcción y promoción de unidades y componentes.

9.2. Herramientas de distribución libre para SCM.

Hay herramientas disponibles en internet, las cuales se obtienen sin costo alguno, a continuación se da una lista que fue tomada de "http://gille.loria.fr:7000/cgi-bin/cm/wilma/fcmt"

AEGIS - A transaction-based CM Tools
BCS - Baseline Configuration System
The CVS Bubbles - The Concurrent Version System
ICE - The Incremental Configuration Engine
PRCS - The Project Revision Control System
QVCS - Quema Version Control System
RCS - Revision Control System
SCCS - Source Code Control Syste
Shape Tools

9.3. Herramientas comerciales para SCM.

Las siguiente lista de herramienta tiene un costo y fue tomada de "http://gille.loria.fr:7000/cgi-bin/cm/wilma/ccmt", aunque pueden bajarse demos que expiran en un plazo determinado.

Aide-de-Camp/Pro (ADC/Pro) - True Software Inc.
AllChange
AllChange
Apps*Integrity - Change Management and Software Deployment
Change Man MVS SCM from Optima Software - Change Man is the leading MVS process-based SCM solution.
ClearCase
CMIS Product Services Home Page - The DoD Standard Solution for Configuration Management
CMstat Corporation - CM/PDM Tools for Commercial and Government
CMVC
CMVision - CMF (Expertware) - Configuration Management - Change Control - Process tracking
CMZ
CO-OP - Reliable Version Control System
ComponentSoftware RCS - Document Revision Control System for Windows 95/NT
Continuus/CM
Control, Control-CS, nciGENESIS
CVS - Now with commercial support available
Depot
DevMan
ENDEVOR/WSX

EPM Technology - EXPRESS Data Manager
HOPE -- Human-Oriented Programming Environment - Repository-based team programming tool
InSync CM/PDM
MKS RCS - Mortice Kern System Source Integrity
Multi-Platform Code Management Tools - Multi-Platform Development Infrastructure Tools
Object Technology International's ENVY/Developer - Collaborative component development for Smalltalk developers
ObjectCycle - Reliable Version Control for any Project
PCMS (by Doug Toppin) - Product Configuration Management System
Perforce - The Fast SCM system
Platinum CCC/Harvest
PVCS
Razor
RCE - Revision Control Engine
Sablime - by Lucent Technologies
SCLM - Library Management
SoftBench CM Product
StarBase Corporation - StarTeam - Version Control, Defect Tracking and Threaded Conversations
Sun WorkShop TeamWare - Team Source Management Tools Visually Coordinate Team Development
TeamConnection - IBM
Tesseract CM tools
VCS - Version Control System
Visual SCCS
Voodoo - Versions Of Outdated Documents Organized Orthogonally

10. ANÁLISIS DEL PROTOTIPO.

El prototipo llamado "Sistema administrador de configuraciones" (SAVE), pretende mostrar los principios de SCM que fueron recopilados en los capítulos anteriores, ya que dentro de SAVE se habla de solicitudes, versiones, elementos de software, liberaciones e informe del estado de la configuración. De hecho SAVE es una base de datos que surge de la actividad del informe del estado de la configuración de SCM. Pretende, también asistir a los desarrolladores para determinar las configuraciones de cada sistema. Además ilustra los conceptos de SCM para que adquiera cultura en esta rama de la ingeniería de software, no cubre todos los conceptos de SCM, pero son suficientes para cualquier principiante que desee aprender de SCM, al estar registrando su sistema.

El análisis y diseño del prototipo de un sistema administrador de configuraciones (SAVE) se lleva a cabo utilizando una técnica de modelado de objeto y la implementación se hace según el modelo relacional. La fase de análisis es orientado a objeto y en la fase de diseño se introduce el modelo relacional. Esto es debido a que la persistencia de los datos se logra mediante una base de datos relacional. El modelo funcional del análisis, no se elabora debido a que el prototipo es un sistema que almacenará y organizará datos y no los transformará por lo cual es muy trivial el modelo funcional [RBPE 95].

10.1. Descripción de requerimientos.

Se requiere desarrollar un sistema que registre la información de proyectos/sistemas de software con el fin de administrar configuraciones; los proyectos se dividirán en subproyectos ("n" dependiendo de la complejidad del problema que se presente), por lo cual el prototipo debe poder registrar los subproyectos asociados al proyecto; cada subproyecto tendrá asociados elementos de software, a los que se llama componentes, los cuales serán en si la materia prima del producto final. Es necesario tanto en proyectos y subproyectos saber quién está administrándolos y quien desarrolla o modifica a los componentes.

El proyecto deberá tener un nombre, acrónimo, fecha de creación, ubicación, persona que lo está administrando, orden de solicitud y las versiones de éste. Las revisiones se obtendrán de proyectos liberados. El proyecto liberado tendrá uno o más subproyectos asociados a él con sus respectivas versiones.

El subproyecto deberá tener un nombre, acrónimo, fecha de creación, ubicación, persona que lo está administrando, órdenes de solicitud y las versiones de éste. El subproyecto que se origine tendrá uno o más componentes asociados a él con sus respectivas versiones. Se permitirá combinar diferentes componentes siempre y cuando ese subproyecto con su versión respectiva no forme parte de un proyecto liberado.

Los componentes serán los elementos de software que producen los desarrolladores y sus tipos pueden ser: archivos, clases, funciones y procedimientos. La forma como se elaboran los componentes queda sujeto a los paradigmas procedural y orientado a objeto. También podrá considerarse algo híbrido de estos dos paradigmas, es decir, una clase podrá invocar a una función o procedimiento de un elemento procedural, y una función o procedimiento invocará a un método de una clase. El componente archivo puede ser utilizado en los dos paradigmas, quedando restringido a no invocar a los otros componentes. ¿Por qué el registro híbrido? Hay dos razones, la primera es porque ciertos lenguajes como "C", permiten esta forma; y la segunda debido a que hay código valioso el cual, la empresa, no se puede dar el lujo de desechar, sobre todo cuando ha pasado rigurosas pruebas en el funcionamiento, y también porque migrar ese código significa dinero a la empresa.

Los componentes sufrirán cambios al paso del tiempo, por lo cual será necesario generar nuevas versiones (revisiones) de estos. El prototipo podrá obtener información respecto al componente en cuanto a qué elementos invoca o es invocado por el componente. Estas consideraciones tienen que ser tomadas por cada versión del componente. Las versiones que se originen de cada componente deben saber que versiones las originan.

Para las clases se registrarán sus atributos y métodos, a los métodos se les controlan sus argumentos y tipo de retorno; tanto los atributos como métodos quedan controlados por las versiones del componente clase. En un registro puro orientado a objeto en las invocaciones serán a los métodos que se utilizan de otra clase o de archivos; pero si es híbrido el registro, entonces además de lo anterior se pueden almacenar las funciones y procedimientos del paradigma procedural que invoque. En los métodos se debe incluir si son privado y publico, los públicos son los que podrán ser invocados por otras clases.

Las funciones y procedimientos en sus invocaciones podrán hacer uso de archivos y de otras funciones o procedimientos; pero si es híbrido, se podrán invocar métodos de otras clases.

Los componentes deberán tener: Nombre, fecha de creación, ubicación, persona que lo desarrolla o modifica, orden de solicitud y versiones de éste. Si es clase; contendrá los atributos y métodos; si es función; tipo de retorno y argumentos; si es procedimiento; los argumentos.

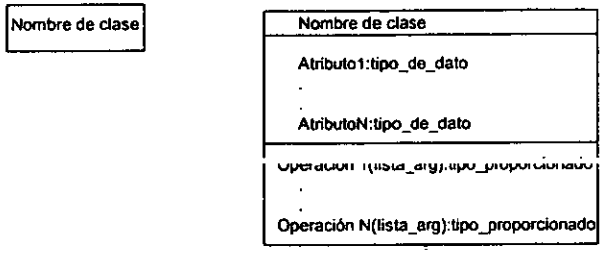
10.2. Modelo de objeto.

En esta fase de análisis tratamos de identificar los objetos del problema los cuales representan la estructura del sistema (objetos del dominio del problema). Los objetos más importantes serían: Componente, Proyecto, Subproyecto y Persona (los que desarrollan y dan mantenimiento a los componentes; los que administran

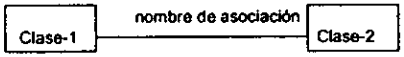
proyecto y subproyecto). La forma de como representarlo queda definido por el diagrama de las figuras 10.1 y 10.2. En estas figuras aparecen los objetos, sus relaciones, atributos y operaciones (más significativas). Este modelo es la base para todo el desarrollo del prototipo, ya que representa los objetos del problema.

10.2.1. Notación del modelo de objeto.

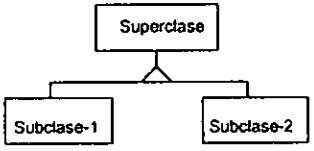
Clase:



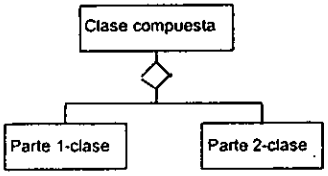
Asociación:



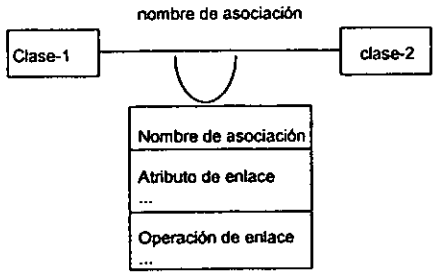
Generalización (herencia):



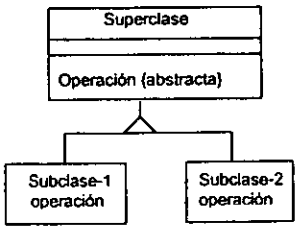
Agregación:



Asociación como clase:



Operacion abstracta:



Uso:

10.3. Supuestos.

Los supuestos se toman como las reglas del juego en el sistema y serán las siguientes:

a) Un proyecto en sus diferentes versiones es administrado por una persona o usuario.

b) Un subproyecto en sus diferentes versiones es administrado por una persona y queda sujeto con todas sus versiones a un solo proyecto; es decir se crea el proyecto y de ahí se proceden a crear los subproyectos.

c) Una persona puede administrar uno o más proyectos o subproyectos.

d) Un proyecto que se quiera liberar no debe ser igual en sus ensambles de subproyectos con sus respectivas versiones a uno que ya esté liberado.

e) Un subproyecto en su ensamble de componentes tiene uno o más componentes, no se contempla el alcance de validar que los componentes de un subproyecto en su versión sean diferentes a las otras versiones.

f) Tanto los proyectos, subproyectos y componentes pueden tener "n" versiones, las cuales sabrán de que versión se originaron.

g) Una persona puede desarrollar o modificar "n" componentes, también diferentes versiones del componente tendrán diferentes personas que las modifiquen.

h) Un componente puede invocar a cero o más componentes.

10.4. Modelo dinámico.

Una vez que se identifican los objetos que compondrán la estructura del sistema, se procede a establecer el modo en como se llevará la comunicación entre los objetos. Esto se logra planteando las funcionalidades del sistema por medio de escenarios. Se establece el modelo dinámico del prototipo en base a diagramas de seguimiento de trazo de sucesos. En las figuras 10.3, 10.4, 10.5, 10.6, 10.7, 10.8 y 10.9 Se presentan algunos escenarios que contendrá el sistema y son: alta de proyecto (10.3), subproyecto (10.4), clase (10.5) y función (10.6); captura de invocaciones en la clase (10.7); ensamble de componentes en un subproyecto 10.8) y generación de una revisión de una función (10.9).
Los diagramas de estado no se realizan debido a que el sistema será interactivo con el usuario.

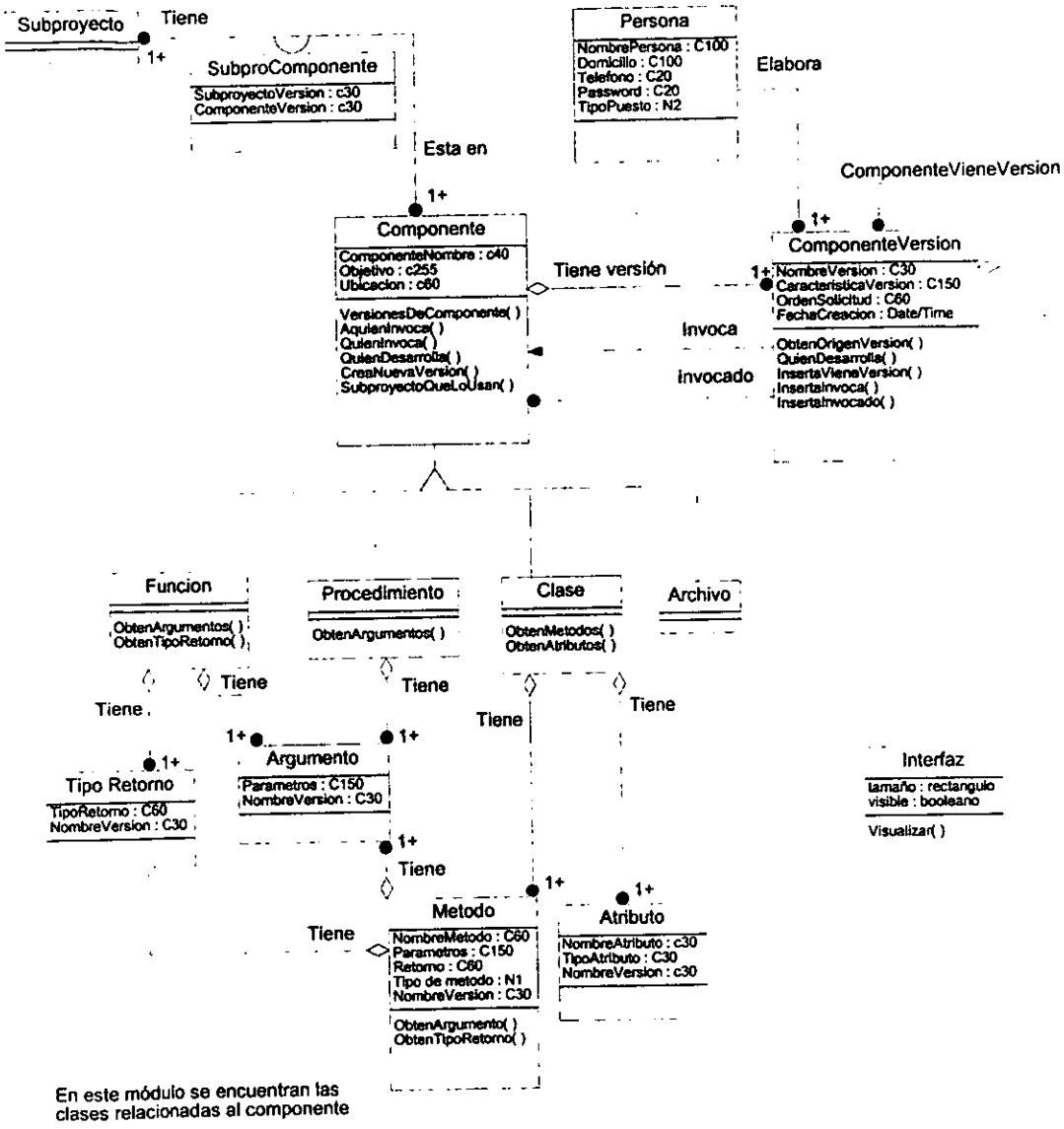


Figura 10.1 Módulo de componente.

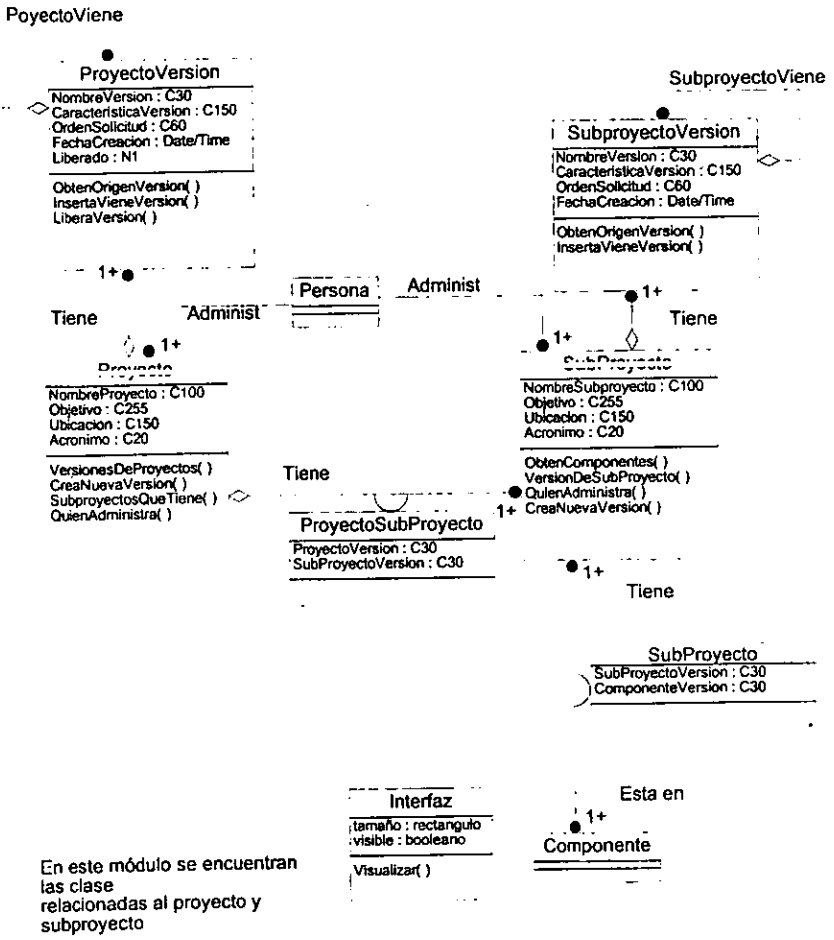


Figura 10.2 Módulo de Proyectos y SubProyectos

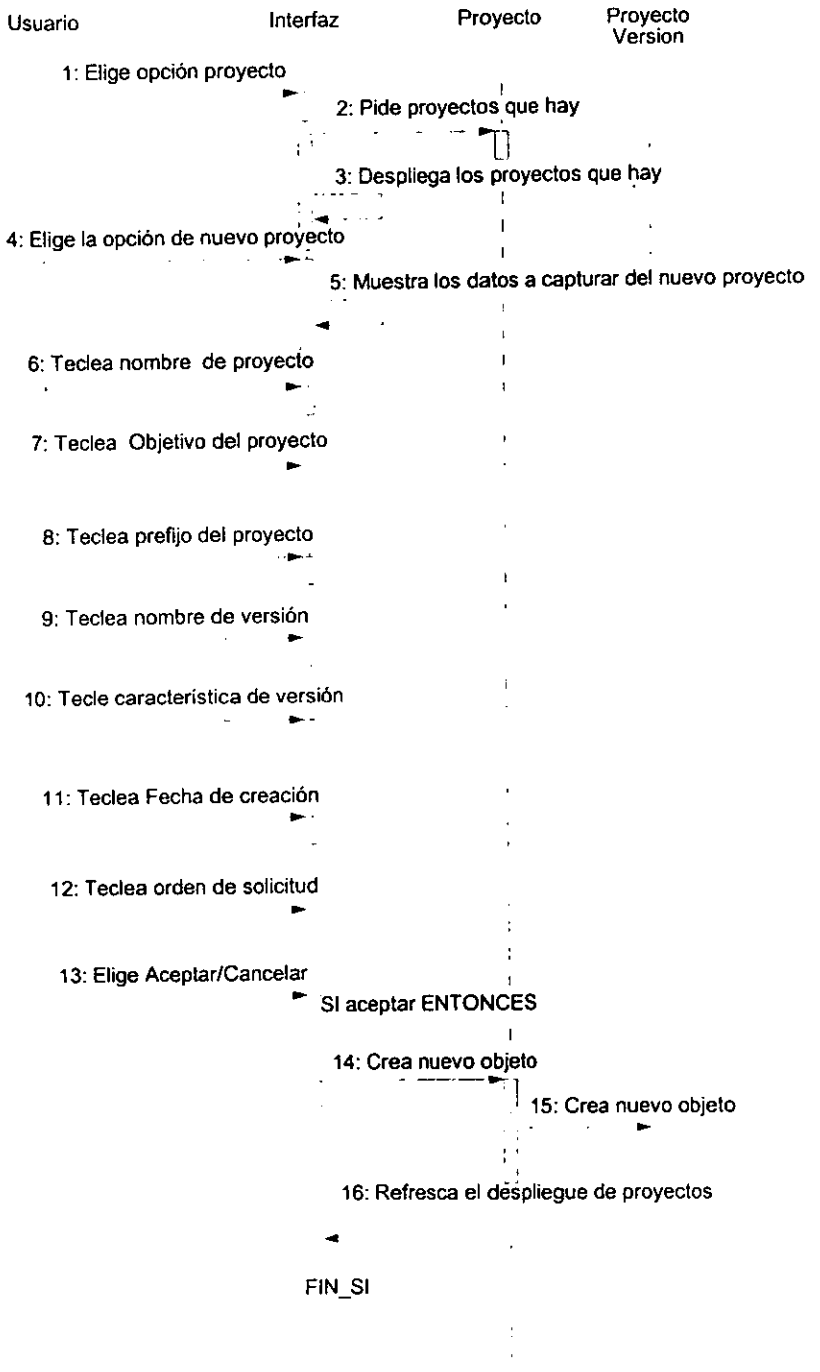


Figura 10.3 diagrama de trazo de sucesos de una alta

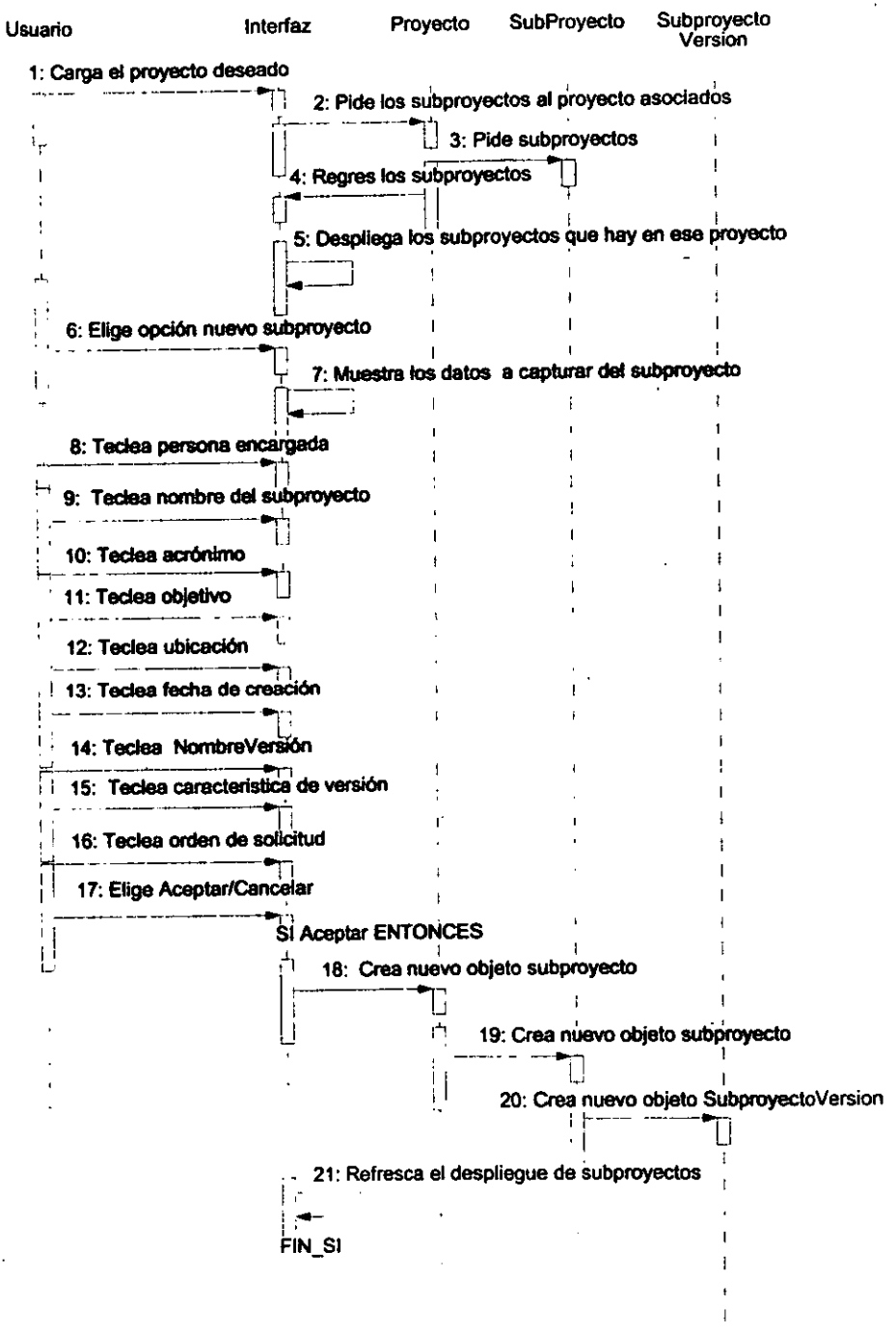


Figura 10.4 diagrama de trazo de sucesos para una alta de un subproyecto

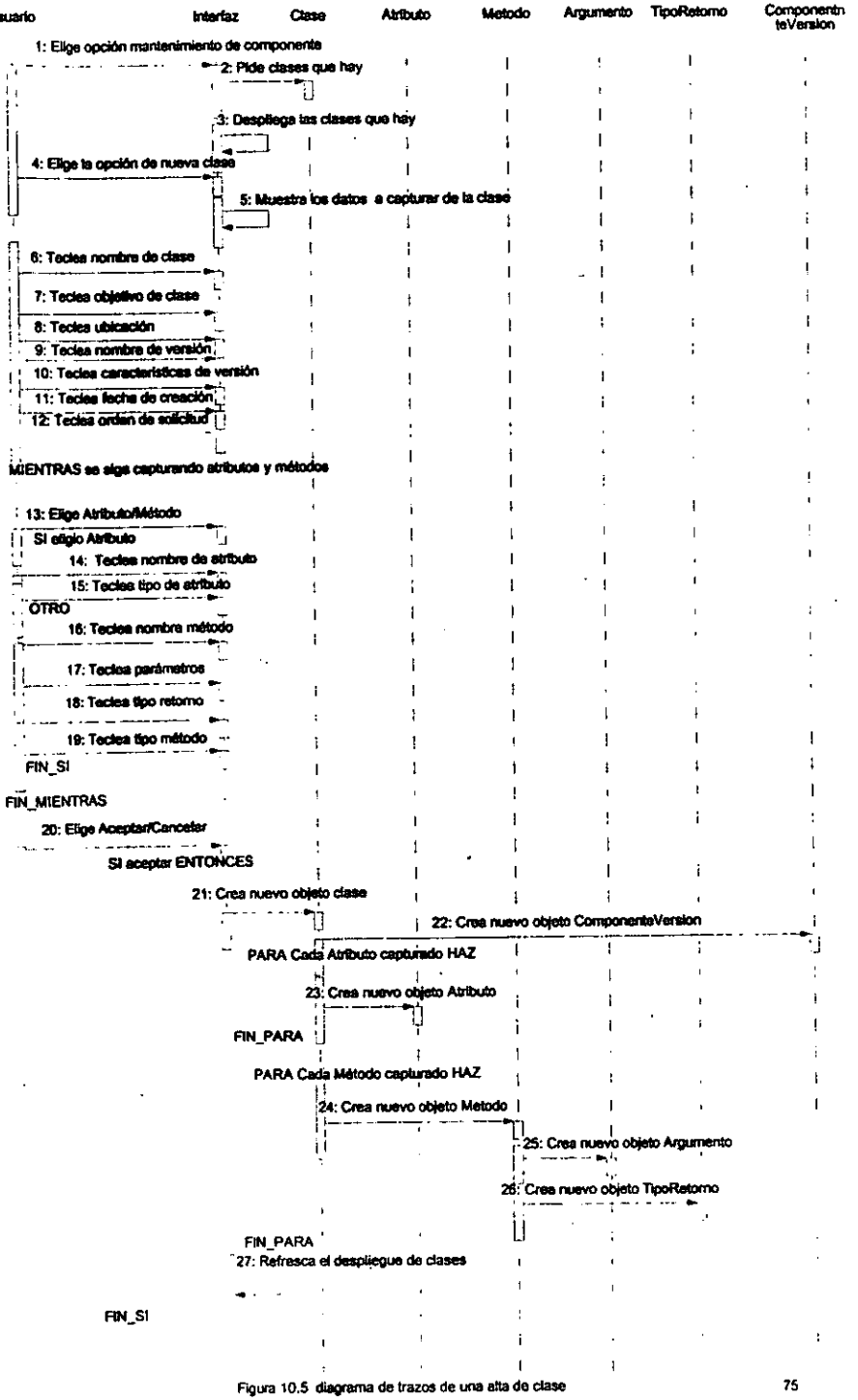


Figura 10.5 diagrama de trazos de una alta de clase

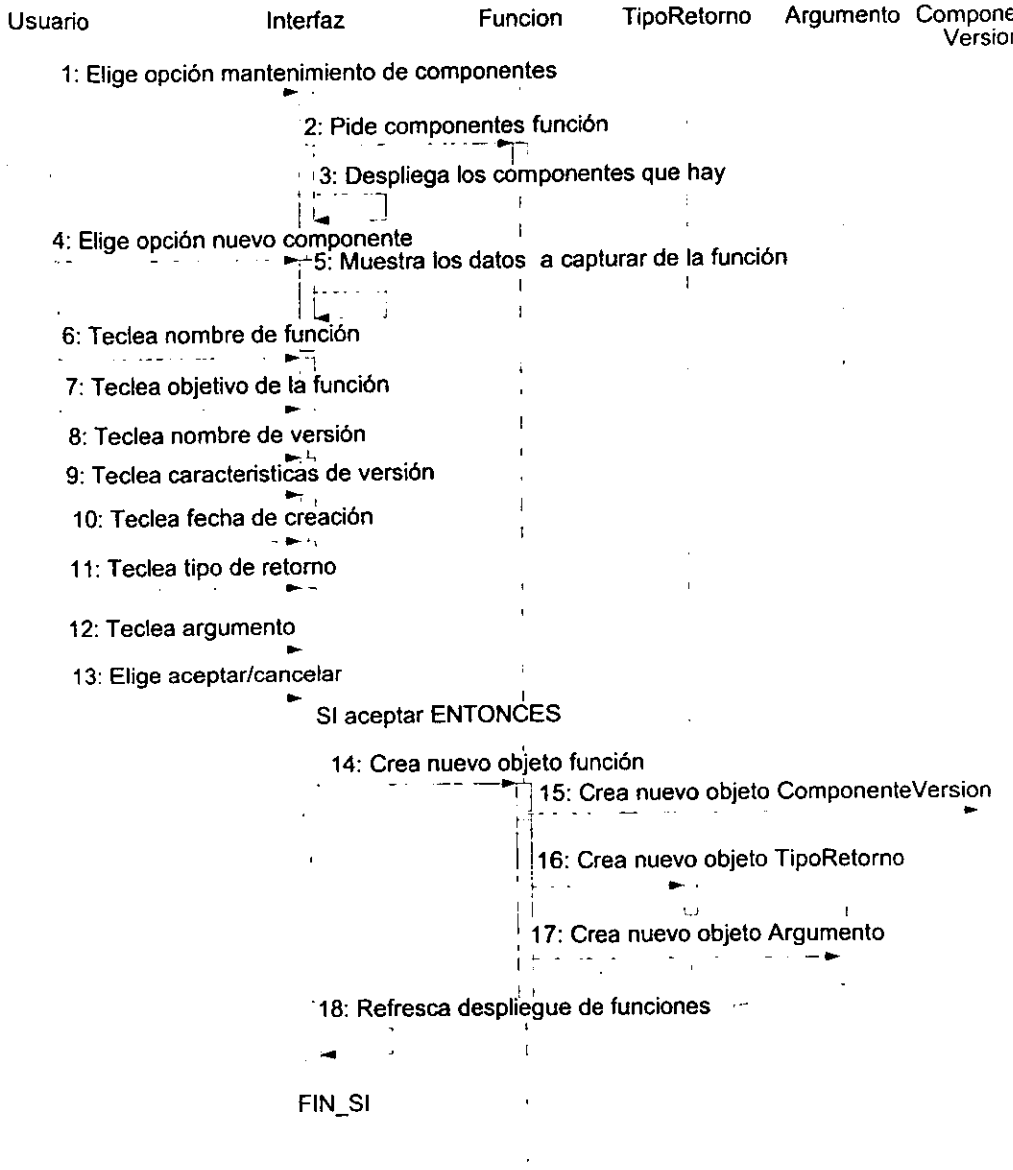


Figura 10.6 diagrama de trazo de un alta de componente función

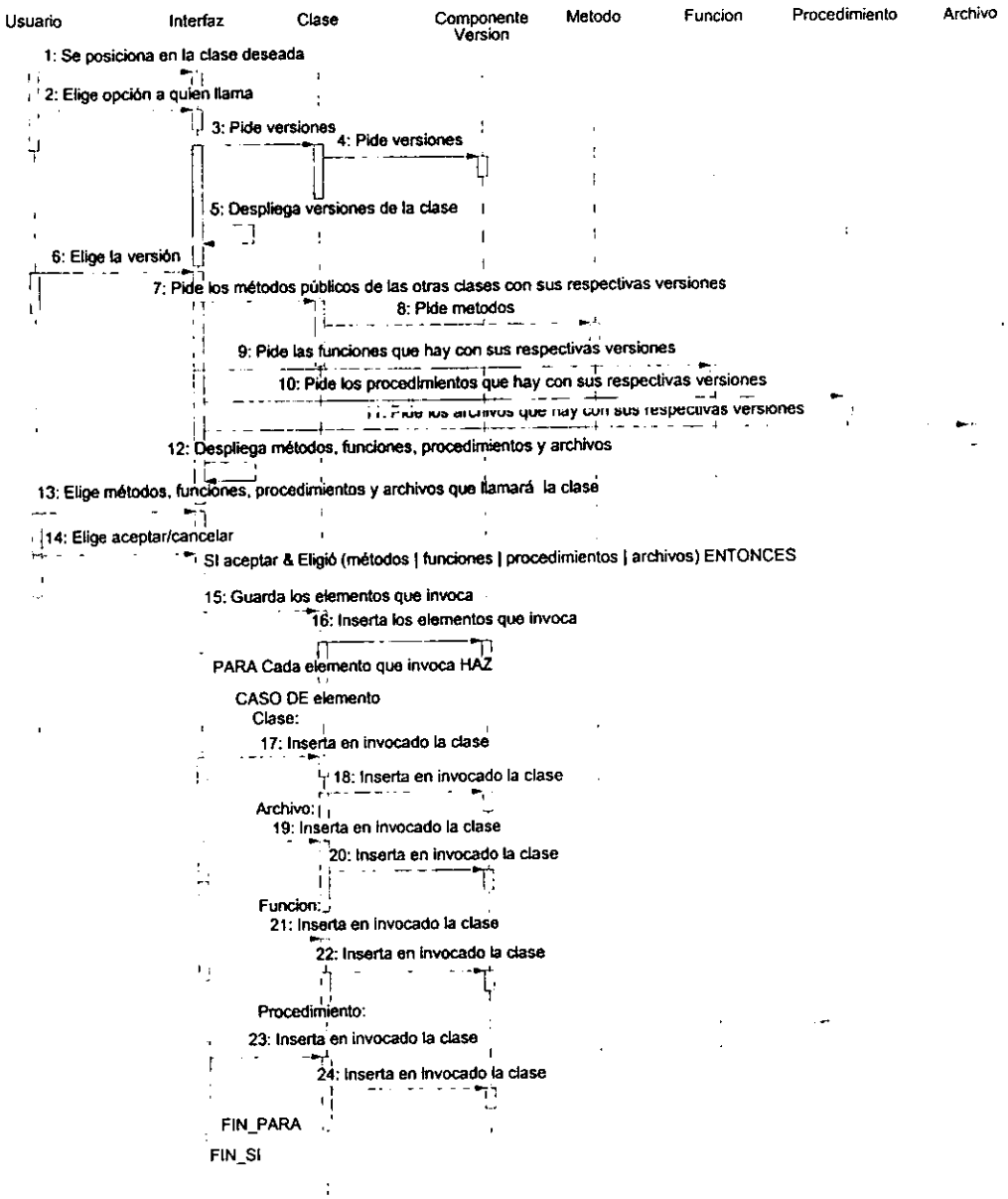


Figura 10.7 diagrama de trazo de sucesos para a quién invoca un componente clase

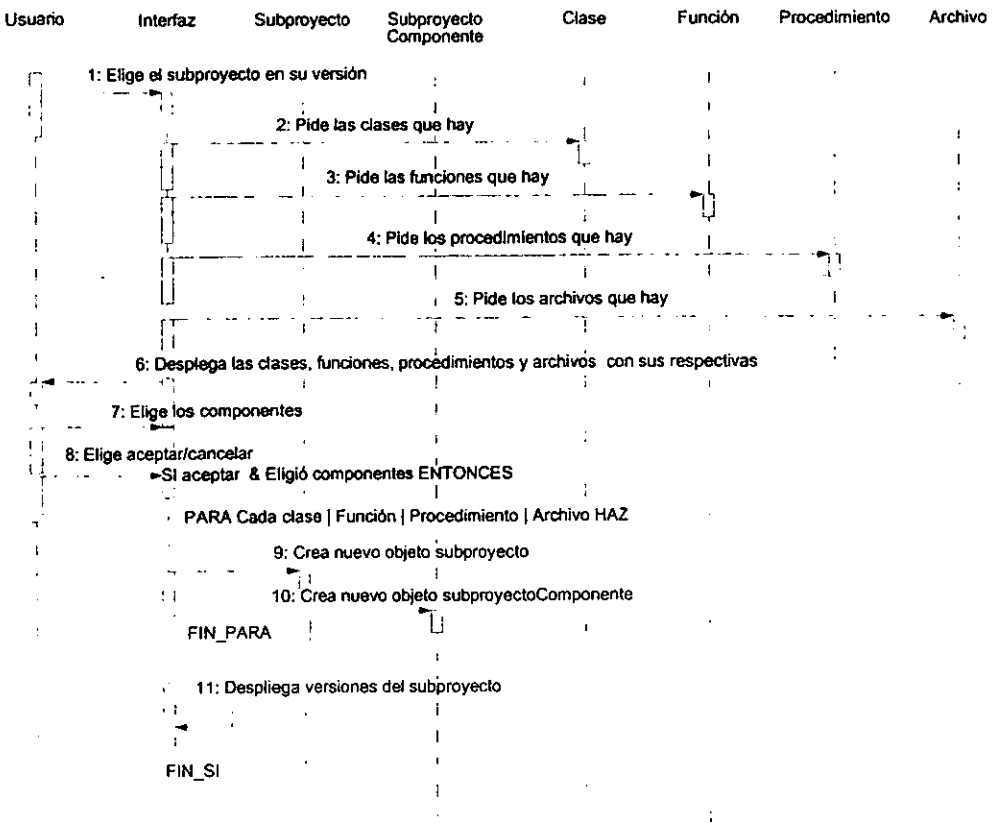


Figura 10.8 diagrama de trazo de sucesos de ensamble de componentes a un subproyecto

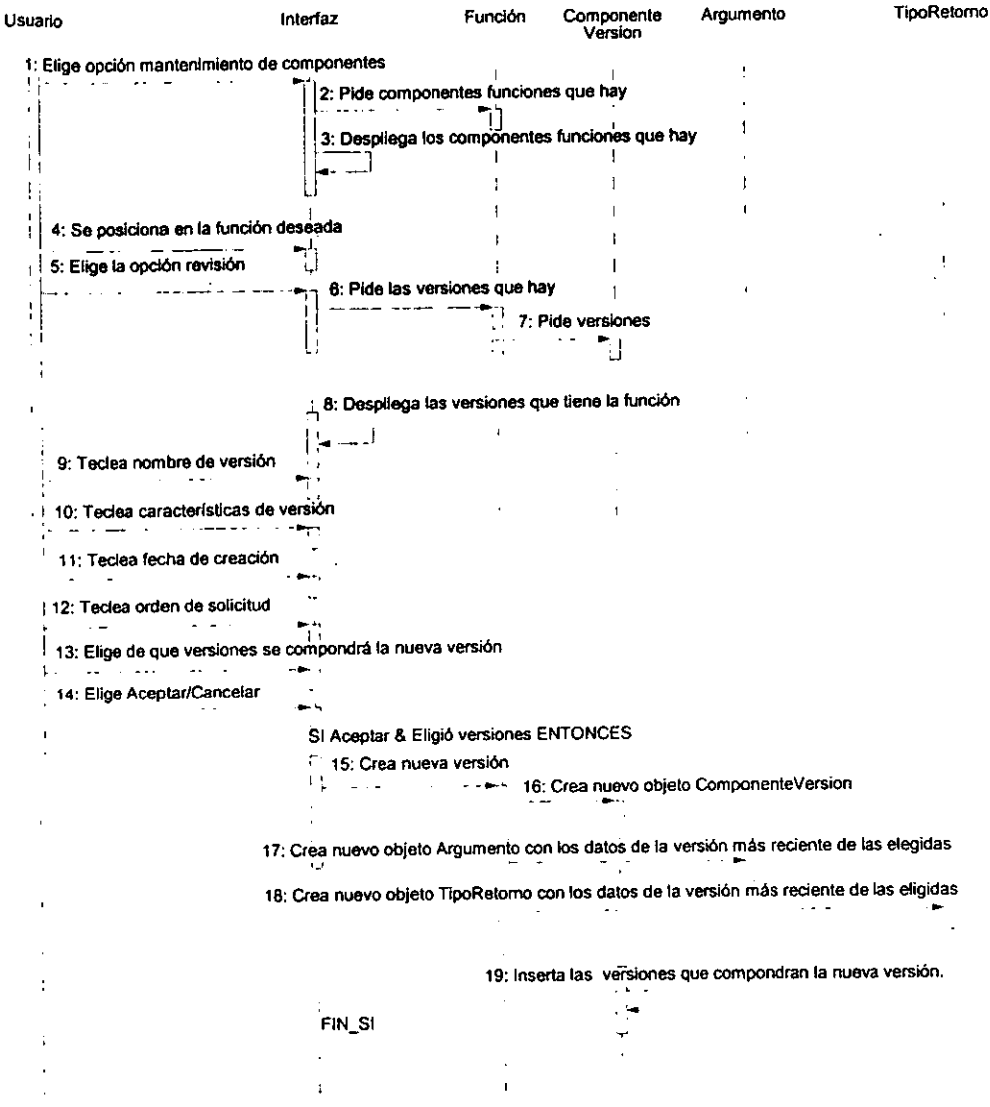


Figura 10.9 diagrama de trazo de sucesos para una generación de una revisión de un componente función

ESTA TESIS NO DEBE
SALIR DE LA BIBLIOTECA

11. DISEÑO DEL PROTOTIPO.

El diseño del sistema es la estrategia de alto nivel para resolver el problema y construir una solución [RBPE 95]. En esta fase se incluye parte del diseño de objeto (decisiones de implementación). Con las que se obtienen hasta las pantallas que mostrará el prototipo, así como el modelo de datos correspondiente, donde quedarán los datos almacenados, por lo cual se puede codificar el sistema.

11.1. Estrategia básica para implementar persistencia a los datos.

El modelo de objetos (estructura estática del sistema) se mapea a una base de datos relacional utilizando el modelo entidad relación.

En modelo de objeto o diagrama de clases se tienen: las clases, asociaciones y herencias; mientras que en el modelo entidad relación se tienen entidades, relaciones, y subtipos de entidades; las clases se convierten en entidades; las asociaciones, agregaciones y relación de uso se convierten en relaciones y la herencia que se presenta de una clase a subclase se convierten en subtipos de entidades. Lo que se mantiene de una clase son los atributos, porque los métodos carecen de sentido en el modelo entidad relación. En el modelo entidad relación la entidad es un depósito de registros, los cuales son distinguidos de manera única por un atributo que se denomina llave primaria, este atributo no es definido por el modelo de objeto, sin embargo es básico dentro del modelo entidad relación debido a que en el modelo relacional además de poder identificar de manera única a las entidades, hay que identificar de forma única cada una de las tuplas que se almacenan. Partiendo de esta base se procede a llevar a cabo el proceso de traslado del modelo de objeto a el modelo entidad relación de SAVE.

1. Los objetos versiones tanto de **ComponenteVersion**, **SubproyectoVersion** **ProyectoVersion.** La llave primaria de estas entidades serán: el identificador de componente/proyecto/subproyecto y el identificador de versión. Las entidades resultantes serían:

Componente_Version(Identificador de componente, identificador de versión, nombre de versión, característica, orden de solicitud, fecha de creación).

Subproyecto_Version(Identificador de subproyecto, identificador de versión, nombre de versión, característica, orden de solicitud, fecha de creación).

Proyecto_Version(Identificador de proyecto, identificador de versión, nombre de versión, característica, orden de solicitud, fecha de creación)

2. En las clases **ComponenteVersion**, **SubproyectoVersion** y **ProyectoVersion**, la agregación de las VersionOrigen, pasarían a ser entidades nuevas que se

llamarán: **Componente_Viene_Version**, **Subproyecto_Viene_Version** y **Proyecto_Viene_Version**. Los atributos que tendrán serían:

Componente_Viene_Version(Identificador de componente, Identificador de versión, Identificador de viene versión).

Subproyecto_Viene_Version(Identificador de subproyecto, Identificador de versión, Identificador de viene versión).

Proyecto_Viene_Version(Identificador de proyecto, Identificador de versión, Identificador de viene versión).

3. Entre **Componente** y **ComponenteVersion** hay una relación de uso que se llama **Invoca**. Esta relación de uso menciona que componente en su versión utiliza los servicios de otro(s) componente(s). La asociación de **Invocado** representa a quienes invocan a ese componente. Estos dos tipos de relaciones se representan con dos entidades que se llamarán **Invoca_ArcFunPro** y **Invoca_Metodo**. Con **Invoca_ArcFunPro** se representan a los componentes archivos, funciones y procedimientos. **Invoca_Metodo** representará las llamadas a métodos de otra clase que utiliza una clase. Los atributos de esta quedan de la siguiente manera:

Invoca_ArcFunPro(identificador de componente padre, identificador de componente padre versión, identificador de componente hijo, identificador de componente hijo versión)

Invoca_Metodo(identificador de componente padre, identificador de componente padre versión, identificador de componente hijo, identificador de componente hijo versión, identificador de método)

4. A la clase **Persona** se agrega un identificador para formar la entidad correspondiente y el atributo TipoPuesto pasaría a ser una llave primaria en otra entidad que se llamará TipoPuesto. La entidad queda **Persona**(identificador de persona, nombre de persona, domicilio, teléfono, password, TipoPuesto)

5. En la clase **Proyecto** se agrega el identificador de proyecto y el de persona como llave foránea para obtener la entidad. Con la clase **Subproyecto** se hace lo mismo, sólo que además se debe agregar el identificador de proyecto como llave foránea, de esta manera obtenemos la entidad requerida. Estas entidades quedan de la siguiente manera:

Proyecto(identificador de proyecto, identificador de persona, nombre de proyecto, objetivo, prefijo, ubicación).
Subproyecto(identificador de subproyecto, identificador de proyecto, identificador de persona, nombre de subproyecto, objetivo, prefijo, ubicación).

6. En **componente**, para representar la herencia tanto de **archivo**, **clase**, **función** y **procedimiento**, se agrega un tipo componente para que no haga la distinción de que tipo de componente estamos hablando. Se crean subtipos de entidades de archivo, función y procedimiento; la clase **clase** se representa con relaciones hacia la entidades atributo y método.

La subcategoría (subentidad) archivo tiene los atributos identificador de componente y versión y tipo de archivo.

La subcategoría función sus atributos son: identificador de componente y versión, tipo de retorno y argumentos.

La subcategoría procedimiento sus atributos son: identificador de componente y versión, y argumentos.

La entidad Atributo tiene los atributos: identificador de componente y versión, nombre y tipo de atributo.

La entidad Método tiene los atributos: identificador de componente y versión, tipo método y tipo retorno.

Las entidades que se originaron de la herencia son las siguientes:

Archivo(Identificador de componente, identificador de componente versión, tipo de archivo)

Funcion(identificador de componente, identificador de componente versión, tipo de retorno, argumentos).

Procedimiento(identificador de componente, identificador de componente versión, argumentos)

Atributo(identificador de componente, identificador de componente versión, identificador de atributo, nombre de atributo, tipo de atributo)

Metodo(identificador de componente, identificador de componente versión, identificador de método, nombre de método, tipo de método, argumentos, tipo de retorno)

7. La clase **componente** para pasarla a una entidad se le agrega su identificador, la llave foránea de tipo de componente, el resultado de esta entidad sería:

Componente(identificador, tipo de componente, nombre de componente, objetivo, ubicación).

8. Las clases **ProyectoSubproyecto** y **SubproComponente** se dejan tal y como están sólo hay que agregar dos identificadores a las dos entidades en el modelo entidad relación y son: identificador de proyecto y subproyecto en ProyectoSubproyecto; identificador de subproyecto y componente en SubproComponente. Y los nombres en el modelo entidad relación quedan:

Proyecto_Subproyecto(identificador de proyecto, identificador de subproyecto, identificador de proyecto versión, identificador de subproyecto versión)

Subproyecto_Componente(identificador de subproyecto, identificador de componente, identificador subproyecto versión, identificador de componente versión)

11.2. Modelo de datos.

Al tomar la decisión de que por medio de una base de datos se aplicaría la persistencia a los datos, el diagrama final del modelo entidad relación para el Sistema Administrador de Configuraciones (SAVE) se muestra en la figura 11.1.

11.3. Dividir en subsistemas.

El prototipo resultante se divide para el manejo de la complejidad en cuatro subproyectos que fueron los siguientes:

Sistema Administrador de Configuraciones

(1) Proyectos
(2) Subproyectos
(3) Componentes
(4) Catálogos.

El subsistema de **proyectos** maneja las altas, modificaciones, bajas y consultas del proyecto; cargar un proyecto para conformar su configuración o consulta de ésta.

El subsistema **subproyectos** controla las altas, modificaciones, bajas, consultas y revisiones; así como ensamblar los componentes.

El subsistema **componentes** controlará las altas, modificaciones, bajas, consultas y revisiones de componentes; así como las consultas de qué proyectos utilizan a cierto componente; poder llenar también la lista de invocaciones (A quien llama el componente).

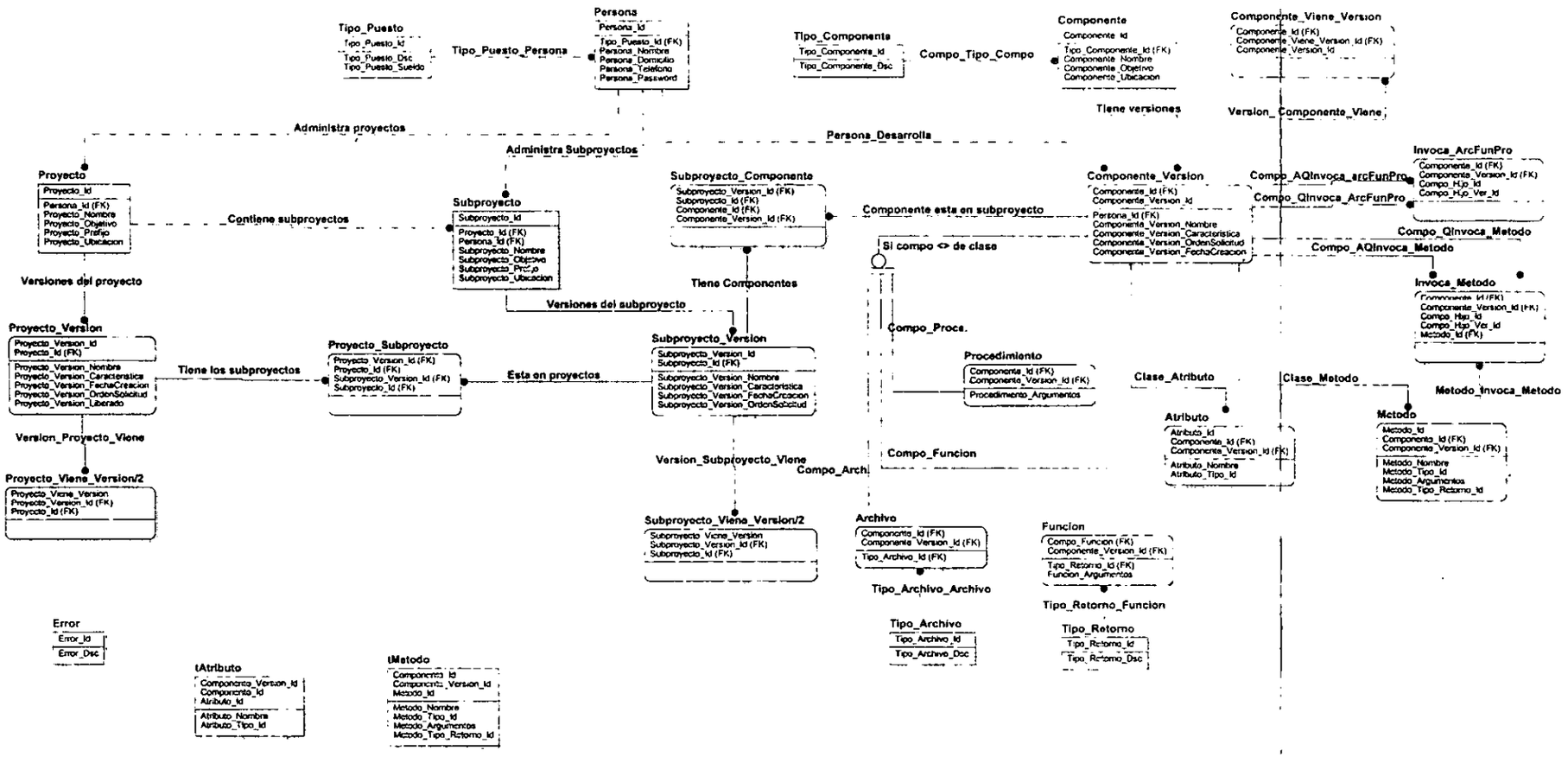


Figura 11.1 Modelo Entidad Relación para el Sistema Administrador de Configuraciones

El subsistema **catálogos** controla las altas, modificaciones, bajas y consultas de persona, tipo de retorno, tipo de archivo y tipo de puesto.

11.4. Recursos.

El prototipo se desarrollará en ambiente Windows 95, utilizando el lenguaje orientado a objeto Delphi 3.0; un manejador de base de datos Microsoft Acces 2.0 (ODBC de 32 bits); ERWin/ERX 2.5 para la generación del scrip de la base de datos; una computadora personal (PC) pentium con espacio de 20 Mb. (Megabytes) en disco duro y RAM de 16 megabytes (Mb); el prototipo quedará sujeto a una PC corriendo solo a nivel monousuario.

Para la operación del prototipo se requerirá una PC 486 DX o mayor con espacio en disco duro de 50 megabytes, memoria RAM mínimo 16 megabytes, la velocidad de la PC que sea mínimo 60 Mhz.

11.5. Manejo de condiciones de entorno.

Al inicio el sistema (cuando se opera por el usuario) pedirá un clave de entrada la cual se cotejará con su nombre para determinar si tiene acceso de entrada; al desplegarse el sistema no tendrá proyectos, ya que el usuario tendrá que cargar uno para que de esta manera se configure o consulte.

Las ventanas que maneje se tendrán que crear y liberar para que no consuma demasiada memoria (operando el sistema se podrá notar si conviene crearla cada vez que se muestran las ventanas o crear todas cuando arranca éste, en principio se crean las ventanas y se destruyen cada vez que se invocan.)

Para la terminación del sistema debe poder dejar la base en un estado consistente y liberar la memoria que esté ocupando.

El prototipo controlará los errores que surjan cuando esté corriendo. Buscará la forma de recuperarse de los errores para que el usuario pueda salvar lo que está haciendo o con un mensaje avise porque la operación es válida. Hay una entidad Error, la cual se puede ir retroalimentando con los errores que surjan cuando esté operando el sistema.

11.6 Pantallas.

Las pantallas serán la interfaz que presente, al usuario, el prototipo; se escogen seis pantallas para ilustrar algunas funciones de SAVE, si se desea saber más acerca de las pantallas del prototipo ver el sistema administrador de configuraciones. Las figuras 11.2, 11.3, 11.4, 11.5, 11.6 y 11.7 representan las pantallas que aparecen en los cuatros subsistemas en que se dividió SAVE.

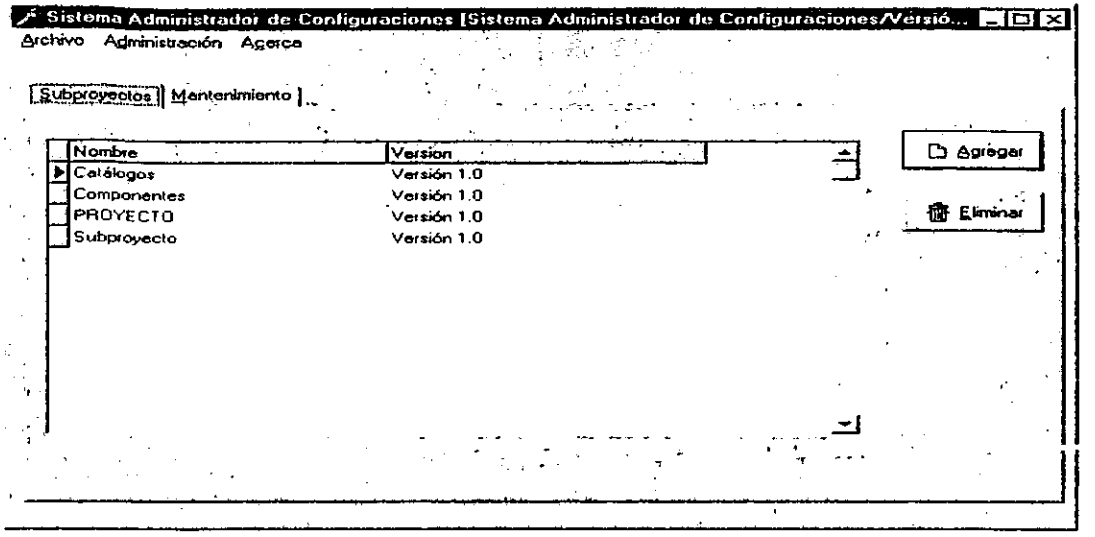


Figura 11.2

Esta pantalla representa el menú principal de SAVE con un proyecto cargado, desplegando los subproyectos que tiene éste. Hay dos botones que son: Agregar y Eliminar; con agregar se asigna un subproyecto al proyecto de los que están en mantenimiento, eliminar quita uno de los proyectos subproyectos que están desplegados en la pantalla (el seleccionado con la barra del cursor). La pestaña que sobresale de mantenimiento, al activarla con el ratón se pasa a dar mantenimiento a subproyectos (alta, baja, consulta y modifica).

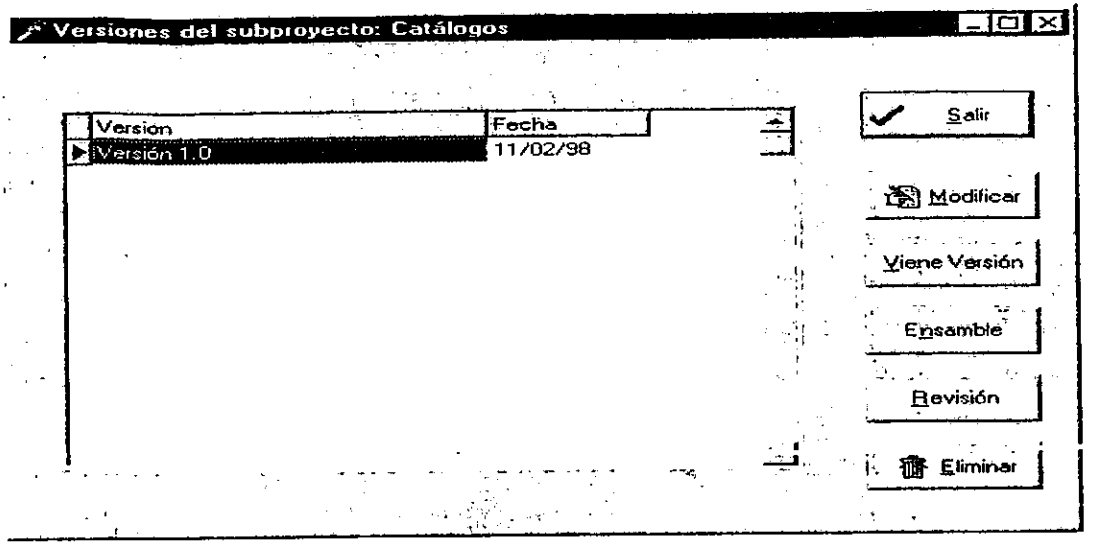


Figura 11.3

Esta pantalla despliega las versiones que posee un subproyecto elegido. El botón modificar permite cambiar los datos de la versión activa. El botón Viene Versión desplegará las versiones que dieron origen a la que se está activa en la rejilla. El botón Ensamble permite ensamblar los componentes que llevará ese subproyecto. El botón Revisión genera una nueva versión del subproyecto. El botón Eliminar borra la versión resaltada con la barra de selección.

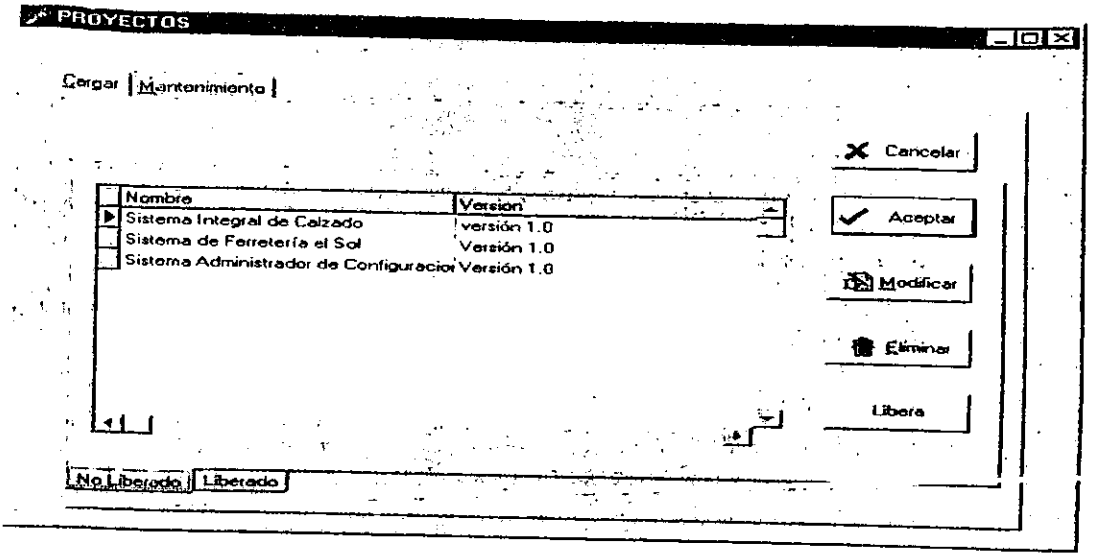


Figura 11.4

Es es la pantalla que muestra los proyectos que se deben cargar para armar su configuración o consultarla. Al entrar al subsistema de proyectos esta pantalla mostrará los proyectos no liberados. Con la pestaña de liberado se muestran y se cargan los proyectos liberados. En la parte superior izquierda existe una pestaña para ejecutar la opción de mantenimiento con la que se pueden introducir nuevos proyectos y eliminar o modificar los ya existentes. El botón cancelar; abandona esta pantalla sin cargar ninguno. El botón aceptar; carga el proyecto resaltado con la barra de selección. El botón modifica; permite cambiar datos al proyecto activo en la rejilla. El botón eliminar; borra un proyecto no liberado. El botón libera; traslada el proyecto activo al conjunto de los liberados, no sin antes hacer una previa validación.

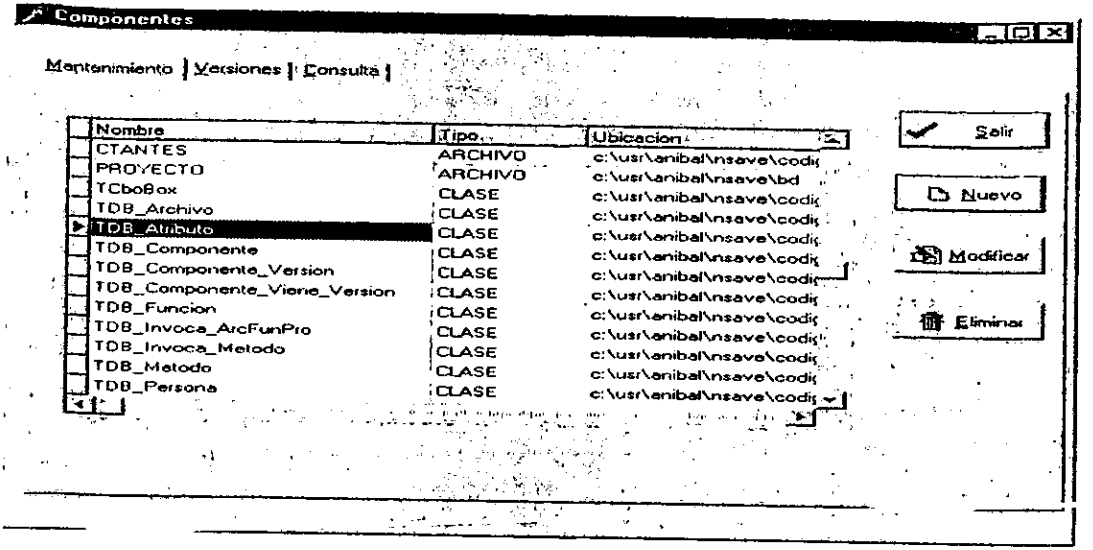


Figura 11.5

Esta pantalla es la de mantenimiento de componentes. Aquí se pueden hacer las operaciones sobre el componente que son: alta, modificar y eliminar.

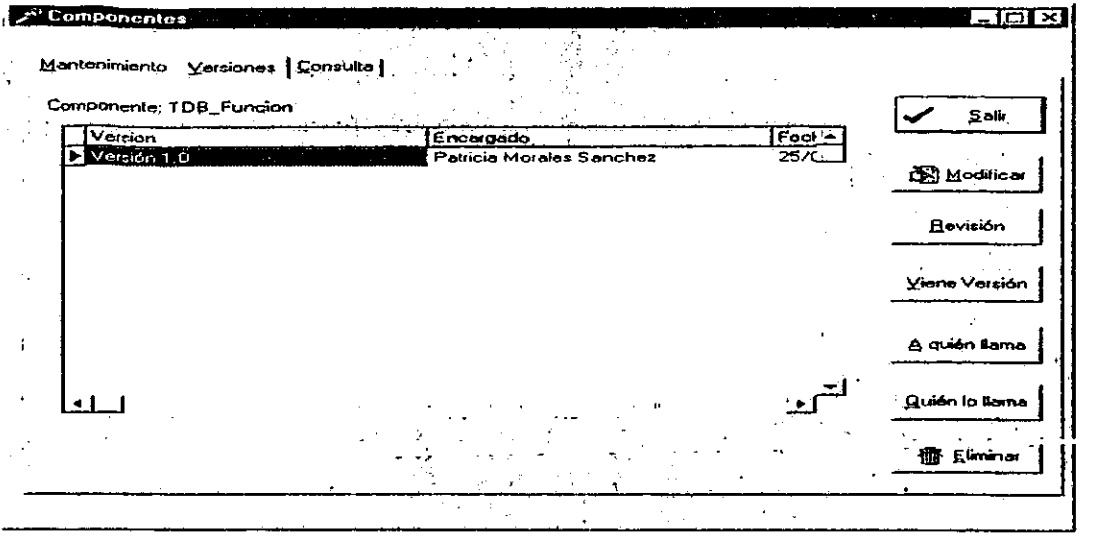


Figura 11.6

Esta ventana muestra las versiones que tiene un componente elegido. El botón salir; permite abandonar esta ventana. El botón modificar: cambia los datos de la versión activa en la rejilla. El botón Revisión: permite generar una nueva versión del componente elegido. El botón Viene Versión: visualiza las versiones que dieron origen a la que está activa en la rejilla. El botón A quién llama: permite asociar (insertar) insertar los componentes a los cuales invoca éste. El botón Quién lo llama: despliega los componentes los cuales invocan a éste. El botón Eliminar: borra la versión resaltada con la barra de selección.

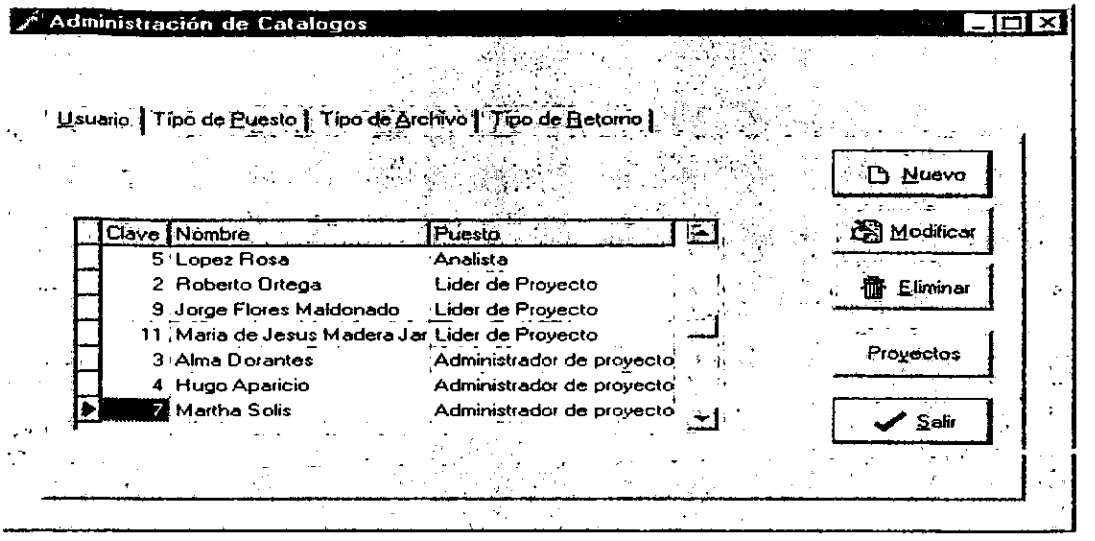


Figura 11.7

Esta es la pantalla de administración de catálogos que utiliza SAVE. Los catálogos que se tienen son: Usuario, Tipo de puesto, Tipo de archivo y Tipo de retorno. Las operaciones que son llevadas sobre éstos son: altas, bajas, cambios y consulta. El botón proyecto despliega los proyectos a los que está administrando esa persona. El botón salir abandona esta pantalla.

11.7. Arquitectura.

El sistema SAVE es un sistema interactivo. En la estación PC que se instale deberá interactuar con seres humanos, los cuales le proporcionarán la información. La cual se almacenará en una base de datos, que también podrá permitir actualizarla, borrarla, consultarla, y explotarla por otros sistemas.

La figura 11.8 muestra la arquitectura del prototipo.

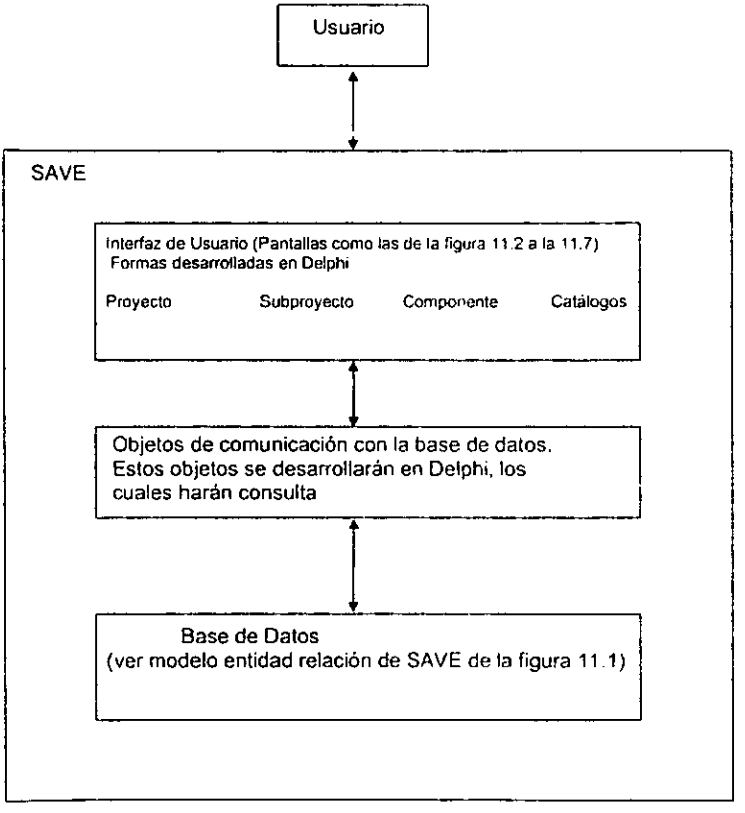


Figura 11.8

11.8. Implementación.

La forma como se implementará el prototipo, utilizando Delphi y la base de datos, será la siguiente:

De la arquitectura de la figura 11.8 se observan tres capas (interfaz, objetos de comunicación y base de datos) cada una de ellas efectúa una función para que el sistema opere.

La primera capa es la interfaz con el usuario, por lo cual se tendrán que elaborar primero las pantallas. Estas ventanas se crean en Delphi. La forma creada (ventana) llama a otras ventanas o invoca a un objeto de comunicación de la segunda capa, con el fin de que afecte o consulte a la base de datos.

La segunda capa de comunicación constituida por objetos que representan a las entidades del modelo entidad relación (ver figura 11.1). Cada clase tendrá los mismos atributos del modelo entidad relación, sus métodos que se diseñan para consultar o actualizar a la base de datos, se deben llevar a cabo a través de objetos consulta (Query) que manden como parámetros por referencia, de esta manera la consulta o modificación de la información se hará a través de la consulta (Query). Por lo cual, para las ventanas, la comunicación con la base de datos será trasparente.

La tercera capa, que es la base de datos, responderá a las solicitudes que se le hagan a través de la consulta (Query). Estará pasiva y sólo emitirá mensajes de error al manejador cuando se quiera violar la integridad de la base de datos. Con los errores que surjan se puede alimentar a la entidad Error, dando el número de error y su descripción traducida al español.

CONCLUSIONES.

El presente trabajo se enfocó en dos vertientes y son: una recopilación teórica de SCM (principios) y la construcción de un prototipo de un administrador de configuraciones (SAVE).

La recopilación teórica de SCM podría utilizarse para:
(1) Los lectores que deseen conocer que es SCM y como ayuda en la producción o mantenimiento de un proyecto de software podrán recurrir a este trabajo para conocer sus actividades y los conocimientos básicos de SCM, los cuales les servirán para profundizar más en este campo.
(2) A los que conozcan les servirá para refrescar sus conocimientos.
(3) El material puede servir de apoyo en cursos de ingeniería de software
(4) Como difusión de SCM en cursos externos a la comunidad universitaria
(5) Poder obtener conocimiento para elaborar una herramienta que ayude a desempeñar las actividades de SCM, como el caso de SAVE.
Esta recopilación cumple con el objetivo de la tesis, ya que engloba todo lo relacionado con SCM.

Al construir el prototipo se logró lo siguiente:
(1) Modelar los principios de la recopilación teórica para representar una de las características que posee una herramienta básica de ayuda en las actividades de SCM [IEEE-1042 87].
(2) Instruir a un principiante, ya que al usar a SAVE, él se familiarizará con los términos de SCM; al realizar el trabajo de registro comprenderá los conceptos de versión, cambios, revisión, liberado y orden de solicitud.
(3) Se consigue tener un registro preciso de la configuración de un sistema, de tal manera que se puedan explotar los datos para obtener el documento de descripción de versión, el cuál contendrá la versión del sistema y los elementos que lo constituyen con sus respectivas versiones. Habría que llevar a cabo el diseño de estos documentos para que sean emitidos como un reporte, en el alcance de SAVE no fueron considerados, pero la información está presente.
(4) Al llevar a cabo un registro cuidadoso de un sistema utilizando SAVE, apoya a los programadores/desarrolladores en cuanto al impacto del cambio, ya que el prototipo determina los componentes que llaman al componente en el que se desea realizar un cambio, solo hay que registrar estas llamadas.
(5) La evolución de los proyectos, subproyectos y componentes, se lograron de manera satisfactoria con la representación de sus versiones, ya que hasta se puede determinar que versiones dieron origen a una versión dada.
(6) SAVE también muestra como las herramientas que existen en el mercado, logran la implementación de un sistema administrador de configuraciones en una base de datos.
(7) Se utilizó la técnica de modelado de objetos para realizar el análisis y diseño. En la parte del diseño correspondiente a la persistencia de los datos se uso el modelo relacional.

(8) La implementación se llevó a cabo utilizando Delphi y el manejador de base de datos Acces. Con estas dos herramientas se implantó la administración de los datos del modelo relacional.

El objetivo que no se logró cubrir es el manejo de las condiciones de entorno de errores con SAVE. En sección 11.5 se mencionó que SAVE manejaría una entidad Error. Está entidad no se llenó con los errores que surgian cuando el sistema fue puesto en marcha. Uno de los problemas por el cual no se alcanzó a cubrir fue porque los errores que arroja el lenguaje de programación con la base de datos, no corresponden con el manejador de base de datos; los identificadores son diferentes. Para solucionarlo hay que consultar los manuales del motor del lenguaje de programación y el manejador de base de datos.

Recomendaciones para extender SAVE:

(1) Agregar los clientes. Una empresa de desarrollo de software podrá saber que versiones tienen cada cliente de su producto y de esta manera saber mandarle una actualización correspondiente que le repare errores. Servirá de mucho esta parte porque habrá errores que manden los clientes y que ya fueron reparados en nuevas versiones.
(2) Manejar una orden de solicitud. Si se tiene esta entidad con un atributo identificador que sea el número de solicitud, se sabrá cual es el estado de esta solicitud (pendiente, aprobada o rechazada).
(3) Introducir un atributo de estado a la entidad componente. Este atributo determinará la situación o línea de base en que se encuentra el componente (diseño, implementándose, pruebas o liberado).
(4) Trabajar en el problema de particionar los subproyectos. En la práctica un subproyecto por su complejidad se puede subdividir y estas subdivisiones también pueden descomponerse para facilitar el manejo de la complejidad. Por el momento SAVE, soluciona el problema, aunque no de la manera adecuada; al nombrar el subproyecto y éste se deriva de otro subproyecto, en su nombre se le debe agregar un prefijo, el cual indique de que subproyecto viene o añadir el nombre.
(5) Mejorar el desempeño de SAVE, esto se detectará mediante su uso continuo.

Concluyendo podemos decir que el trabajo suministra apoyo para administrar la configuración de un producto de software. El apoyo que aporta es teórico y práctico.

BIBLIOGRAFÍA.

[Ber 80] Bersoff, E.H, V.D Henderson and S.G Siegel, Software Configuration Management, Prentice-Hall

[Bab 86] Babich W.A., Software Configuration Management, Addison-Wesley, 1986.

[Som 92] Sommerville IAN, Software Engineering/IAN Sommerville, Addison-Wesley, 1992.

[Dav 91] David Whitgift, Methods and Tools for Software Configuration Management, Jhon Wiley & Suns, 1991

[Roy 93] Roger S. Presma, Ingeniería de Software, MacGraw-Hill, 1993

[StGu 94] Stephen B. Comptom/Guy R. Conner, Configuration Management for Software Library of Congress Cataloging-in- Publication Data, 1994.

[Jac 95] Jacky Stubier, Software Configuration Management, Springer, 1995.

[Ron 92] H. Ronald Berlack, Software Configuration Management, John Wiley & sons, Inc.

[Mar 94] John J. Marciniak, Encyclopedia of Software Engineering, John Wiley & Suns, Inc.

[EIA] Bulletin 4-3, Computer Software Libraries.

[RBPE 95] Rumbaugh, Blaha, Premerlani, Eddy, Lorensen, Modelado y Diseño Orientados a objetos, Prentice Hall, 1995.

[GECC 86] Prapared by General Electric Company Corporate Information System Briggeport, Connecticut, Software Engineering Handbook, McGraw - Hill Book Company, 1986.

[ThMc 93] Richard H. Thayer and Andrew D. McGettrick, Software Engineering A European Perspective, IEEE Computer Society Press, 1993.

[Fed 79] Stuart Y. Feldman, Make-A Program for Maintaining Computer Programs, Software Practice and Experiencie, 1979.

[Tichy 95], Configuration Management, John Wiley & Suns Ltd, 1995.

[IEEE-1028 88] IEEE Std 1028-1988 , Standard for Software Reviews and Audits, IEEE, New York, 1988.

[IEEE-1042 87] ANSI/IEEE Std 1042-1987,Guide To Software Configuration Management, American National Standad / IEEE, New York, 1990.

[IEEE-828 90] ANSI/IEEE Std 828-1990 , Standard for Software Configuration Management Plans, IEEE, New York, 1990.

[IEEE-729 83] ANSI/IEEE Std 729-1983, Glossary of Software Engineering Terminology, IEEE, New York, 1988.

[MIL STD 480A] U.S. Goverment, Department of Defense, Configuration control Engineering Changes, Desviation, and Waiver (supersede MIL STD 480 30 Oct 1968), 1978.

[MIL STD 973] U.S. Goverment, Department of Defense 1992, Configuration Mangement, 1992.

[MIL STD 483] U.S. Goverment, Department of Defense, Configuration Management Practices for Systems, Equipment, Munitions, and Computer Programs, 1970.

[MIL STD 1521A] U.S. Goverment, Department of Defense, Technical Review and Audit for Sistems, Equipment, and Computer Software, 1 June 1976.