Ingeniería del Software

Departamento de Tecnologías de la Información
y Comunicaciones

Curso 2002-2003

Alumna: Laura M. Castro Souto
Profesores: Juan Ares Casal
Javier Andrade Garda

Índice general

Capítulo 1

Introducción a la Gestión de la Calidad en el Proceso Software

1.1. ¿Es necesario mejorar el Proceso Software?

1.1.1. Objetivos en un Proyecto Software

Al abordar cualquier nuevo proyecto se buscan tres objetivos claros:

1. Realización en el plazo estimado.
2. Realización con el coste estimado.
3. Realización de un producto de calidad.

Los dos primeros objetivos se abordan mediante la gestión y planificación de proyectos. El último requiere de un proceso software correctamente gestionado.

La calidad del software se define para cada empresa. Según sus actividades, objetivos, mercado,... se hará más hincapié en ciertos aspectos o en otros (por ejemplo, en el caso del MAP –Ministerio de Administraciones Públicas–, cualquiera que trabaje para organizaciones públicas debe tener establecido *Métrica3*).

Podrían ser parámetros de calidad:

- √ Nivel de errores incluidos (idealmente, 0).
- √ Tiempo medio transcurrido entre fallos (MTBF, *Mean Time Between Failures*).
- √ Satisfacción del usuario (en relación con las funcionalidades y la facilidad de uso): verificación y validación.
- √ Soporte del proveedor (rapidez y efectividad).

Todos estos parámetros deben ser cuantificables y verificables.

1.1.2. Análisis del marco en EE.UU.

La consultora Capers Jones realizó un estudio a finales de los 80 sobre el software adquirido por la Administración Pública de EE.UU. que arrojó los siguientes datos:

- ▷ 5-10 % del software era usable directamente
- ▷ 30-40 % del software nunca se había usado ni llegaría a ser usable

Considerando el volumen de negocio de la administración pública americana, está claro que esto supone un considerable gasto inútil.

¿La solución? Podría pensarse que las pruebas son la solución a la verificación de la calidad del software, pero:

- ↪ El ratio de errores esperable de un programador experto es de 10 defectos por cada 1000 LOC (*line of code*[1]) aproximadamente.
- ↪ El ratio de éxito de las pruebas normalmente es bastante inferior al 50 %. Las pruebas pueden ser de diferentes tipos:
  - ○ unitarias (relativas a un sólo módulo)
  - ○ de integración
  - ○ de sistema
  - ○ de aceptación (las únicas exigidas por ISO-9000)

  Una prueba se considera que tiene éxito si localiza un error.
- ↪ A mayor número de defectos antes de las pruebas, mayor número de defectos latentes en el producto.

Todo esto se ve agravado porque normalmente en la realidad las pruebas no se hacen o se hacen mal.

La verdadera solución es empezar la búsqueda de la calidad *antes*, desde los primeros pasos del proceso de desarrollo. A la salida de las pruebas no tendremos un producto de calidad salvo que en las mismas trabajemos con un producto de calidad.

Los que siguen son las estadísticas proporcionadas por la consultora Capers Jones sobre los principales riesgos y la medida en que afectan en la realidad a los proyectos software.

[1] Según un extenso estudio de 1987, titulado "*Methodology for Software Reliability Prediction*", de J. McCall y otros. Por otra parte, también se usa en ocasiones la medida SLOC, *source line of code*, que es igual que LOC pero no incluye líneas en blanco ni comentarios..

Factor de riesgo	Proyectos afectados
Aumento de requerimientos de usuario	80 %
Excesiva presión de planificación	65 %
Baja calidad	60 %
Desbordamiento del coste	55 %
Inadecuada configuración y control	50 %

Cuadro 1.1: Riesgos en MIS (Management Information System) o Sistemas de Gestión.

Factor de riesgo	Proyectos afectados
Largas planificaciones	70 %
Inadecuada estimación del coste	65 %
Excesivo papeleo	60 %
Módulos propensos a errores	50 %
Proyectos cancelados	30 %

Cuadro 1.2: Riesgos en el software de SS.OO., control de equipos, etc.

Factor de riesgo	Proyectos afectados
Documentación de usuario inadecuada	70 %
Baja satisfacción del usuario	55 %
Excesivo tiempo de salida a mercado	50 %
Acciones competitivas dañinas	45 %
Gastos de litigación	30 %

Cuadro 1.3: Riesgos en el software comercial: paquetes que se compran, software de gestión de empresas, de valores, etc.

Factor de riesgo	Proyectos afectados
Excesivo papeleo	90 %
Baja productividad	85 %
Largas planificaciones	75 %
Aumento de requerimientos de usuario	70 %
Software no usado o no reutilizable	45 %

Cuadro 1.4: Riesgos en el software militar: control de armamento, etc.

Factor de riesgo	Proyectos afectados
Altos costes de mantenimiento	60 %
Fricción cliente-contratado	50 %
Aumento de requerimientos de usuario	45 %
Gastos de propiedad intelectual	20 %

Cuadro 1.5: Riesgos en el software que se externaliza (subcontratado, outsourcing)

Factor de riesgo	Proyectos afectados
No transferible	80 %
Errores ocultos	65 %
Inmantenible	60 %
Aplicaciones redundantes	50 %
Propiedad intelectual	20 %

Cuadro 1.6: Riesgos en el software desarrollado por usuario para uso particular

En vista de estos resultados, desde la administración pública americana se inició un programa para reducir el riesgo en la adquisición del software contratado, patrocinado por el DoD (Departamento de Defensa) y desarrollado por el SEI[2] (*Software Engineering Institute*) en la Carnegie Mellon University: el **CMM** (*Capability Maturity Model*). El objetivo de este programa era poder determinar qué proveedor tenía más capacidad técnica de desarrollar software de calidad.

El **CMM** establece 5 posibles niveles (tipos de empresas):

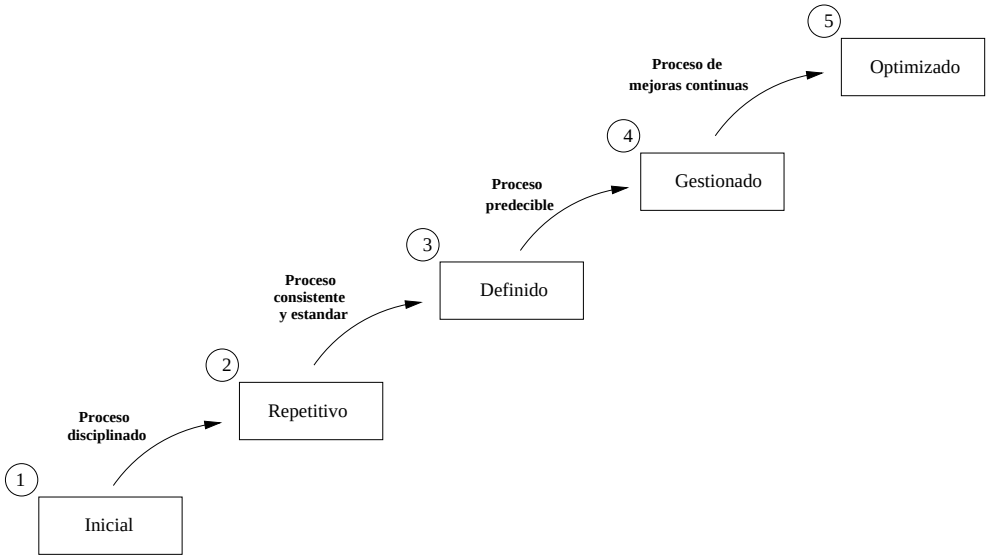


Figura 1.1: Niveles del modelo CMM.

En el nivel inicial el proceso software puede ser poco menos que caótico. Cuando la empresa lo somete a una pauta repetitiva, más o menos estándar y regular, puede pasarse al segundo nivel, en el que la introducción de una mejora se puede aplicar a otros proyectos (documentación, plantillas, herramientas *case*,...). Actualmente, en España se exige a las empresas certificación ISO 9000-3 ó nivel 2 CMM para poder presentarse a concursos de desarrollo de software de la administración pública.

[2]El equivalente europeo es el ESI –European Software Institute–, con sede en Bilbao.

Nivel CMM	Áreas clave
Optimizado	Prevención de defectos Gestión de cambios tecnológicos Gestión de cambios del proceso
Gestionado	Gestión del proceso (métricas) Gestión de la calidad
Definido	Enfoque en el proceso Definición del proceso Programa de formación Gestión integrada del software Ingeniería del producto (estándares) Coordinación entre grupos de trabajo Revisión por iguales (peer 2 peer[a])
Repetitivo	Gestión de requisitos Planificación y seguimiento de proyectos Gestión de configuraciones Gestión y garantía de calidad Gestión de subcontratación
Inicial	–

Cuadro 1.7: Áreas clave del modelo CMM.

[a]La estrategia *Peer 2 Peer* busca la revisión por un "igual", un ingeniero o analista independiente y externo que valide lo realizado.

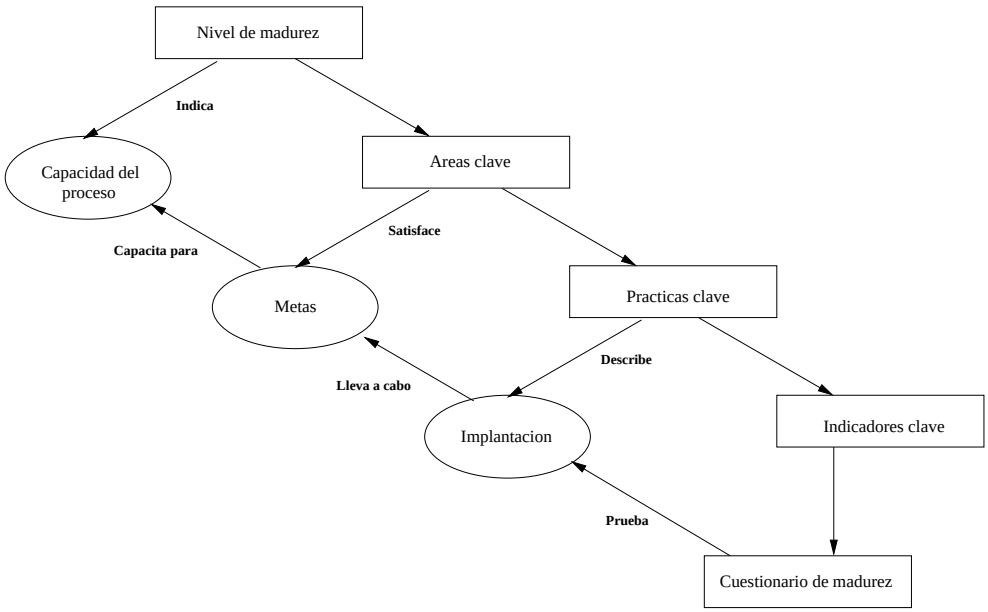


Figura 1.2: Estructura del modelo CMM.

1.1.3. Análisis del marco en Europa

En un estudio del ESI (*European Software Institute*) junto con Gartner Group a mediados de los 90 se obtuvieron los siguientes datos tras entrevistas a 500 directores de Sistemas de Información de toda Europa sobre las creencias con respecto al concepto de calidad:

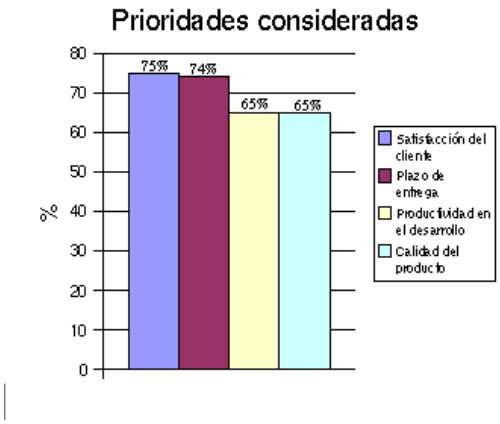


Figura 1.3: Prioridades consideradas en los sistemas de información.

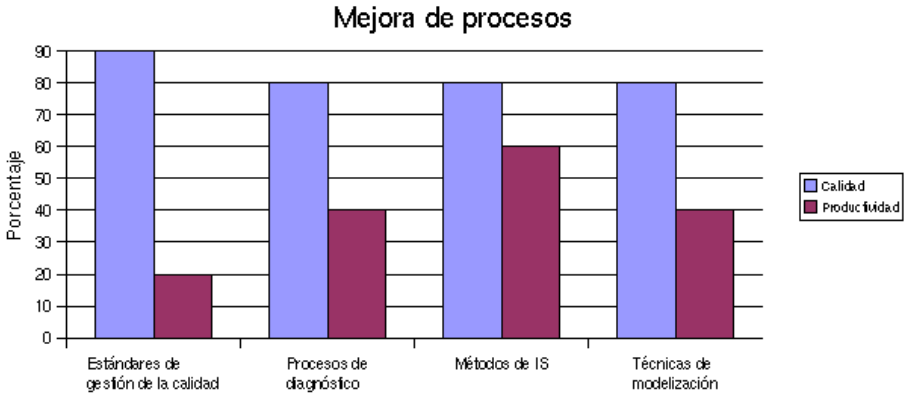


Figura 1.4: Mejora de los procesos: impacto en la productividad

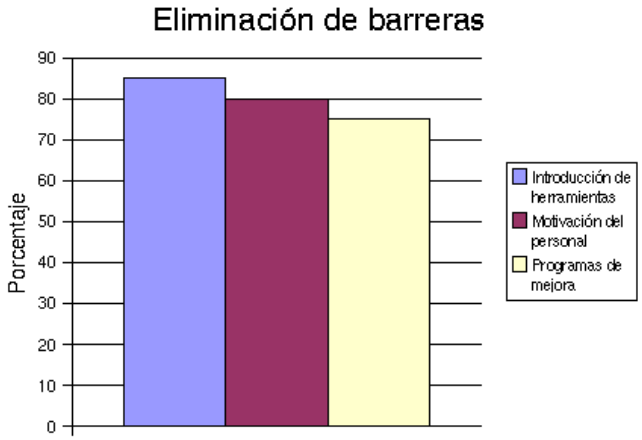


Figura 1.5: Barreras en la mejora de la productividad.

Como se puede apreciar, la idea de los empresarios es que al mejorar la calidad se disminuye la productividad y, recíprocamente, la mejora de la productividad tiene un impacto negativo en la calidad.

Las conclusiones del estudio son:

- Calidad y productividad se ven como elementos contrapuestos e inversamente proporcionales, poco menos que antónimos.

Por suerte, como veremos, la realidad no sigue las creencias que revela el estudio. De hecho, es el *mantenimiento* la fase más costosa en cualquier ciclo de vida y/o desarrollo. En un estudio de Gartner Group del año 95 se evaluó en un 95 % el porcentaje de recursos, costes, etc. que se dedica al mantenimiento.

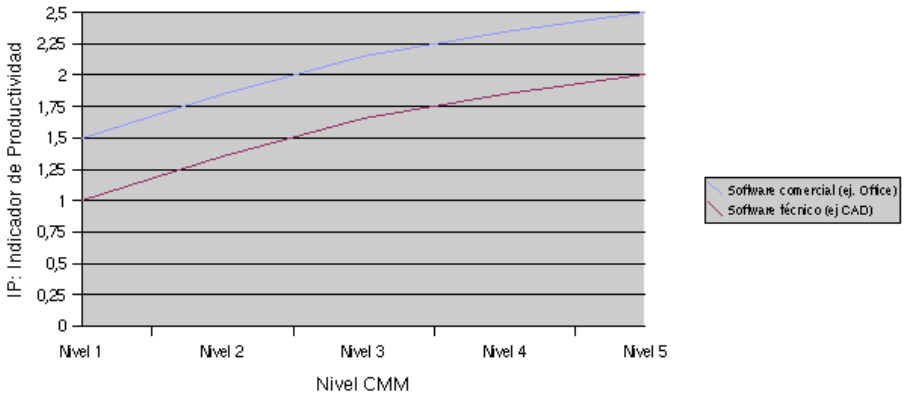


Figura 1.6: Relación productividad-niveles CMM.

1.1.4. Realidad en la productividad y calidad

Estudios económicos mostraron que el ROI (Retorno de la Inversión) compensa con creces el esfuerzo de la mejora en pro de la calidad.

La media obtenida tras el análisis de resultados en 13 grandes empresas fue:

Inversión anual en mejora de procesos	245.000 $
Años en actividad de mejora	3,5
Inversión por ingeniero de software	1.375 $
Incremento anual de la productividad	35 %
Reducción anual en los plazos	20 %
Reducción anual en errores post-entrega (mantenimiento)	39 %
Retorno de la inversión (por cada $)	5 $

Reducción de costes de detección y reparación de defectos del software en proporción al coste total del sistema frente al incremento de productividad en Raytheon, primer fabricante mundial de misiles (fuente: Report in Scientific American):

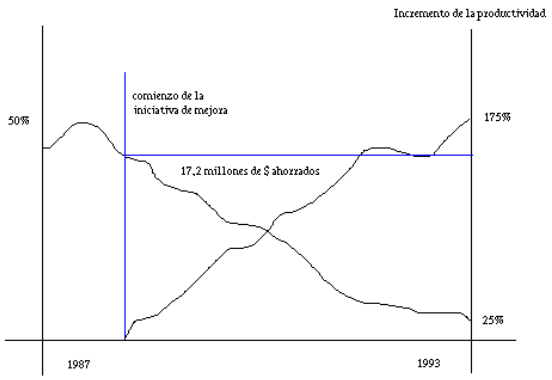


Figura 1.7: Reducción de costes vs. incremento de la productividad.

En el congreso de la OTAN de 1967 en Garmisch (Alemania) se *certificó* la crisis por la que estaba pasando el desarrollo del software (no se realizaba desde el punto de vista ingenieril). Se necesitaba mejorar el proceso software, como punto de partida para poder mejorar el producto software en sí.

Para solventar esta crisis se empezó a trabajar en los diferentes componentes y factores implicados en el desarrollo de software, generalmente conocidos como "las cuatro P":

- ⋆ **P**roducto (se buscan métricas y metodologías para su desarrollo)
- ⋆ **P**roceso (la clave de un buen producto comienza por un buen proceso, idea que adoptan CMM, ISO 9000-3...)
- ⋆ **P**ersonas (hay que mejorar su capacidad)
- ⋆ **P**roblema (es necesario su entendimiento y conceptualización)

¿Qué se pretende realmente con la mejora en el proceso de desarrollo de software?

- mejorar la calidad del producto final
- aumentar la capacidad de control a lo largo del desarrollo y mantenimiento del producto
- reducir los plazos de desarrollo y los costes de mantenimiento

A la hora de implantar mejoras:

↪ la tecnología suele suponer el 20 % del trabajo

↪ el cambio organizativo conlleva el 80 % del esfuerzo

El porcentaje de proyectos de mejora que tienen éxito, es decir, que se implantan, es relativamente bajo (del orden del 35 %). Esto es debido a las presiones, el día a día, etc. Si no se está convencido de la mejora, acabará por cancelarse (más costes, incredulidad, rechazo, prioridad de otras cosas...).

1.1.5. Firmas Europeas. Tendencias

En EE.UU. se maneja principalmente CMM. En Europa se empleó inicialmente ISO 9000, porque se adapta a cualquier empresa. Se elaboró una *Guía ISO 9000-3* especialmente adaptada para desarrollo y mantenimiento de software, ya que la ISO 9000 genérica[3] no lo hacía.

Actualmente se está migrando desde ISO 9000 hacia referencias más comparables (*benchmarkables*, ISO 9000 no lo es, es un ránking), principalmente CMM (que aporta no sólo su comparación entre niveles 1 a 5 con estancos 2, 3 y 4, sino la posibilidad de utilizar valores como $1'25$, $1'75$, etc).

√ El objetivo actual es obtener nivel 3 CMM (idealmente).

√ Hay problemas con la KPA (*Key Process Area*) de nivel 2.

1.2. Calidad, ISO-9000 y software

En primer lugar, haremos una aclaración importante. ISO-9000 es una familia de normas de calidad (alrededor de 20 requisitos), que se compone de 3 miembros: ISO-9001, ISO-9002 e ISO-9003. Cada uno de los miembros es una adaptación de la filosofía común de ISO-9000 a distintos casos según los criterios que se cumplen o no. Por su parte, ISO-9000-3 es la adaptación de ISO-9001 al desarrollo de software y no debe confundirse con la ISO-9003.

Veremos ISO-9000 como estándar de gestión de calidad porque sirve para todas las empresas y tiene más antigüedad[4].

[3] ISO 9000 y en particular ISO 9000-3 es un modelo de calidad global que integra un gran número de actividades: hardware, software, fabricación, administración, mantenimiento,...

[4] Para indicar el año de vigencia de una revisión ISO, se hace del siguiente modo: ISO-9000:2000 sería la revisión del año 2000 de la ISO-9000.

1.2.1. Evolución de la calidad

Teniendo como objetivo alcanzar un nivel **TQM** (*Mantenimiento –Gestión– Total de la Calidad*), y partiendo de un inicio en el que simplemente se realizan pruebas al producto antes de sacarlo al mercado, la evolución para por una serie de fases en las que primero las pruebas se trasladan a la cadena de producción, posteriormente se posee un sistema de calidad integral (que involucra a toda la empresa), llegando a alcanzar idealmente una mejora continua basada en un ciclo PDCA sobre una base estable (repetible):

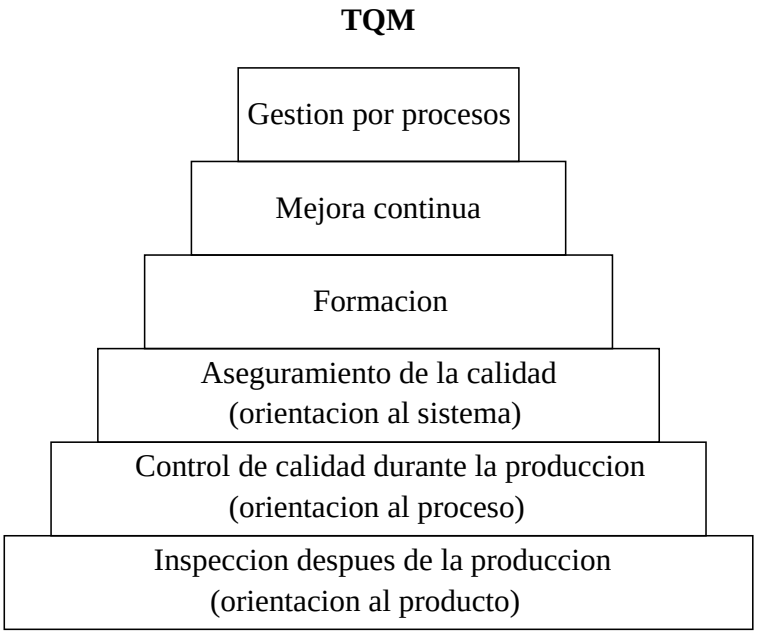


Figura 1.8: Niveles en la evolución de la calidad

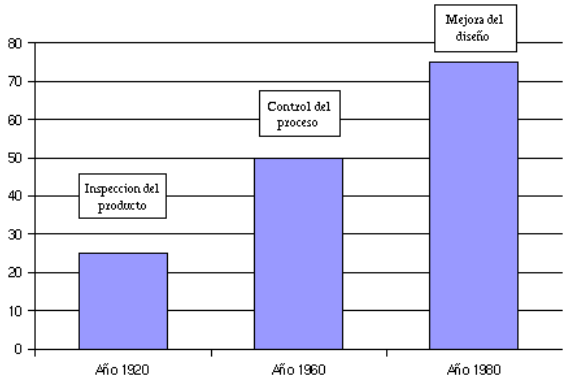


Figura 1.9: Evolución de la calidad

Como se puede ver en la gráfica anterior, en los años 50 el proceso software tenía como único control de calidad las pruebas al producto antes de su puesta en operación;

en los años sesenta el control de todas las actividades del proceso seguido se había extendido considerablemente, y a principios de los 80 se instaura la detección de problemas antes del proceso (cuando se conceptualiza).

Algunos principios básicos de la calidad son:

- ▷ orientación al cliente (tanto interno –compañeros de proyecto– como externo; cliente no tiene por qué significar necesariamente usuario)
- ▷ compromiso de la dirección
- ▷ decisiones basadas en hechos y datos
- ▷ implicación de todos los empleados y su motivación
- ▷ gestión por procesos: identificación, definición y actuación sobre ellos para mejorarlos
- ▷ mejora continua
- ▷ aseguramiento independiente de la calidad: certificación de que lo que se hace es lo que se debería hacer (se busca alguna entidad independiente, ajena, externa, que supervise y audite el trabajo)
- ▷ sistemas concertados cliente/proveedor: condiciones de aceptación del producto desarrollado preestablecidas
- ▷ establecimiento de objetivos a corto, medio y largo plazo al implantar un sistema de calidad

La *gestión por procesos* consiste en identificar los procesos de negocio y sus entradas y salidas relacionadas, con el fin de elaborar un *mapa de procesos*.

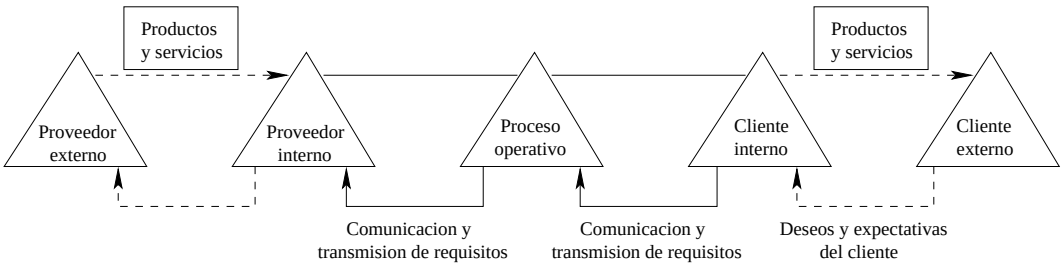


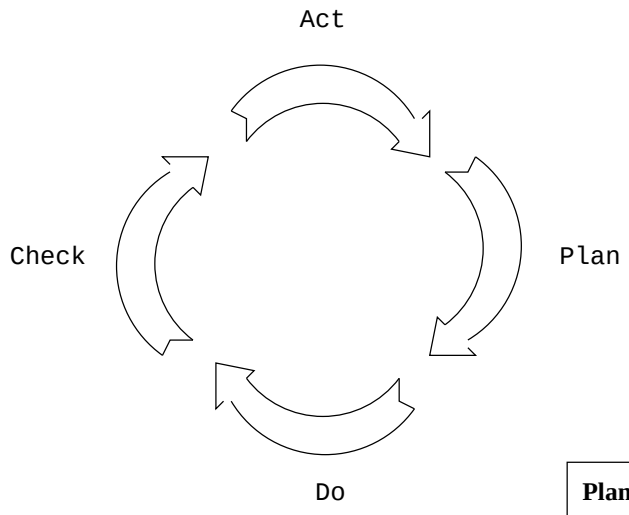
Figura 1.10: Esquema de un mapa de procesos.

Es necesario identificar (principios básicos de la gestión por procesos):

- ▶ objeto del proceso, entradas y salidas
- ▶ responsable del proceso
- ▶ requisitos del cliente
- ▶ evaluación de la conformidad con los requisitos del cliente

- medidas de control del proceso
- oportunidades de mejora en el proceso, manejando dichas medidas
- priorización de oportunidades de mejora y fijación de objetivos para mejorar la eficiencia del proceso

La mejora continua se lleva a cabo a través de un ciclo de mejora continua, de **PDCA** o *ciclo de Deming* (personaje que fue el "padre" de la calidad).



Plan (planificar lo que se quiere realizar)

Do (realizar lo planificado)

Check (comprobar los resultados de lo realizado)

Act (actuar para modificar las realizaciones y replanificar con el fin de asegurar los mejores resultados −se vuelve de nuevo al inicio−)

Figura 1.11: Ciclo de Deming de mejora continua o PDCA.

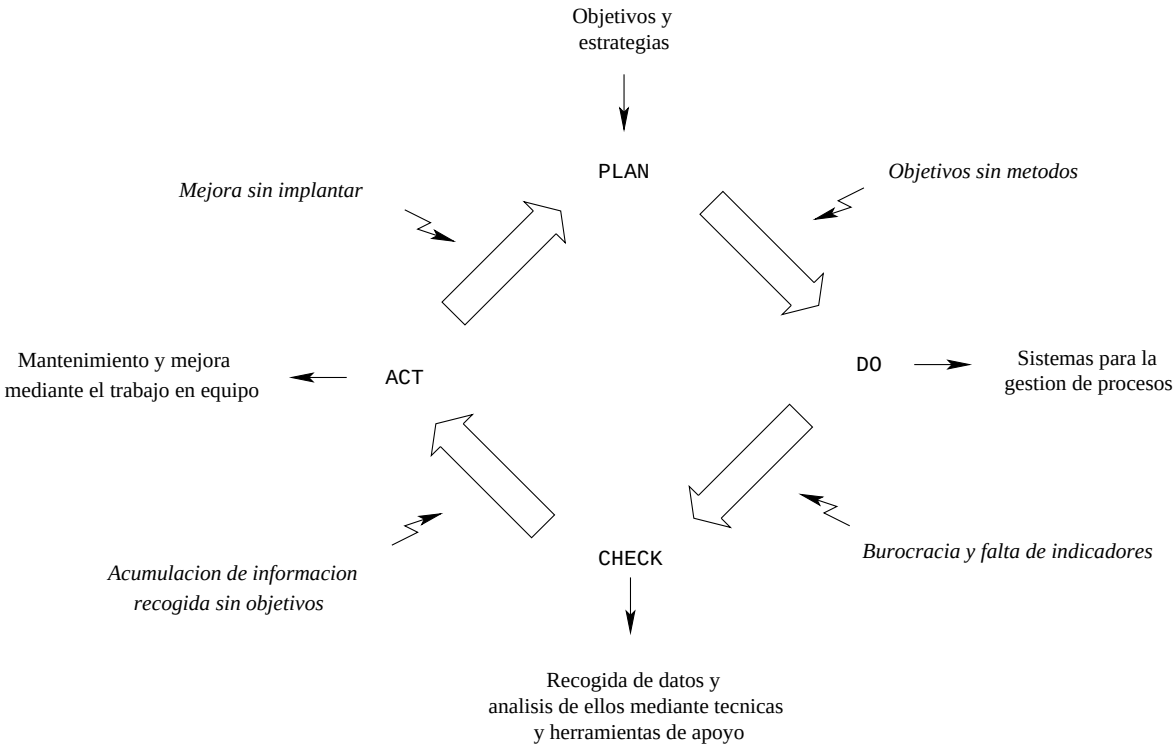


Figura 1.12: Fases, objetivos y riesgos del ciclo de Deming.

El SEI ha desarrollado un modelo, **IDEAL**, similar al PDCA, basado en los pasos: *inicio*, *establecimiento*, *actuación*, *aprendizaje* y vuelta a la *diagnosis*. En este modelo IDEAL del SEI para la mejora continua se consideran las fases:

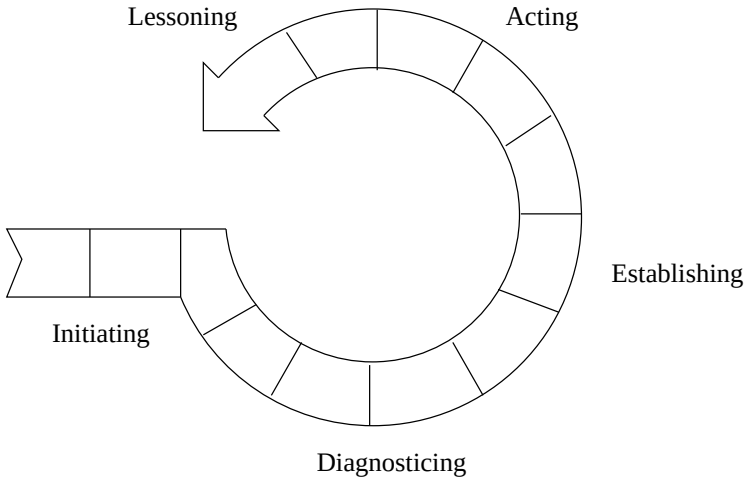


Figura 1.13: Esquema del modelo IDEAL del SEI.

Que implican:

Iniciar Implicar a la dirección, especificar los objetivos de negocio de la mejora, identificar impactos en otras acciones en curso y definir las infraestructuras de gestión asociadas.

Diagnosticar Identificar estado actual, identificar los objetivos, respecto del modelo seleccionado y las acciones necesarias para el cambio.

Establecer Priorizar las acciones que se llevarán a cabo en el marco de la mejora y planificar detalladamente los recursos y actividades que se seguirán.

Actuar Poner en operación el plan (con la asignación de recursos internos, externos y herramientas), implantar las mejoras en un proyecto piloto, afinar las acciones y difundir los resultados.

Lecciones aprendidas Validar los resultados obtenidos, identificar las lecciones aprendidas en el proyecto y derivar las recomendaciones oportunas para futuros ciclos de mejora.

1.2.2. ¿Qué es la calidad?

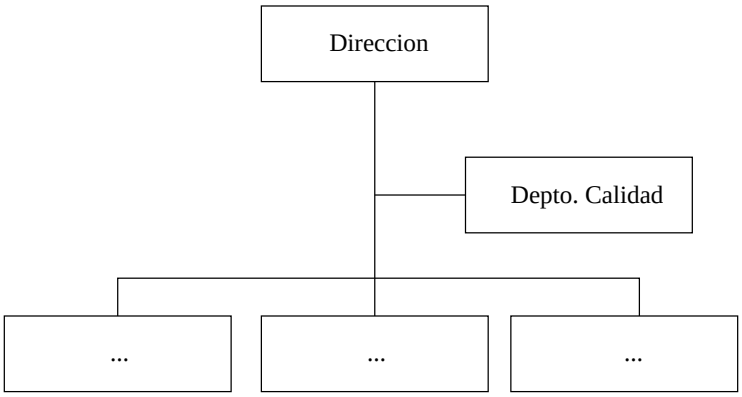
La calidad del software no es algo que se considere una vez se ha codificado (pruebas). La **SQA** es una actividad de protección que se aplica a lo largo de todo el proceso de Ingeniería del Software y que engloba:

- un enfoque de gestión de calidad

- métodos, técnicas y herramientas de IS
- revisiones técnicas formales aplicadas durante el proceso
- estrategia de pruebas incremental
- control de cambios (gestión de la configuración)
- un procedimiento para ajustarse a los estándares de desarrollo
- mecanismos de medición y generación de informes

Como *función*, la **calidad** es una visión independiente (externa) en el proyecto o empresa (el área auditada) dependiente de la Dirección y directamente vinculado a ella (sólo informa a la Alta Dirección, que delega en ella la cuestión de calidad, los demás responden ante este staff).

Su labor es velar por el buen hacer, según lo definido (y explícitamente documentado) en el Sistema de Calidad, en la empresa y en sus proyectos informando de las desviaciones a la Dirección (que delega, pues, el trabajo pero no la responsabilidad).



Hay dos funciones muy importantes directamente relacionadas con la calidad:

1. **Actividades de aseguramiento de la calidad**, como por ejemplo auditorías de fin de fase. Aseguran que se sigue el proceso definido, los estándares y las herramientas marcadas.
2. **Actividades de control de la calidad**, como la revisión por técnicos de los productos entregables (tanto documentos como software). Se controla la calidad técnica de los productos (*peer to peer*).

1.2.3. Problemas con la calidad en el software

Pueden enumerarse los siguientes escollos a la hora de hablar de calidad en el software:

- Se parte de una definición vaga y a veces incorrecta del proyecto.

- Los requisitos cambian durante el proyecto y la gestión de requisitos y/o una ERS (Especificación de Requisitos Software) no existen.
- El cliente no participa en el proyecto, ni en su definición ni en su desarrollo.
- El cliente pide constantemente ampliaciones y cambios sobre los requisitos iniciales.
- Cada diseñador o programador trabaja a su aire, de modo que es imposible entender o reutilizar nada por falta de estándares.
- Nunca están disponibles ni accesibles versiones de los elementos del proyecto. No se controlan los cambios.
- La planificación del proyecto es ficticia o inexistente y tampoco se ha controlado su evolución en tiempo y recursos.
- Al final, con las prisas, se ahorra esfuerzo en las pruebas y en la documentación.
- Los cambios de última hora debidos a la entrega inminente del software generan nuevos problemas.
- Se quiere implantar un sistema de calidad pero faltan conocimientos, requisitos y recursos.
- Cuando se quiere hace esto, se dota al sistema de demasiada burocracia.

1.2.4. Requisitos ISO 9000

ISO 9000 es una **norma de gestión de la calidad** que certifica el *proceso*, la actividad, no los resultados (el producto). El padre de la gestión de la calidad fue Philip Crosby.

Un **sistema de aseguramiento de la calidad** se puede definir como la estructura organizativa, responsabilidades, procedimientos, procesos[5] y recursos para implementar la gestión de la calidad.

ISO 9000 describe los elementos de garantía de calidad (requisitos) en términos genéricos aplicables a cualquier negocio con independencia de los productos o servicios ofrecidos.

Verificación de requisitos ISO 9000

Para certificarse con uno de los tres modelos ISO 9000 el sistema de calidad es examinado minucionsamente por auditores externos para comprobar si se ajusta a los estándares y a la operativa efectiva. Después de un examen correcto (es decir, sin

[5]Un proceso está normalmente formado por varios procedimientos, que a su vez se modelan en base a plantillas.

hold points[6] y con no demasiados **on going improvement**s[7], la compañía recibe un certificado avalado por los auditores.

Para asegurar el ajuste continuado a los estándares se realizan auditorías de seguimiento. La auditoría se hace con respecto a ISO 9000 (9001, 9002, 9003) y sistema de calidad.

Los propios auditores están organizados jerárquicamente y se controlan entre ellos de la siguiente manera:

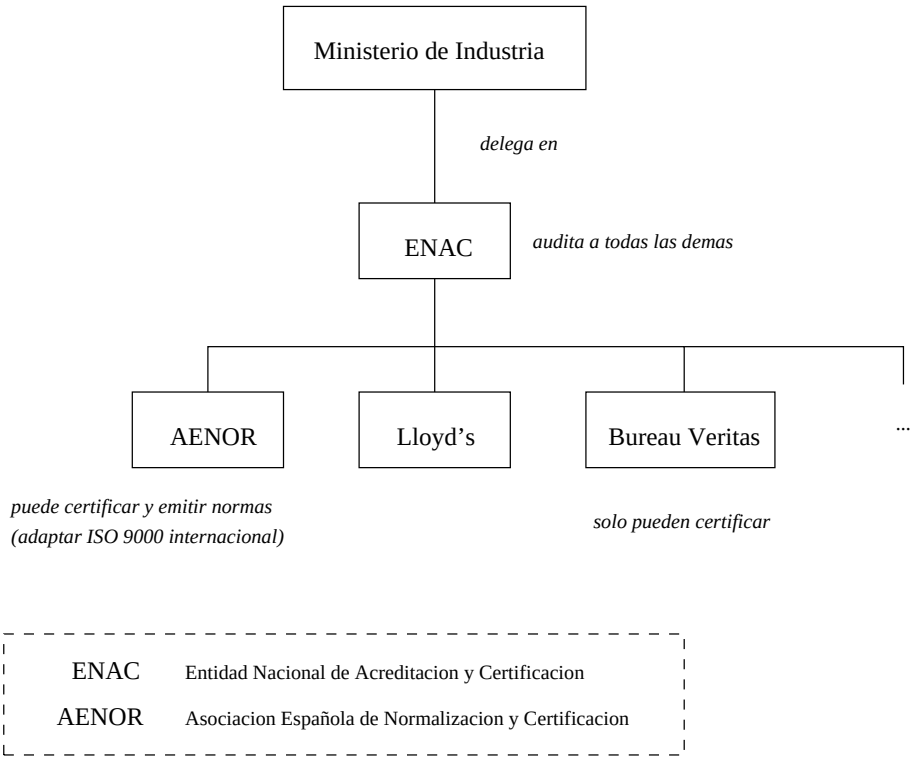


Figura 1.14: Jerarquía nacional de auditores.

Una certificación ISO 9000 requiere:

- √ Diseño del sistema de calidad.
- √ Implementar el sistema de calidad[8]
- √ Después de un mínimo de tres meses de registros de calidad para que el sistema sea considerado auditable, se solicita presupuesto a empresas de auditoría o certificadoras.
- √ Auditoría inicial: tras un estudio de la documentación y una visita previa, se realizará la auditoría inicial, que tendrá su correspondiente Informe. Se dispone de un plazo de treinta días para solucionar no-conformidades detectadas, con evidencias de que las acciones correctivas se están llevando a cabo.

[6]Falta grave.
[7]Falta leve.
[8]Estos dos pasos suelen realizarse paulatinamente y abarcar de 6 a 12 meses.

- √ Dicho informe y las sesiones correctivas derivadas pasarán por una comisión que decidirá si la empresa es apta para certificarse o no.

Para renovar la certificación:

- √ Cada año se harán auditorías de seguimiento.
- √ Cada tres años se realizará una auditoría de renovación (recertificación).

Otras consideraciones generales:

- √ Desde que se llama a la certificadora hasta que se está certificado suelen pasar seis meses.
- √ Las auditorías de seguimiento imponen una mejora continua (ciclo PDCA) además de cumplir el sistema de calidad establecido.
- √ Los modelos de aseguramiento ISO 9000 tratan a la empresa como una red de procesos interconectados.
- √ Para que un sistema de calidad se ajuste a ISO 9000, estos procesos (figura 1.10) deben afrontar áreas identificadas en el estándar y se deben documentar[9] y practicar como se ha descrito.
- √ Documentar un proceso ayuda a que una organización lo entienda, controle y mejore[10].
- √ Es la oportunidad de comprender, controlar y mejorar la red o el mapa de procesos.
- √ Es quizá el beneficio más grande para las empresas que diseñan e implementan los sistemas de calidad conforme a ISO 9000.

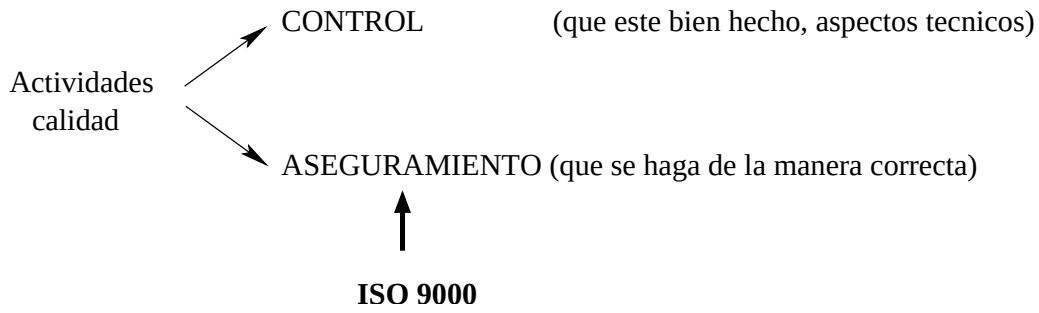
ISO 9000 describe, en términos generales, los elementos de un sistema de aseguramiento de la calidad. Estos elementos incluyen la estructura organizativa, procedimientos, procesos y recursos necesarios para implantar la planificación, el control, la garantía y la mejora de la calidad.

Sin embargo, ISO 9000 no describe cómo una empresa debería implantar estos elementos del sistema de calidad.

Por tanto, el reto se encuentra en diseñar e implantar un sistema de garantía de calidad que cumpla los estándares y considere los productos, servicios y cultura de la empresa.

[9] La excesiva exigencia de documentación es una de las cosas que se le criticaba a la norma ISO 9000 del 94 y que se corrige en la versión del 2000. Como herramientas de documentación podemos citar PLATÓN, que usa tecnología Lotus Notes y ARA, que usa tecnologías web y Java, ambas del grupo NorControl Soluziona.

[10] No en vano la base de una mejora es un proceso repetible.



1.2.5. Los tres modelos ISO 9000

Cada uno de los tres modelos ISO 9000 contempla unas circunstancias de aplicabilidad:

ISO 9001 Cumple los 20 requisitos de ISO 9000

- Se realiza diseño y fabricación del producto.
- El sistema de Calidad es un Modelo de Aseguramiento de Calidad en Diseño/Desarrollo, Producción, Instalación y Servicio Post-Venta.

ISO 9002 Cumple todos los requisitos ISO 9000 salvo el 4.4, referido a Control de Diseño

- No se realiza el diseño del producto (sólo se fabrica).
- El sistema de Calidad sólo es un Modelo de Aseguramiento de Calidad en Producción e Instalación.

ISO 9003 Cumple todos los requisitos ISO 9000 salvo el 4.4, el 4.6, referido a Compras, y el 4.19, referido al Servicio Post-Venta

- Se considera que la empresa no tiene proceso: almacenaje, manipulación o embalaje de producto.
- El sistema de Calidad es un Modelo de Aseguramiento de Calidad en Inspección Final y Pruebas.

Otras normas ISO son:

ISO 8402
Vocabulario de Gestión y Aseguramiento de la Calidad.

ISO 9000
Standard para la Gestión y Aseguramiento de la Calidad.

ISO 9004
Gestión de Calidad y Elementos del Sistema de Calidad.

ISO 10012
Requisitos de Aseguramiento de Calidad para Equipos Especializados.

ISO 10013
Guías para el Desarrollo de Manuales de Calidad.

ISO 14000
Standard para la Gestión y Aseguramiento de la Calidad en temas de Medio Ambiente.

1.2.6. Requisitos de la norma ISO 9001

Se enumeran a continuación:

1. Sistema de Gestión (*6 requisitos*)
2. Ciclo de Vida del Producto (*7 requisitos*)
3. Actividades de Soporte (*7 requisitos*)

De manera que los requisitos a cumplir en un programa de mejora acorde con ISO 9001 se desglosan en:

1. Requisitos del Sistema de Gestión de Calidad:
   a) Responsabilidad de la Dirección
   b) Sistema de Calidad
   c) Control de la Documentación (versiones, difusión, comprensión...)
   d) Acciones Correctivas y Preventivas
   e) Control de los Registros de Calidad
   f) Auditorías Internas de Calidad
2. Requisitos del Ciclo de Vida del Producto:
   a) Revisión del Contrato
   b) Control del Diseño[11]
   c) Control del Proceso
   d) Estado de las Inspecciones y Pruebas
   e) Control de No-Conformidades de los Productos
   f) Manipulacion, Almacenamiento y Entrega del Producto
   g) Servicio Post-Venta
3. Requisitos de las Actividades de Soporte:
   a) Compras
   b) Control del Producto Suministrado por el Cliente
   c) Identificación y Trazabilidad[12] del Producto
   d) Inspecciones y Pruebas
   e) Control del Equipamiento para Inspección, Medida y Pruebas
   f) Formación
   g) Técnicas Estadísticas

[11] ISO 9000-3 para interpretarlo relativo al software, conforme a un análisis, requisitos, codificación, pruebas...
[12] Poder seguir los cambios hacia atrás; Gestión de la Configuración en IS.

Se ha desarrollado, para su adecuación al proceso software, un conjunto especial de directrices ISO para ayudar a interpretar el estándar para su uso en el proceso software: ISO 9000-3.

ISO 9001 es,pues, un estándar de Garantía de Calidad que contiene 20 requisitos que deben estar presentes en un sistema de garantía de calidad efectivo. Dichos requisitos se enfrentan con los siguientes temas:

1. Responsabilidad de la gestión
2. Sistema de Calidad
3. Revisión del Contrato
4. Control del Diseño
5. Control de los Datos y Documentos
6. Compras
7. Control de Producto Suministrado por el Cliente
8. Identificación y Posibilidad de Seguimiento del Producto
9. Control del Proceso
10. Inspección y Prueba
11. Control de Inspección, Medición y Equipo de Pruebas
12. Inspección y Estado de Prueba
13. Control de Producto No Aceptado
14. Acción Correctora y Preventiva
15. Tratamiento, Almacenamiento, Empaquetamiento, Preservación y Entrega
16. Control de Registros de Calidad
17. Auditorías Internas de Calidad
18. Formación (registro de, no es necesaria una planificación total)
19. Servicios
20. Técnicas Estadísticas

Filosofía de la norma ISO 9001

La abstracción de sustituir los títulos de cada requisito por las acciones que representa, refleja una empresa funcionando bajo ISO 9001:

> *Dirección decide implantar el sistema (requisitos 4.1 y 4.2), llegan los pedidos y se revisan (4.3), se diseñan los productos pedidos (4.4), se documenta todo perfectamente (4.5), se compran los materiales (4.6), se controlan los productos suministrados por el cliente (4.7), se identifican adecuadamente las compras, semielaborados y acabados (4.8), se fabrica controlando los*

procesos (4.9), se inspeccionan los productos que se generan (4.10), se controlan los equipos de inspección (4.11), se identifica el estado de inspección y se controlan los productos no conformes (4.12, 4.13), se toman las acciones correctoras y preventivas pertinentes (4.14), se embala, almacena, conserva y entrega adecuadamente el pedido (4.15), se archivan todos los registros de la calidad (4.16), se efectúan auditorías internas (4.17), se prevé que sea necesaria la formación del personal (4.18), se establece el servicio post-venta (4.19), y, para terminar, se procesan estadísticamente todos los registros (4.20).

1.2.7. Guía para la aplicación al software

ISO 9000-3 presenta estándares para la Gestión y Aseguramiento de la Calidad y en su apartado 3 una Guía para la aplicación de la Norma ISO 9001 al desarrollo, provisión y mantenimiento de software. Conceptualmente, no añade requisitos a ISO 9001, aunque puede reorganizarlos o dividirlos.

ISO 9000-3 posee los siguientes 22 aspectos clave:

1. Marco (*4 elementos*)
2. Ciclo de Vida del Producto (*9 elementos*)
3. Actividades de Soporte (*9 elementos*)

Su desglose es el siguiente:

1. Marco
   a) Responsabilidad de la Dirección
   b) Sistema de Calidad
   c) Auditorías Internas del Sistema de Calidad
   d) Acciones Correctivas y Preventivas
2. Ciclo de Vida del Producto
   a) Revisión del Contrato
   b) Especificación de los Requisitos del Cliente
   c) Planificación del Desarrollo
   d) Planificación de la Calidad
   e) Diseño e Implementación
   f) Pruebas y Validación
   g) Aceptación (Pruebas)
   h) Réplicas, Entrega e Instalación
   i) Mantenimiento
3. Actividades de Soporte
   a) Gestión de Configuración

b) Control de la Documentación del Sistema de Calidad
c) Registro de Calidad
d) Métricas
e) Prácticas, Responsabilidades y Convenciones
f) Herramientas y Técnicas
g) Compras
h) Control del Software Incluido
i) Formación

Requisitos ISO 9001:94

1. Sistema de Gestión

   a) Responsabilidad de la Dirección

   - *Política de Calidad* (4.1.1), mostrando el compromiso con la Calidad de la organización, acorde con las expectativas del cliente (la calidad siempre se ve desde el punto de vista del cliente).

     La Política de Calidad debe ser visible para todo el personal y entendida.
   - *Organización* (4.1.2), mostrando el organigrama de las actividades de la empresa y las responsabilidades esenciales.

     Es conveniente (prácticamente obligado) incluir una descripción de puestos.
   - *Recursos* (4.1.2.2), mostrando la formación y disponibilidad para llevar a cabo las actividades de calidad.
   - *Representante* (4.1.2.3) del Equipo de Dirección responsable del Sistema de Calidad.
   - *Revisión por Dirección* (4.1.3), monitorizando la validez del Sistema de Calidad a ciertos intervalos de tiempo.

   b) Sistema de Calidad

   - *Necesidad* (4.2.1) de disponer de un Sistema de Calidad.

     Formaliza y describe los procedimientos (Manual de Calidad). Muestra el modo de trabajo a externos.
   - Los *procedimientos del Sistema de Calidad* (4.2.2) deben ser consistentes con los estándares ISO 9000 e implantados efectivamente en la organización (¡son los únicos requisitos! ¡no indica nada más!).

     Los procedimientos incluirán estándares de codificación, de documentación, de mantenimiento, de pruebas, de gestión de proyectos, de gestión de la configuración...
   - Cada proyecto, producto o servicio debe *planificar la calidad* en el mismo (4.2.3).

Los planes de calidad no repetirán información estándar, sino las especificaciones (especifidades, desviaciones de lo pactado) de cada proyecto.

c) Control de la Documentación del Sistema de Calidad

- La documentación asociada al Sistema de Calidad (4.5.1) ha de ser *controlada en sus versiones*.

  No debe haber copias en circulación no actualizadas.
  Los cambios en ella deben llevarse a cabo de manera controlada.
  Algunos elementos de este epígrafe: política de calidad, manual de calidad, procedimientos, metodología de diseño...
- Los documentos son *aprobados* (4.5.2) por una persona con el nivel de autoridad apropiado.
- Los cambios y revisiones (4.5.3) han de ser aprobados por las mismas funciones/organizaciones que las llevaron a cabo en la edición inicial (aunque ya no sean encarnadas por la misma persona, sí el mismo cargo —en caso de que desapareciera, el Director debe designar un nuevo responsable y se debe actualizar esta información y la de reorganización en los manuales...—).

d) Acciones Correctoras y Preventivas

- Es *preciso un procedimiento* (4.14.1) para la atención a las acciones correctivas y preventivas.
- El *procedimiento de acciones correctivas* (4.14.2) debe contemplar:

  Gestión de reclamaciones del cliente.
  Análisis de las causas de no-conformidades.
  Control de cierre de las acciones.
- El *procedimiento de acciones preventivas* (4.14.3) debe considerar:

  Fuentes de la información para detectar los orígenes de las no-conformidades.
  Inicio y control del cierre.
  Asegurar que la información adecuada es presentada a la dirección.

e) Control de los Registros de Calidad

- *Necesidad* (4.16) de establecer y mantener un conjunto de procedimientos para:

  Identificar
  Recoger
  Acceder
  Completar
  Almacenar y
  Mantener los registros de calidad (evidencias documentales del proceso).

- Registros serán aquellos documentos que no pueden ser cambiados (demostradamente): Actas, Informes de Revisión, Informe de Auditorías, No-Conformidades, Hojas de Acción Preventiva, Hojas de Acción Correctiva, Contratos, etc. Quedan fuera los documentos de calidad.

f) Auditorías Internas de Calidad

- *Necesidad* (4.17) de disponer y mantener procedimientos que aseguren la planificación y realización de auditorías internas de seguimiento del Sistema de Calidad en los desarrollos, verificando su validez.
- Las auditorías deberán ser realizadas por personal con la experiencia apropiada y externos al área de realización auditada.
- Pueden ser planificadas como auditorías de final de fase.

2. Ciclo de Vida del Producto

a) Revisión del Contrato

- *Necesidad* (4.3.1) de establecer y mantener procedimientos documentados para la revisión de los contratos: se incluyen en este punto contratos, pedidos, ofertas...
- Previamente al envío de una oferta o de la aceptación de un pedido, éstos deben ser revisados para comprobar que:

  Los requisitos han sido reflejados de la manera apropiada, las diferencias entre oferta y contrato son entendidas y aceptadas y la organización está en disposición de poner en operación el sistema solicitado.
  Es conveniente definir una "checklist" de revisión que guíe a la persona encargada de realizarla y unifique los criterios.
- Debe ser recogido el *proceso de inclusión de revisiones* (4.3.3) al contrato.
- Los resultados de la revisión deben ser *registrados* (4.3.4).

b) Control del Diseño

- *Necesidad* (4.4.1) de disponer de un procedimiento para la revisión del diseño asegurando que contempla los requisitos especificados por el cliente.

  Se incluye documentación del diseño, de manual de usuario, de mantenimiento y de formación.
  Deben incluirse entradas, métodos, captura de experiencia en otros proyectos y salidas.

- Las actividades de diseño deben ser *planificadas* (4.4.2), su responsabilidad definida y asignada a personas con la formación apropiada.
- Las *interacciones entre grupos* (4.4.3) deben ser definidas y su seguimiento establecido[13].

[13]Es decir, estos dos puntos señalan la necesidad de establecer un Ciclo de Vida.

- Las *entradas a la fase de diseño* (4.4.4) deben ser recogidas y revisadas previamente (recoger y revisar requisitos, algo que ya se comentó en oferta y contrato).
- Las *salidas* del diseño deben ser documentadas (4.4.5) de manera que puedan ser validadas contra los requisitos de entrada.
- De manera planificada, las salidas del diseño deben ser *revisadas* (4.4.6).
- De manera planificada, se llevarán a cabo *verificaciones* (4.4.7) del diseño comprobando la correlación entre los requisitos de entrada y salida del diseño.
- La *validación del diseño* (4.4.8) se realizará para asegurar que el producto satisface los requisitos del cliente.
- Los *cambios y modificaciones* al diseño (4.4.9) deberán controlarse.

c) Control del Proceso

- Se identificará y planificará el diseño, instalación y puesta en servicio de procesos que afecten directamente a la calidad, asegurándose de que éstos se llevan a cabo de manera controlada.
- Este apartado se considerará detalladamente en el Ciclo de Vida que se defina dentro de la empresa[14].

d) Estado de las Inspecciones y Pruebas

- El *estado de las actividades de inspección y pruebas* (4.12) deberán permitir identificar qué elementos han pasado las actividades de inspección y pruebas de manera satisfactoria (para software: y en qué nivel se han pasado —unitario, global...—).

e) Control de No-Conformidades de los Productos

- Los productos no-conformes deberán ser *identificados* (4.13.1) y no serán puestos en operación por el cliente (no se le presentarán).
- La responsabilidad de la *realización de las revisiones de no-conformidades* (4.13.2) debe ser definida (ya que cuando se detecta algo mal no sólo no se sigue, sino que hay que decidir qué se hace).
- Opciones definidas en ISO 9001:

  Modificar el producto hasta cubrir el/los requisitos faltantes.
  Aceptar el producto con o sin modificaciones (¡no hay productos perfectos!).
  Sustitución por elementos alternativos.
  Rechazo.
- Este punto no trata de conseguir software sin errores (para eso tenemos las acciones preventivas), sino de qué hacer cuando se detecta una no-conformidad (acciones correctivas).

f) Manipulación, Almacenamiento y Entrega del Producto

[14] Así, lo primero que haríamos al llegar a una empresa sería Gestión y Planificación de proyectos e instauración de un (o varios) Ciclo de Vida.

- *Definir procedimientos* para manipulación, almacenamiento y entrega del producto (4.15).

  En Ingeniería del Software, esto es Gestión de la Configuración.

g) Servicio Post-Venta

- En los casos en que sea necesario (no existe obligación[15]), se establecerá el *procedimiento* que regule las actividades de prestación de servicio Post-Venta (4.19).

3. Actividades de Soporte

a) Compras

- Se deben *establecer procedimientos* (4.6.1) para asegurar que los productos adquiridos satisfacen los requisitos especificados. Esto se aplica esencialmente a:

  Desarrollo subcontratados de hardware, software, formación...
  Hardware y software estándares.

- La *evaluación* (4.6.2) de subcontratistas deberá incluir:

  Evaluación de los subcontratistas en base a criterios definidos.
  Definición de controles adicionales al subcontratista en función del impacto en la calidad el producto final de la entrega de aquél.
  Establecer y mantener registros de subcontratistas (*Calidad Concertada*, con procedimientos ISO de ingreso, permanencia y eliminación de proveedores de la BB.DD.).

- Los *documentos de compra* (4.6.3) deben contener una clara descripción del equipo comprado, incluyendo:

  Tipo, clase e identificadores adicionales necesarios.
  Identificación de datos técnicos relevantes.
  El identificador del Sistema de Calidad que se le aplicará.

- El producto adquirido deberá ser *verificado* (4.6.4) previamente a su aceptación:

  Si se estableciera una verificación en la sede del proveedor (4.6.4.1) deberá especificarse el modo de actuación para dicha comprobación (qué se inspeccionará y cómo).
  Si se estableciera la necesidad de una verificación por parte del cliente final en las premisas del subcontratista (4.6.4.2), este punto deberá recogerse en el contrato/pedido.

b) Control de Productos Proporcionados por el Cliente

- *Definición de un procedimiento* (4.7) para el control de la verificación, almacenamiento y mantenimiento de los componentes que puedan ser proporcionados por el cliente, para su inclusión en el producto en desarrollo.

[15] Otra cosa es que las empresas lo hagan casi siempre por cuestiones de imagen, y que sea irremediablemente un *on going improvement*.

c) Identificación y Trazabilidad del Producto[16]

- En los casos en que sea necesario, se establecerá el procedimiento para *identificar* el producto a lo largo del ciclo de desarrollo (4.8).

d) Inspección y Pruebas

- Se deben *establecer procedimientos* para las actividades de inspección y pruebas (4.10.1) para verificar que los requisitos de los proyectos son satisfechos.
- Las actividades de inspección deben abarcar a los *elementos recibidos de terceros* (4.10.1), con los criterios:

  Los productos externos no deben ser incluidos en el desarrollo hasta que hayan sido *inspeccionados* de acuerdo con las exigencias del plan de calidad (4.10.2.1).
  La *profundidad de las inspecciones* dependerá de las evidencias exigidas al suministrador (4.10.2.2).
  Si la urgencia no permitiese dicha inspección, debe *registrarse e identificarse* la situación (4.10.2.3).
- Durante el desarrollo, se llevarán a cabo las inspecciones y pruebas definidas en el Plan de Calidad. Los productos deberán ser retenidos hasta que la totalidad de las actividades de inspección y pruebas hayan sido *finalizadas*, a menos que conscientemente, exista una solicitud formal del cliente (4.10.3).
- Debe realizarse un *conjunto de pruebas finales* para validar que el sistema es conforme con los requisitos definidos (4.10.4).
- Deben establecerse mecanismos para el *registro de las evidencias de realización*, con éxito, de las actividades de inspección y prueba (4.10.6).

e) Control del Equipamiento para la Inspección, Medida y Pruebas

- Debe *definirse un procedimiento* para controlar, calibrar y mantener los equipos de medida y pruebas, asegurando que la operativa identifica las tolerancias de medida (4.11.1).
- El *procedimiento de control* (4.11.2) indicará:

  Medidas que se tomarán y precisión de las mismas, para la selección de los equipos apropiados.
  Identificar los equipos de inspección y medida que pueden afectar a la calidad del producto final, calibrándolos y ajustándolos de la manera apropiada.
- Definir el proceso de calibración.
- Identificar los equipos de medida e inspección con un registro de calibración.
- Mantener los registros de calibración (son registros de calidad).
- Evaluar y documentar las medidas derivadas de que el equipo estuviera fuera de calibración.

[16] Trazado a través de la gestión de la configuración del software.

- Asegurar que las condiciones ambientales no perjudiquen las mediciones.
- Asegurar la operación y almacenamiento del equipo para garantizar sus prestaciones.
- Salvaguardar el entorno de pruebas y medidas de ajustes que invaliden la calibración.

f) Formación

- Debe definirse un *procedimiento* (4.18) para identificar las necesidades de formación y su implantación posterior.
- Se mantendrá un resigro de formación del personal. Es conveniente (no obligatorio) que se evalúe la satisfacción con cada una de las sesiones de formación llevadas a cabo.
- Tampoco es obligatorio que se hagan planes de formación (periódicos), aunque sí es recomendable.

g) Técnicas Estadísticas

- Debe identificarse la *necesidad de técnicas estadísticas* para establecer, controlar y verificar el proceso y los productos (4.20.1).
- Debe *definirse un procedimiento* para implantar y controlar la aplicación de las técnicas estadísticas definidas anteriormente (4.20.2).

ISO 9000:94 vs. ISO 9000:2000

Las normas ISO se revisan cada 5 años. Todo lo visto hasta ahora concierne a la norma ISO 9000 del 94. Actualmente ya se ha revisado y está vigente la norma del 2000 (aprobada el 15-12-2002).

Esta nueva revisión no tiene (aún) guía para la aplicación al proceso software, motivo por el cual estudiamos la anterior. Además, los certificados emitidos según la norma UNE EN ISO 9001,2,3:94 tienen todavía una validez máxima de tres años desde la aprobación de la norma UNE EN ISO 9001:2000 y AENOR admitirá solicitudes de certificación según la versión del 94 hasta el 15-12-2002.

Hay cambios muy importantes en la nueva versión:

√ Enfatiza la necesidad de realizar un seguimiento de la satisfacción de los clientes (encuestas escritas, telefónicas,... es decir, ya exige, obliga a proveer un Servicio Post-Venta).

√ Intenta facilitar su aplicación a una empresa.

√ Pretende promover la utilización de los principios de la gestión de la calidad (gestión por procesos y enfoque a cliente fundamentalmente, esto es, obliga a elaborar el mapa de procesos de la empresa).

√ Reducción de la cantidad de documentación requerida.

√ Aplicabilidad de todos los requisitos a cualquier empresa (ya no hay 9001,2,3). Las exclusiones sólo podrán referirse al apartado 7 de la

norma ("Realización del producto") y deberán ser definidas y justificadas.

√ Nuevos requisitos y otros que se amplían.

√ No permite publicidad engañosa.

1.3. Modelo a seguir en un Proyecto de Mejora

Las normas ISO indican lo que hay que hacer, pero no cómo hacerlo. Veremos por último, pues, un listado de los pasos recomendados a considerar:

- ▷ Alguien de Dirección debería encabezar el proyecto.
- ▷ Diagnóstico inicial: ¿cómo está la empresa y a dónde quiere llegar?
- ▷ Inicio de un Modelo de Actuación: ¿cómo se empieza?
- ▷ Manual de Calidad: ¿qué entiende la empresa por calidad?
- ▷ Procesos: ¿cómo desarrolla la empresa las actividades?
- ▷ Documentar el Sistema de Calidad: describir lo que se hace.
- ▷ Implantar el Sistema de Calidad: comenzar a aplicar paulatinamente las directrices establecidas en el Sistema de Calidad elaborado.
- ▷ Planes de Calidad: planificar lo que se hace en cada proyecto nuevo.
- ▷ Revisión de Pruebas: ¿se ha realizado lo que está establecido en el contrato?
- ▷ Registros de Calidad: se reflejará todo lo que se haya hecho.
- ▷ Revisión de la Dirección: ¿el Sistema de Calidad es útil?
- ▷ Mejora: ¿cómo se mejora la operativa de la empresa?
- ▷ Certificación: obtener el reconocimiento público e independiente del trabajo realizado.

El principio de calidad (origen del sistema) en ISO 9000 pasa por redactar el manual de calidad. Los mecanismos que aporta ISO 9000 para la mejora continua son las actividades correctivas/preventivas, las auditorías internas/externas, las técnicas estadísticas...

Conclusión:

> *La calidad raramente aparece, y nunca continuamente, por casualidad. Es el resultado de un esfuerzo continuado y planificado de todas las personas implicadas.*

Capítulo 2

Ciclos de vida

En este capítulo hablaremos de **ciclos de vida** y **ciclos de desarrollo**. Los *ciclos de vida* están orientados al producto, incorporan operaciones y mantenimiento. Por su lado, los *ciclos de desarrollo* están orientados al proceso de desarrollo de software, las fases por las que se va a pasar desde que el software nace hasta el final del desarrollo de las pruebas, desde que se inicia el diseño hasta que se entrega el producto al cliente y éste lo acepta. Se puede considerar que:

```
Ciclo de vida = Ciclo de desarrollo + Mantenimiento
```

2.1. ¿Qué es un proyecto software?

Un **proyecto software** es una actividad humana por la que se transforman unos *requisitos* (necesidades) de un cliente/usuario en un producto software *entregable* (que satisface al menos en gran medida las necesidades del usuario), aplicando un *proceso de desarrollo establecido* (Ciclo de Desarrollo, predecible, repetible y no por la misma persona), en un plazo y con un presupuesto preacordados con un riesgo asociado y una calidad determinada[1].

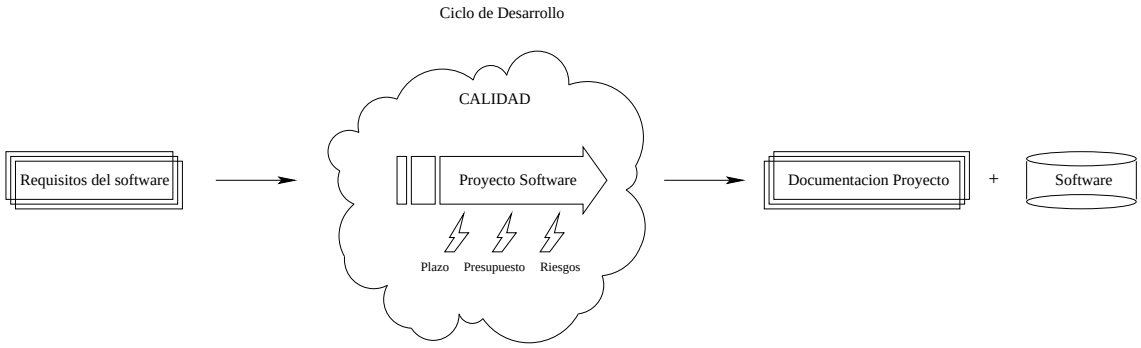


Figura 2.1: Esquema de desarrollo de un proyecto software.

[1]El hecho de que no exista un criterio de aceptación basado en calidad es un gran handicap en el mundo del desarrollo software.

Se entiende por *cliente* la persona o entidad que encarga el producto software, mientras que el *usuario* es quien lo manejará. La transformación de la necesidad a la implementación es equivalente a la composición de tres:

1. $\mathbf{T_1}$, necesidad $\longrightarrow$ modelo conceptual
2. $\mathbf{T_2}$, modelo conceptual $\longrightarrow$ modelo formal
3. $\mathbf{T_3}$, modelo formal $\longrightarrow$ implementación

Este modelo se denomina **modelo esencial del software** y es debido a Bruce D. Blum.

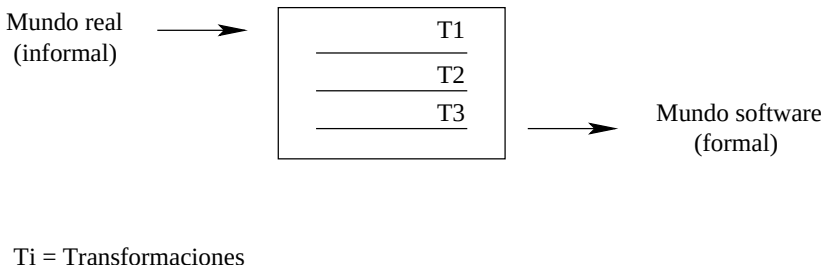


Figura 2.2: Modelo esencial del software.

Normalmente, un proyecto software arranca de una definición del producto a realizar. Consiste en transformar esta definición en un sistema operacional (incluyendo diseños, programación, documentación de usuario,...).

Sin embargo, existe mucha casuística de realización de proyectos software que no se ajusta a lo anterior (proyecto de mantenimiento, de tranformación tecnológica,...).

Pese a todo, los proyectos software no dejan de ser proyectos normales y por ello la mayoría de las técnicas de gestión de proyectos –en general– son aplicables a la gestión de proyectos software.

Las características específicas de los proyectos software son:

Invisibilidad El software es mucho menos tangible que otros productos o sistemas de ingeniería (es más fácil ver el avance en la construcción de un edificio que en un programa).

Las métricas aplicables a otro tipo de "obras" de ingeniería están siendo "inventadas" y normalizadas actualmente para los proyectos software (LOC[2], puntos de función[3], etc.).

[2]Parámetro que como métrica es muy relativo, porque depende del lenguaje, el paradigma de programación,...

[3]Miden la funcionalidad que se le presenta al usuario.

La invisibilidad también afecta a la dificultad de estimación o ingeniería de valoración.

Complejidad El software posee mayor complejidad que cualquier otro producto de ingeniería. Esto se pone de manifiesto cuando se trata de probar el software. Una prueba al 100 % de un sistema software es inviable.

Cualquier otro sistema de ingeniería siempre se prueba al 100 % en cada fase.

Flexibilidad El software es más "fácil de cambiar" y los requisitos del software son, asimismo, cambiantes.

Las especificaciones de un edificio no cambian una vez existen planos, ni se toma a la ligera una remodelación después de acabada su construcción.

Según un estudio de Gartner Group, el 95 % de todos los recursos de una empresa se derivan a mantenimiento y sólo el 5 % al desarrollo.

Juventud de la Ingeniería Muchas veces se dice que "el desarrollo de software es una ingeniería reciente y adolece de procesos y estrategias de desarrollo variadas, poco probadas, definidas y, lo más negativo, puestas en práctica", lo cual no es cierto.

Existen muchas maneras (*Ciclos de Desarrollo*) de producir software. En este capítulo repasaremos las más empleadas:

- En cascada
- En V
- Prototipado
- DRA
- Incremental
- Espiral

2.2. Contexto de un proyecto software

Un proyecto software es inevitablemente desarrollado en un determinado entorno, contexto, dominio; no es una actividad aislada e independiente de todos los demás procesos de una organización, ni mucho menos.

2.3. Consideraciones sobre los Ciclos de Vida

2.3.1. Necesidad

Para resolver los problemas reales del software, se debe incorporar una estrategia de desarrollo que defina el proceso acompañándose de métodos y herramientas.

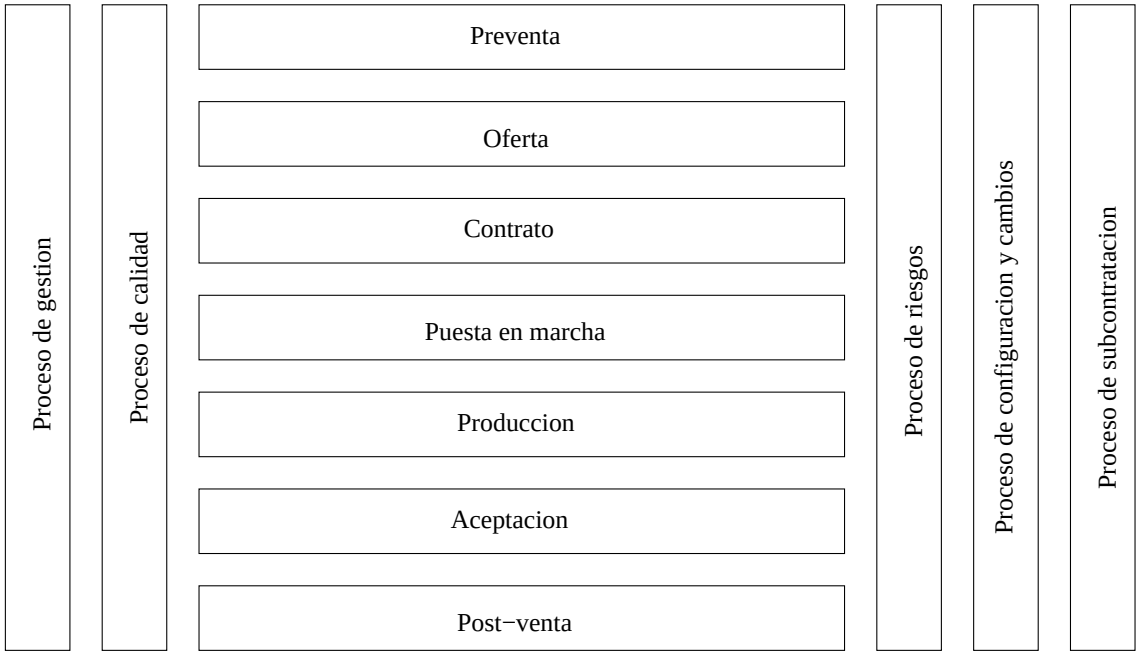


Figura 2.3: Contexto de un proyecto sofware.

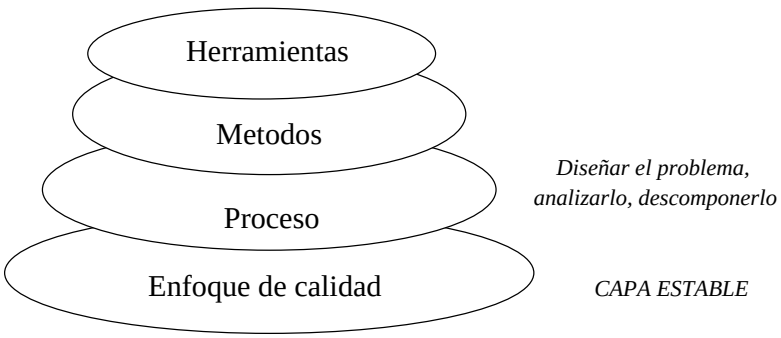


Figura 2.4: Capas de Ingeniería del Software.

Cualquier enfoque de ingeniería debe descansar sobre un empeño de enfoque de calidad. Los cimientos de la Ingeniería del Software están orientados a la calidad.

El fundamento principal de la Ingeniería del Software es la capa de proceso (ver figura 2.4, página 34): unión que mantiene la tecnología.

Asimismo, los métodos seguidos nos indican cómo construir técnicamente el software (estructurado, orientado a objetos,...) utilizando las herramientas pertinentes.

El Ciclo de Vida que se va a aplicar se selecciona, al igual que los métodos y las herramientas, según el proyecto a desarrollar. Se debe:

1. Definir las actividades (análisis, diseño, implementación y pruebas) a realizar al desarrollar el proyecto.
2. Introducir consistencia entre los diferentes proyectos de desarrollo en la misma empresa.
3. Proporcionar *checkpoints* (hitos) para gestionar las decisiones de seguir o no, y de seguimiento.

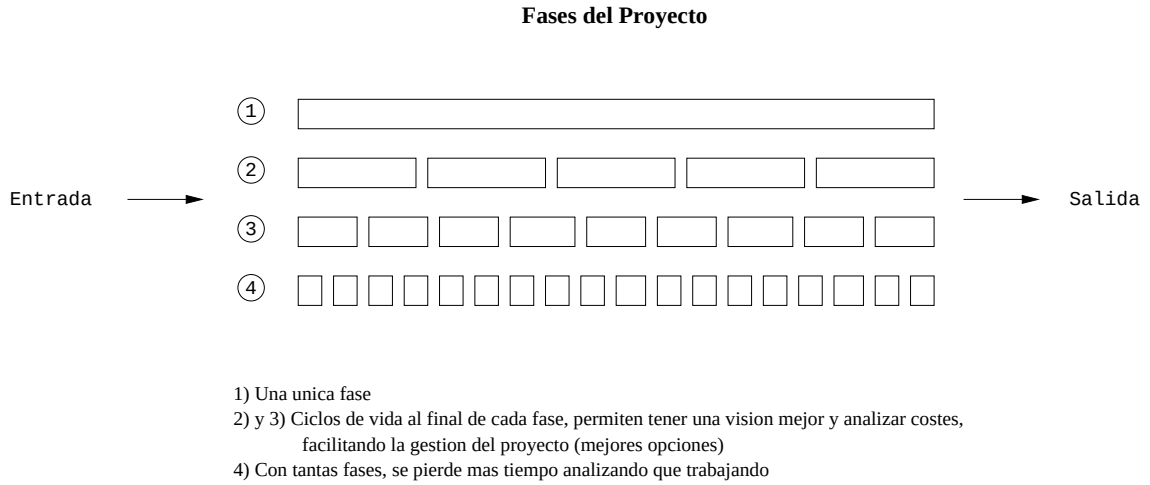


Figura 2.5: División de un proyecto en fases.

2.3.2. Diferentes nombres

Lo que llamamos *Ciclo de Desarrollo* es lo mismo que *Modelo de Producción* o *Modelo de Proceso*, todos estos términos representan la misma idea.

```
Development Cycle = Development Model = Process Model
```

Uno de los aspectos estratégicos determinantes del éxito de un proyecto software es la elección adecuada del Ciclo de Desarrollo para llevarlo a cabo.

El Ciclo de Desarrollo no es más que una formalización del propio proceso de desarrollo. Es un marco de referencia para el proceso que se inicia con la decisión de desarrollar un producto software y finaliza cuando se ha entregado. Incluye en general:

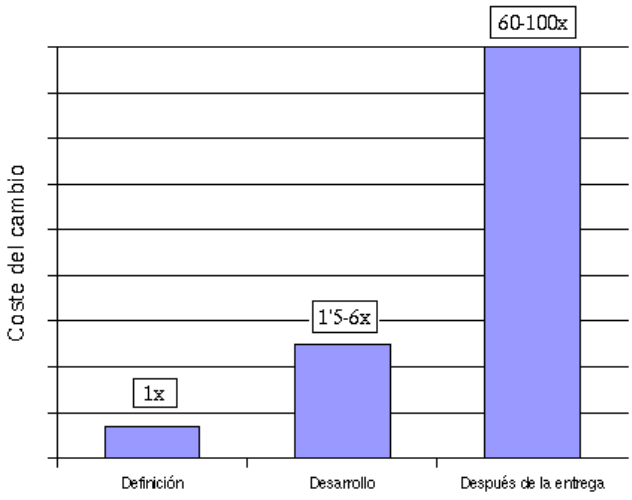
- √ Fase de requisitos[4]
- √ Fase de diseño
- √ Fase de implementación
- √ Fase de prueba y, en ocasiones,
- √ Fase de instalación, aceptación y mantenimiento[5]

No debemos confundir los Ciclos de Vida del Software (que indican qué hay que hacer) con las Metodologías de Desarrollo de Software (que indican cómo hay que hacerlo).

Por último, comentaremos un aspecto a destacar en los modelos más avanzados, como es la inclusión en los Ciclos de Vida de actividades de *Verificación* y *Validación* (*V&V*), como por ejemplo:

- ○ Revisiones
- ○ Pruebas unitarias, de integración, de sistema, de aceptación
- ○ Walkthroughs

Con estas actividades se pretende la detección temprana de defectos para evitar costes mayores:



La **verificación** consiste en comprobar que el producto se está realizando correctamente, y la **validación** en comprobar que estamos realizando el producto correcto.

[4]Esta fase consta, según Alan Davis —una de las personalidades de la IS— de: elaboración de la Especificación de Requisitos Software (ERS) y análisis del problema (modelos de análisis, DFD's, ER,...), que en IC se denomina *conceptualización*.

[5]Incluir esta fase hace del Ciclo de Desarrollo un Ciclo de Vida.

2.4. Diferentes Ciclos de Vida

2.4.1. Ciclo de Vida de Codificación Directa

El Ciclo de Vida denominado de **Codificación Directa** (también denominado *Code and Fix*) suele ser usado por empresas más inmaduras (de menor nivel CMM), a la par que es el de mayor coste asociado por los inconvenientes que arrastra.

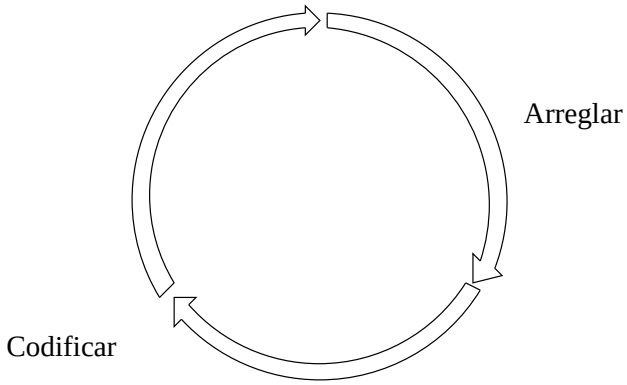


Figura 2.6: Esquema del Ciclo de Vida de Codificación Directa.

Se puede considerar sinónimo de "arte" en el desarrollo software, al tiempo que antónimo de "ingeniería". Es *en absoluto* recomendable para proyectos un poco más grandes que "enanos".

En los siguientes ciclos de vida son típicas tres fases genéricas:

Definición (Pensar) Se centra en el qué: qué información debe ser procesada, qué rendimiento se espera, qué criterios de validación se usarán, qué datos, qué procesos... para tratar de identificar requisitos y aspectos clave.

Desarrollo (Hacer) Se centra en el cómo: cómo es la comunicación entre módulos, detalles procedimentales, cómo se hacen las pruebas, cómo se codifica...

Mantenimiento (Mantener) Se centra en el cambio: corrección de errores, mejoras por requisitos cambiantes,... Es prácticamente como volver a aplicar el ciclo de vida pero no desde cero.

Y, además, existen los siguientes *tipos* de mantenimiento:

Correctivo Modifica el software para corregir errores no detectados.

Adaptativo Adapta el software al entorno cambiante, ya sean debidos a cambios técnicos (por ejemplo, cambios en el SO) o cambios de forma (por ejemplo un cambio en el IVA).

Perfectivo Intenta ampliar los requisitos.

Preventivo Se aplica, por ejemplo, a cosas que cambian con el tiempo, se toma lo que hay y se adapta.

2.4.2. Ciclo de Vida en Cascada

Es el ciclo de vida más tradicional, más conocido y más aplicado, siendo conocido también con el nombre de *WaterFall*. Es el ciclo que utilizan un gran número de metodologías, entre ellas por ejemplo Métrica (la metodología de la administración pública española).

Fue concebida durante los años 70 como solución a los problemas del *Code and Fix*, que se habían puesto de manifiesto con la crisis del software.

Hace especial énfasis en la realización temprana de actividades de definición (de requisitos) y documentación (análisis y diseño), como paso previo a la codificación, intentando asegurar una documentación explícita de todas las fases o subproductos.

Las fases (análisis, diseño, codificación, pruebas y puesta en producción) se encadenan linealmente, sin comenzar una hasta que se termina la anterior, pasando siempre a la siguiente en la cascada utilizando en ella los productos que se obtuvieron en la previa.

El modelo original (figura 2.9), que se debe a Winston Royce (1970), hacía provisiones para bucles de realimentación, pero la mayoría de las organizaciones lo aplicaron como si fuera estrictamente lineal (figura 2.8).

Cuando mencionamos *ingeniería de sistemas* nos referimos a establecer los requisitos de todos los elementos (o al menos los más importantes) del sistema y sus relaciones. El *análisis de requisitos* es el estudio de los requisitos propios del software a desarrollar (enumeración de necesidades, de las que el cliente debe recibir una copia).

Una vez identificados, se acomete el *diseño*, que se basará en datos o estructuras de datos, definirá una arquitectura de datos (conexión entre módulos), representará una interfaz (el usuario o/y otros sistemas) y establecera una serie de algoritmos para resolver problemas. El modelo surgido del diseño se documenta (ER, UML, DFDs...) perfectamente antes de abordar la fase de *codificación*, que será seguida de las *pruebas*.

El mantenimiento sirve para aplicar los cambios a las fases[6].

Características principales

- Consta de una serie de fases sucesivas.
- Cada fase se define como un grupo coherente de actividades de producción y control: sólo se prueba el código al final, pero podemos tener actividades de verificación y validación (V&V, aseguramiento de la calidad) en cada fase (auditorías internas al final de cada fase, por ejemplo, verificación de diagramas UML).

[6] En ISO 9000, una manera de reflejar el mantenimiento es mediante un registro de cambios.

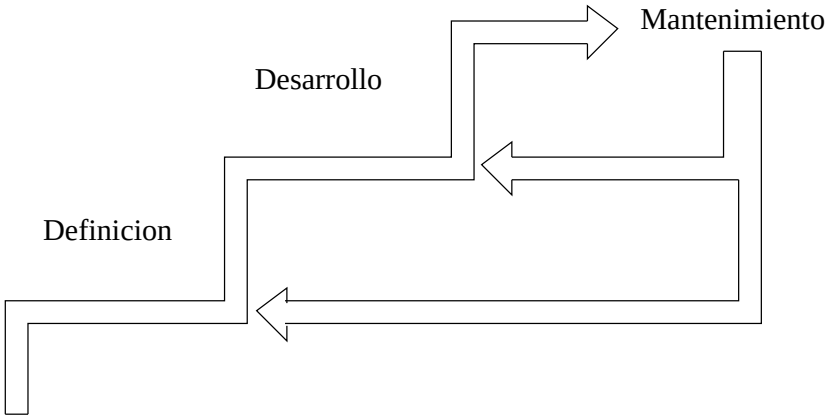


Figura 2.7: Esquema general de un Ciclo de Vida.

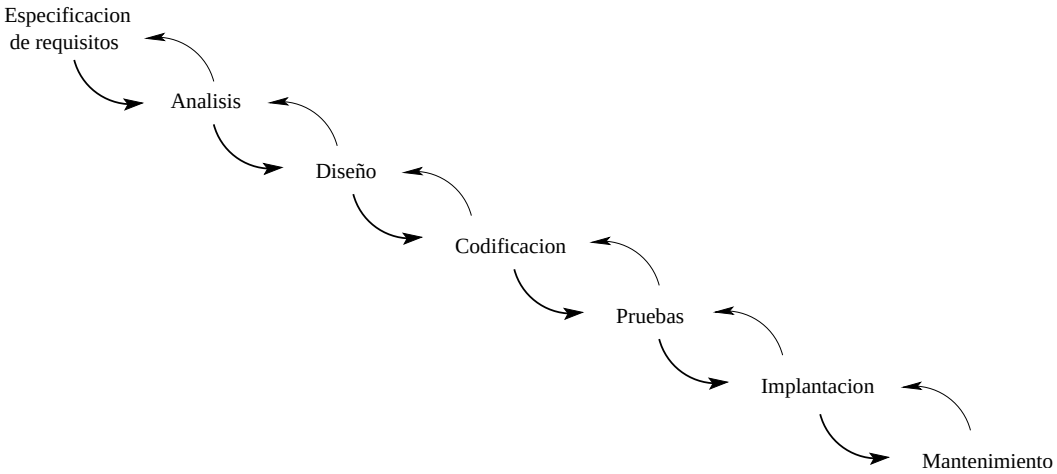


Figura 2.8: Esquema del Ciclo de Vida en Cascada.

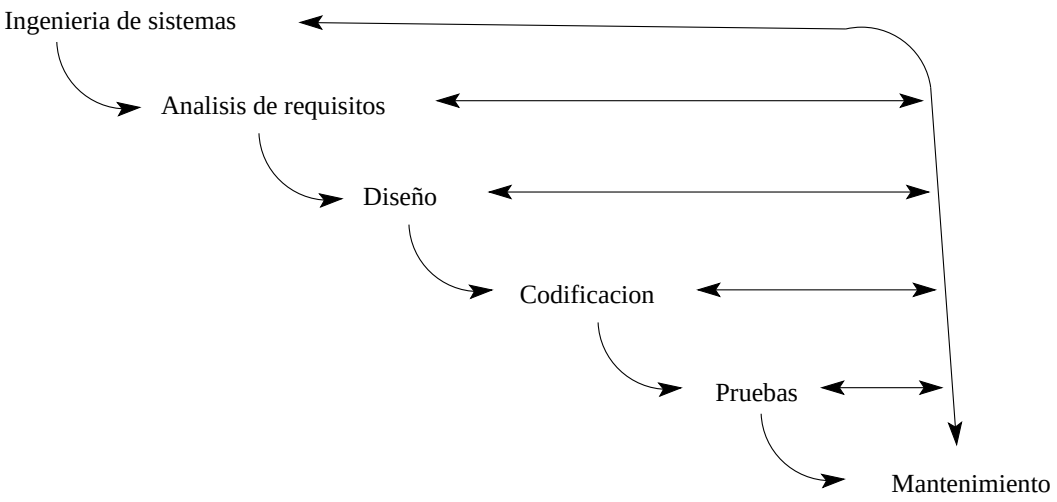


Figura 2.9: Esquema (modificado) del Ciclo de Vida en Cascada.

- Cada fase se caracteriza por una actividad dominante que le da nombre y determina:
  - el tipo,
  - el contenido y
  - el resultado de las actividades

Ventajas

- Es un método estructurado para el desarrollo de software.
- Marca pautas de trabajo muy claras, facilitando la coordinación.
- Facilita la disposición de hitos en el desarrollo del proyecto.
- Facilita la estimación y el seguimiento del progreso de las actividades y, por ende,
- Facilita la detección de desviaciones (no tanto en cuanto a la emisión sino en cuanto a la temporalidad) y ls realización de acciones correctoras.
- Proporciona productos entregables intermedios que forman el conjunto final del producto.

Inconvenientes

- No soporta prácticas modernas de desarrollo (prototipado).
- Posee gran rigidez: cada actividad es prerrequisito de las que le suceden.
- No permite "vuelta atrás". Al ser tan lineal es una utopía.
- Los posibles problemas se detectan tarde aunque se incluyan actividades V&V.
- Los únicos productos parciales aprovechables están en forma de documentos.
- "Nada esta hecho hasta que todo está hecho", por lo que un cambio debido a un error puede suponer un gran coste.

La ventaja principal reside en que suministra un marco de referencia para la asignación de todas las actividades de desarrollo de software, siendo la aproximación más empleada en Ingeniería del Software.

Por el contrario, los inconvenientes principales son el hecho de que comienza por establecer todos los requisitos del sistema, algo que muchas veces no es posible en un primer momento, porque puede ser difícil hasta para el propio usuario, y otras veces porque surgen cambios de parecer de los usuarios sobre las necesidades reales cuando la fase de especificación de requisitos ya se ha acabado.

Debido a la aceptación que tuvo, hay muchas variantes de este ciclo de vida, pero aún así la filosofía es siempre la misma.

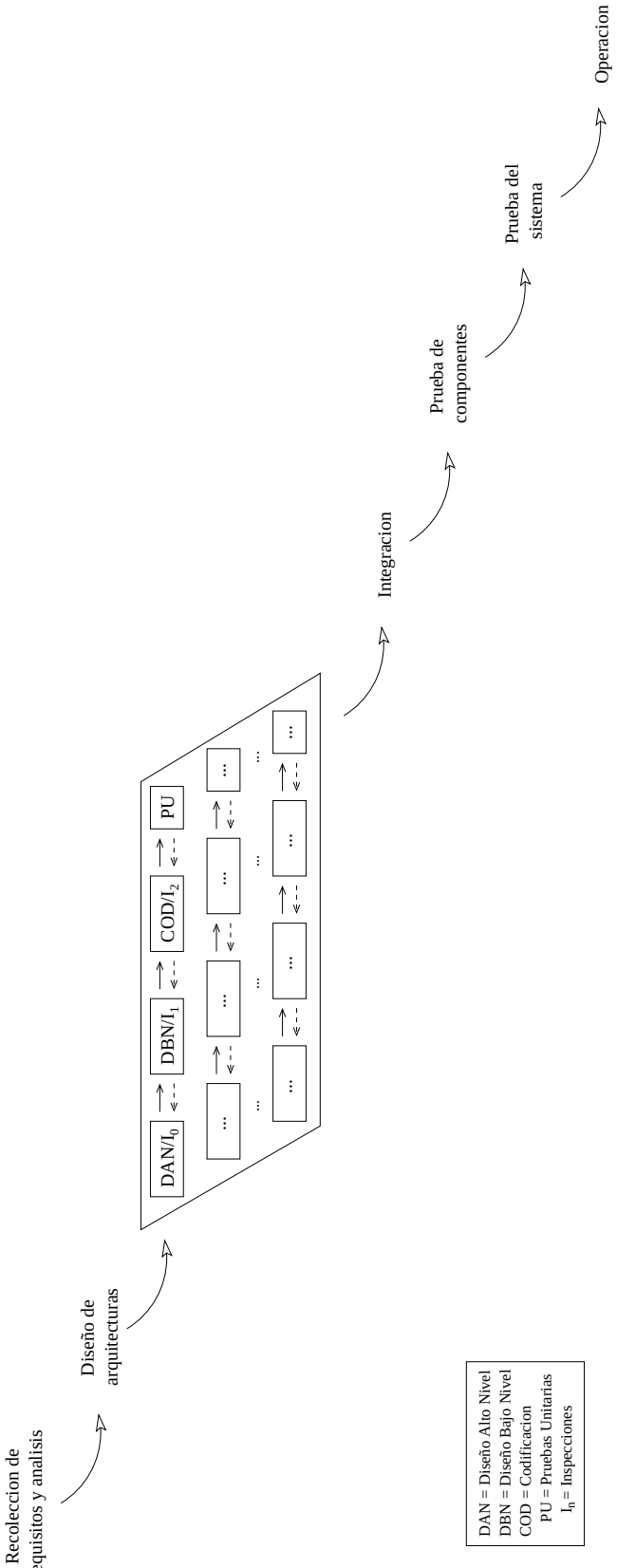
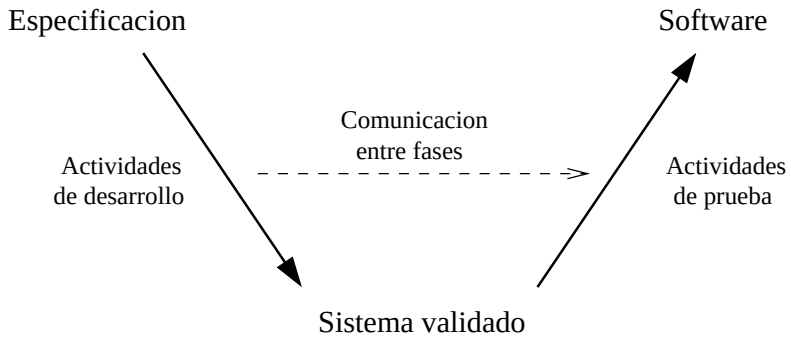


Figura 2.10: Versión actual del esquema del Ciclo de Vida en Cascada.

2.4.3. Ciclo de Vida en *V*

Similar al Ciclo de Vida en Cascada, el Ciclo de Vida en V tiene dos tiempos claramente marcados que no existen sin embargo en el modelo en cascada:

- Actividades de desarrollo (rama descendente)
- Actividades de prueba (rama ascendente)



La comunicación entre fases al mismo nivel representa la idea de que cada componente que se desarrolla, se prueba. En este punto reside la ventaja que presenta este ciclo: a la vez que se desarrolla, de algún modo se preparan las pruebas (sobre cada elemento), es decir, elaborar el plan de pruebas es parte del desarrollo.

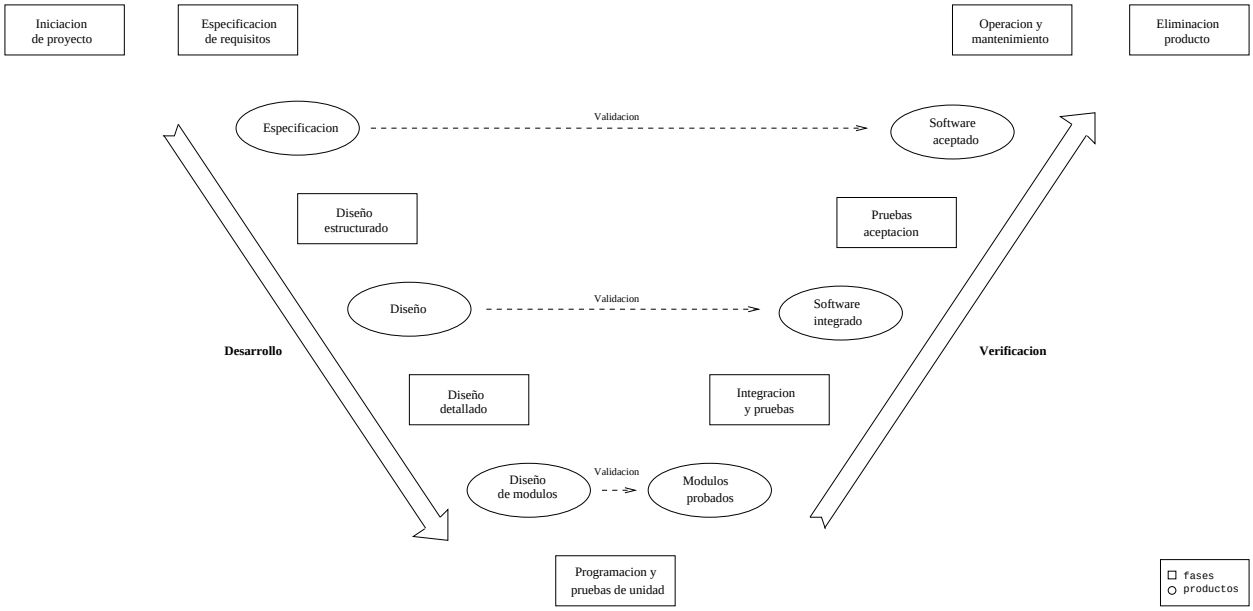


Figura 2.11: Esquema del Ciclo de Vida en V.

Quizás este ciclo de vida sea el más utilizado después del cascada.

Rama descendente

En cada fase se llevan a cabo las actividades propias definidas para dicha fase, se tiene en cuenta y se *verifican* las salidas de la fase previa, sirviendo además de referencia para las actividades de la fase siguiente a través de los productos generados en ella. Por último, prepara las pruebas que permitan validar lo que ha sido definido durante la fase, pruebas que se ejecutarán en la fase situada al mismo nivel en la rama ascendente de la "V".

Rama ascendente

En cada fase se ejecutan las pruebas para validar lo definido durante la fase de su mismo nivel de la rama descendente, se verifican las actividades llevadas a cabo durante la fase anterior y sirve de referencia para las actividades de la próxima fase.

Ventajas

Son similares a las del Ciclo de Vida en Cascada:

- Los subprogramas de las diferentes fases permiten materializar el progreso del proyecto y asegurar si los objetivos de la fase se han cumplido o no.
- Favorece la consideración de las pruebas lo antes posible (comunicación horizontal en la "V").

Inconvenientes

- No soporta prácticas modernas de desarrollo, como por ejemplo el prototipado.
- Posee gran rigidez (aunque es menor que la del modelo en cascada debido a la comunicación horizontal que se introduce).
- No permite "vuelta atrás". Al ser tan lineal es una utopía.
- Los únicos productos parciales aprovechables están en forma de documentos.
- De nuevo, "nada está hecho hasta que todo está hecho", por lo que un cambio debido a un error puede suponer un gran coste.

La ventaja principal reside en que se suministra un marco de referencia para la asignación de todas las actividades de desarrollo de software, incluyendo actividades de verificación y validación (V&V) de lo que se hace en las sucesivas etapas.

El inconveniente más importante, por su parte, es el comenzar por establecer todos los requisitos del sistema (igual que en el modelo en cascada), lo que muchas veces no es posible desde el primer momento porque puede ser difícil para el propio usuario o bien porque en ocasiones hay cambios de parecer en éstos sobre sus necesidades reales, cuando dicha fase de especificación de requisitos ya ha sido terminada.

Una versión adaptada a una empresa podría ser la de la figura 2.12:

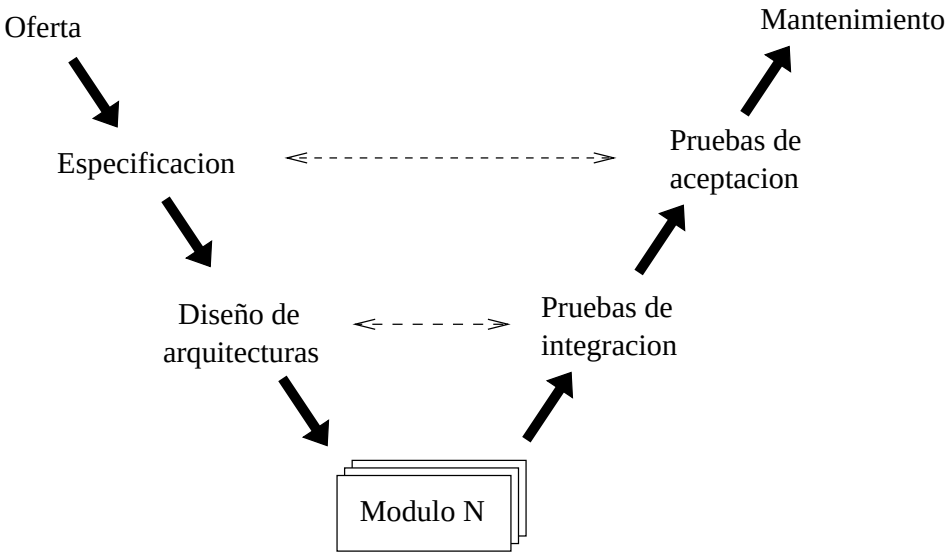


Figura 2.12: Esquema adaptado del Ciclo de Vida en V.

Especificación: A esta fase entra la oferta elaborada y requisitos, y sale el plan de pruebas de aceptación y el plan de proyecto.

Diseño de arquitecturas: A esta fase entra el documento de especificación de requisitos y sale el documento de arquitectura y el plan de pruebas de integración.

Codificación: A esta fase entra el documento de arquitectura y sale un documento en detalle (codificación), que es la entrada al módulo i donde se elabora el plan de pruebas unitarias y se lleva a cabo.

Pruebas de integración: A esta fase entra el plan de pruebas unitarias y el plan de pruebas de integración (resultados del módulo i y de la comunicación horizontal, respectivamente) y se lleva a cabo.

Pruebas de aceptación: A esta fase entra el plan de pruebas de aceptación y el resultado de las pruebas de integración, y se llevan a cabo las primeras, obteniéndose como resultado el producto final.

2.4.4. Ciclo de Vida Prototipado

El Ciclo de Vida Prototipado añade una fase de construcción de un prototipo (que no tiene por qué ser una aplicación, puede estar en papel, ser una presentación, una aplicación ya existente que se modifica, etc) que ayuda a capturar los requisitos y pretende crear un modelo del software a desarrollar.

El *prototipo* es, como decimos, un producto en papel, software o cualquier soporte que permita saber qué quiere el usuario (aspecto exterior, dinámica de pantallas,...) . Debe cumplir dos requisitos:

1. Ser *funcional*: debe poseer una pequeña funcionalidad, implementando pequeños detalles para que salgan a la luz requisitos.
2. Ser *interactivo*: presentar una aplicación existente para que el usuario/cliente indique lo que quiere o no.

Esta estrategia es adecuada cuando no se sabe exactamente lo que se quiere construir por alguna de las siguientes razones:

- $\surd$ el cliente/usuario no sabe exactamente lo que quiere
- $\surd$ el desarrollador no está seguro de la eficiencia de una aproximación
- $\surd$ existen dudas sobre la interfaz hombre-máquina

Todos los ciclos de vida parten de la captación de requisitos. La diferencia que marca éste es que el prototipado se utiliza en conjunción con cualquier otro ciclo de vida.

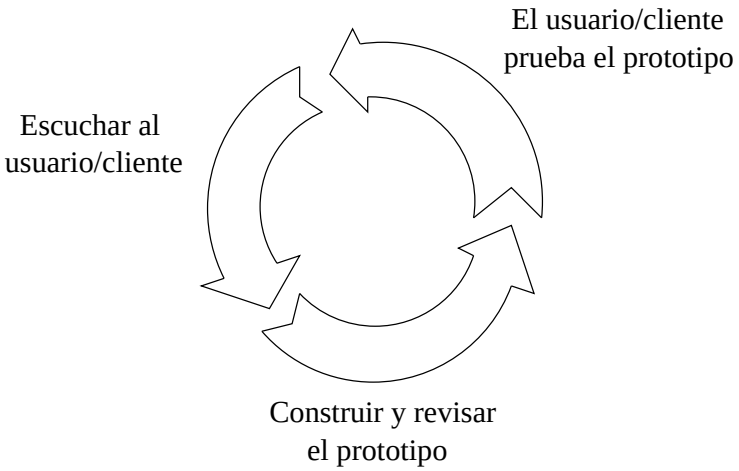


Figura 2.13: Esquema del Ciclo de Vida Prototipado.

Fases

Escuchar al usuario/cliente.-

↪ se corresponde con la recolección de requisitos

↪ se definen los objetivos globales del sistema (sus límites)

↪ se identifican los requisitos conocidos y las áreas donde se precisa una mayor definición

Construir y revisar el prototipo.-

↪ se realiza un desarrollo rápido centrado en los aspectos del software visibles para el usuario/cliente (formatos de entrada, de salida, pantallas...)

↪ este desarrollo rápido conforma el prototipo elaborado en este ciclo (cada ciclo $\rightsquigarrow$ nuevo prototipo, se necesitarán varios ciclos para identificar todos los requisitos)

El usuario/cliente prueba el prototipo.-

↪ el prototipo es evaluado por el cliente/usuario

↪ esta evaluación permite refinar los requisitos del software

↪ este refinamiento es la entrada al siguiente ciclo del modelo de prototipado (un bucle entero del Ciclo de Vida Prototipado origina un prototipo que será la entrada al siguiente ciclo)

¿Qué se hace con el prototipo? En la mayoría de los casos, éste será demasiado lento, demasiado grande y torpe en su uso. Por lo tanto, idealmente se debe desechar el prototipo y empezar de nuevo. El prototipo sólo debe servir para identificar los requisitos del software cuando éstos no estén claros.

Ventajas

El prototipo es un mecanismo ideal para extraer requisitos cuando éstos no están claros, incluso para el usuario.

Inconvenientes

⋆ Tendencia del usuario/cliente a creer que el trabajo ya está hecho y que en breve dispodrá de un sistema funcional, cuando en realidad se está en la primera fase.

⋆ El desarrollador toma decisiones de implementación simplemente porque son conocidas y le permiten desarrollar rápidamente el prototipo, sin que sean las decisiones más adecuadas para el sistema real. Sin embargo, estas decisiones a menudo se mantienen posteriormente en el sistema real.

2.4.5. Ciclo de Vida DRA

El Ciclo de Vida DRA (*Desarrollo Rápido de Aplicaciones*, *Rapid Application Development* o RAD en inglés) es un modelo lineal secuencial que enfatiza un ciclo de desarrollo extremadamente corto. Es una adaptación a "alta velocidad" del modelo en cascada.

Se logra el desarrollo rápido utilizando un enfoque de construcción basado en componentes. Si los requisitos están claros y se limita el ámbito del proyecto[7], este ciclo es muy adecuado y permite crear un sistema en poco tiempo (de 60 a 90 días).

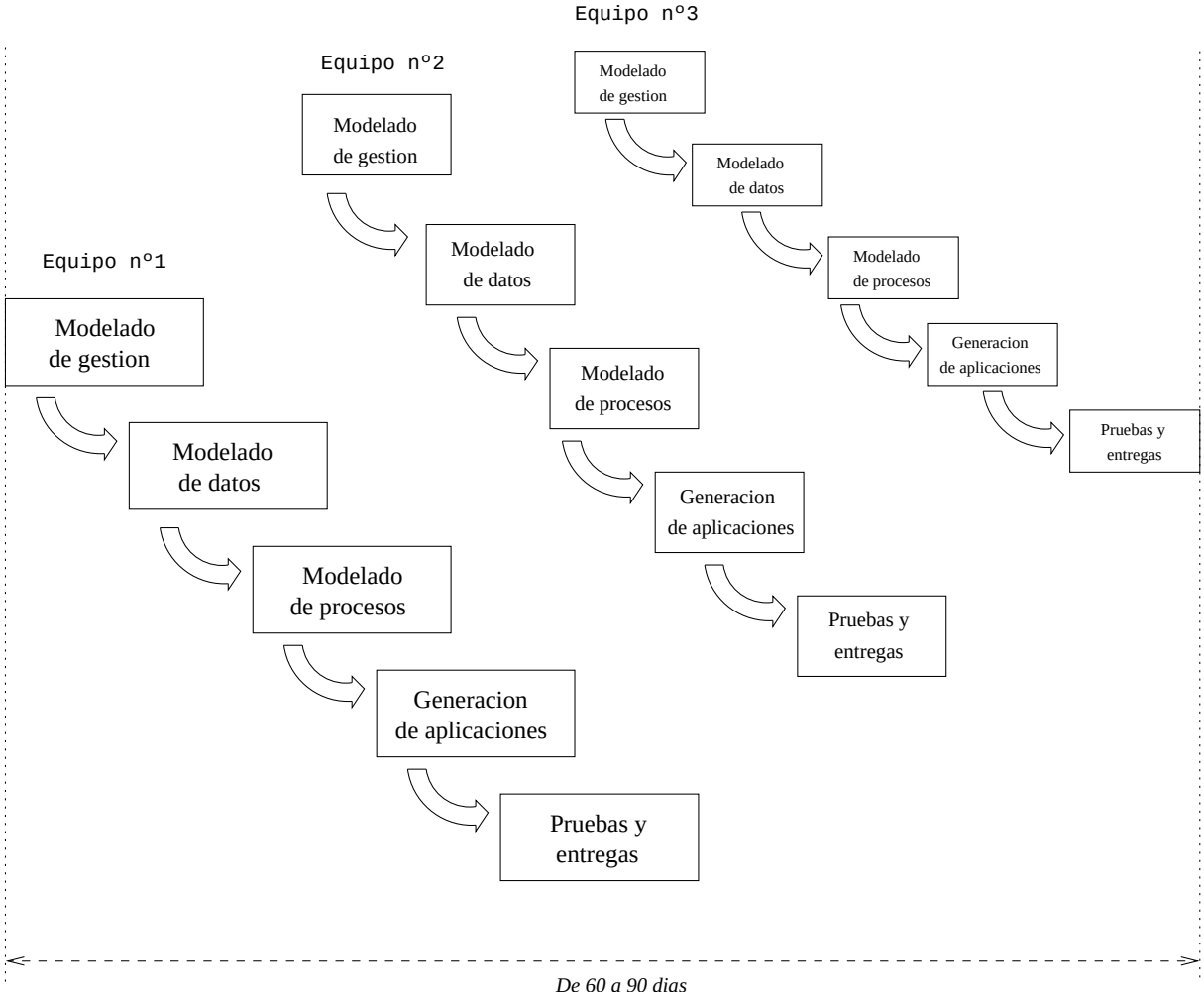


Figura 2.14: Esquema del Ciclo de Vida DRA.

[7]De lo contrario es preferible seguir un Ciclo de vida de Prototipado a fin de conseguir un modelo rápido.

Fases

Modelado de gestión: El flujo de información entre las funciones de gestión se modela de forma que responda a:

- ¿Qué información conduce el proceso de gestión?
- ¿Qué información se genera?
- ¿Quién la genera?
- ¿A dónde va la información?
- ¿Quién la procesa?

Modelado de datos: El flujo de información definido como parte de la fase anterior se refina como un conjunto de objetos de datos. Se definen los atributos de cada objeto y las relaciones entre ellos.

Modelado de procesos: Los objetos de datos definidos en la parte anterior quedan transformados para lograr el flujo de información necesario para implementar una función de gestión. Las descripciones del proceso se crean para manipular los objetos de datos.

Generación de aplicaciones: El DRA asume el empleo de técnicas de cuarta generación (4GL). En vez de emplear el 3GL, el proceso DRA trabaja para:

- volver a utilizar componentes de programas ya existentes (cuando sea posible), o
- crear componentes reutilizables (cuando sea necesario)

Siempre se emplean herramientas automáticas para facilitar la construcción del software.

Pruebas y entregas: Como el DRA enfatiza la reutilización, muchos de los componentes ya están probados (en la mayoría de los casos no hace falta realizar pruebas unitarias), lo que reduce el tiempo de pruebas, limitándolo a el necesario para probar los nuevos componentes y ejercitar todas las interfaces a fondo.

Restricciones

Las limitaciones de tiempo que impone el modelo demandan ámbito de escalas. Si una aplicación se puede modular de forma que cada función principal se puede completar en menos de 3 meses, con el enfoque anterior, el candidato es el DRA:

- cada una de las funciones se asigna a un equipo DRA diferente
- se integrarán en las actividades finales

Ventajas

√ Introduce reutilización (mayor profundidad y menor probabilidad de errores)

√ Ideal para la tecnología O.O. (componentes)

√ Inherentemente introduce el paralelismo

Inconvenientes

▷ Para proyectos grandes, el DRA requiere recursos humanos suficientes para crear los equipos DRA correctos

▷ Se requiere compromiso (de clientes y desarrolladores) en las rápidas actividades necesarias para acabar el sistema

▷ Si un sistema no se puede modularizar adecuadamente (módulos estancos), la construcción de componentes será complicada

▷ No es adecuado si los riesgos son altos (tecnologías nuevas, aplicación compleja o mucha interoperatividad, ya que un problema en un equipo repercutirá en los dependientes, aunque las dependencias sean mínimas)

2.5. Modelos Evolutivos

Hemos visto una serie de ciclos de vida regidos por un denominador común: su linealidad y el producto software como resultado final. Sin embargo, el software, al igual que todos los sistemas complejos, evoluciona con el tiempo. Los requisitos de gestión, del producto, del propio mercado, etc., a menudo hacen que sea imposible finalizar un producto completo.

Esto conlleva la introducción de una versión limitada para cumplir la presión competitiva y de gestión. El problema no está en el conjunto de requisitos principales, que se comprende perfectamente, pero todavía se tienen que definir los detalles de extensiones del producto.

En ésta y otras situaciones semejantes, se necesita un modelo de proceso que se haya diseñado explícitamente para acomodarse a un producto que evolucione con el tiempo, ya que

- el enfoque en cascada asume que al final de la secuencia marcada se entrega un sistema completo
- el enfoque de prototipado ayuda a comprender los requisitos para luego acometer el desarrollo

En general, en la realidad no se desarrolla para entregar un sistema listo para pasar a producción de forma completa. Sin embargo, los paradigmas vistos anteriormente no tienen en cuenta la naturaleza evolutiva del software.

Los **modelos evolutivos** son interactivos. Se caracterizan por permitir desarrollar versiones cada vez más completas del software. Dos ejemplos, que veremos a continuación, son el *Modelo Incremental* y el *Modelo en Espiral*.

2.5.1. Modelo Incremental

Este modelo encarna la filosofía de proceso de construir una implementación parcial del sistema global y posteriormente ir aumentando la funcionalidad del sistema.

Es la aplicación reiterada de varias secuencias basadas en el modelo en cascada. Cada aplicación del ciclo constituye un incremento del software. Cada incremento puede ser:

- √ el refinamiento de un incremento anterior (siguiendo la filosofía de prototipado, el incremento anterior sería el prototipo para el nuevo incremento)
- √ el desarrollo de una nueva funcionalidad no incorporada en incrementos anteriores

En el primer incremento de un sistema se debe abordar el núcleo esencial del sistema (requisitos fundamentales). En incrementos posteriores se abordarán funciones suplementarias (algunas ya conocidas, otras no).

El usuario/cliente evalúa el resultado de un incremento y se elabora un lan para el incremento siguiente. El proceso se repite hasta que se elabore el producto completo, sin que por ello se interfiera en que al final de cada incremento resulta un producto operativo que tiene funcionalidad completa (es un subproducto verificado que el cliente debe validar).

Al seguir un desarrollo incremental, el software debe construirse de tal forma que facilite la incorporación de nuevos requisitos, dotando de este modo al software de mayor flexibilidad. Hay que tener en cuenta que un problema en un incremento puede trasladarse a incrementos posteriores, además de que el desarrollo no tiene por qué ser lineal.

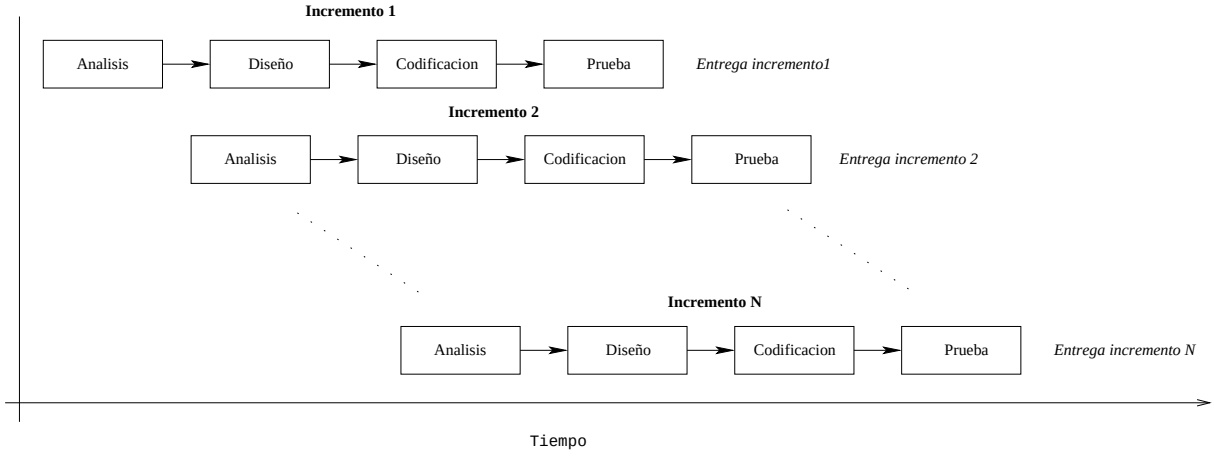


Figura 2.15: Esquema del Modelo Incremental.

Pueden existir problemas de coordinación, pero no hay que entregar un incremento para poder comenzar el siguiente.

Ventajas

- La dotación de personal no tiene que estar disponible para una implementación completa.
- Permite la obtención de incrementos operativos a lo largo del proceso de desarrollo, frente a modelos tradicionales en cascada, esto es, reduce el gasto que se tiene antes de alcanzar cierta capacidad inicial.
- Reduce la posibilidad de que las necesidades del usuario/cliente cambien durante el desarrollo.
- Mayor flexibilidad ante más funcionalidades.

Inconvenientes

- Es complejo definir el núcleo operativo para poder lograr el primer incremento: un software con capacidades suficientes como para ser operativo y que facilite la incorporación de nuevos requisitos.
- Puede resultar complicado establecer y priorizar las funcionalidades que se mejoran o incorporan a los sucesivos incrementos.
- Las soluciones de incremento anteriores pueden no ser válidas para incrementos posteriores.

2.5.2. Modelo en Espiral

Ideado por Barry Boëhm en 1986, el *Modelo en Espiral* es un modelo evolutivo que no está tan sujeto al ciclo de vida en cascada. Gestiona inherentemente prototipado y gestión de riesgos, combinándolos con una filosofía evolutiva, y permite mantener la estructura y sistematización del ciclo de vida en cascada, desarrollando el software de manera incremental.

La idea básica es que cada fase (vuelta de la espiral) establece los siguientes pasos:

1. Determinación de objetivos, alternativas y restricciones
2. Evaluación de alternativas (considerando análisis de riesgos)
3. Desarrollo del siguiente nivel de producto
4. Planificación de la siguiente fase (ciclo en la espiral)

Cada ciclo (vuelta) en la espiral representa una fase del proceso de desarrollo. Por ello, el ciclo más interno se corresponde con la viabilidad del sistema. El siguiente, con el análisis de requisitos, y así sucesivamente.

No hay fases fijas en el modelo, que se adapta a las necesidades que surjan en cada proyecto (podría decirse que es, por tanto, una especie de *metamodelo*).

Ventajas

- Permite adaptar el proceso de desarrollo a las necesidades cambiantes del proyecto y al conocimiento que se va adquiriendo.
- Permite el manejo de prototipos, enlazándolo con el análisis de riesgos.
- Gestiona explícitamente los riesgos, lo que permitirá reducirlos antes de que se conviertan en problemas.

Inconvenientes

- Requiere de una considerable habilidad para la consideración del riesgo. Cuenta con esto para el éxito.
- El modelo es relativamente nuevo y no se ha manejado tanto como los anteriores. Tendrá que ponerse más en práctica en décadas futuras para asegurar su eficacia.

Capítulo 3

Gestión y Planificación de Proyectos

3.1. Introducción

La **Gestión y Planificación** de proyectos es un factor de calidad en el ámbito del software, no en vano la planificación y seguimiento de proyectos son dos áreas clave del nivel 2 CMM, donde se busca sólo la repetitividad.

El problema es que resulta duro aprender sobre temas de calidad trabajando 25 horas al día para sacar adelante los proyectos. Es un círculo vicioso: para planificar necesito parar, pero no puedo parar para acabar.

La solución pasa por controlar los proyectos (planificación) y controlar las planificaciones (seguimiento)[1].

3.1.1. La crisis del software

Durante años el desarrollo del software se ha considerado un *arte* y los Directores (o Jefes de Proyecto) han dirigido más bien bajo consideraciones técnicas que de gestión.

Los problemas originados a raíz de esto son:

- Desviaciones en costes y tiempos inaceptables.
- Incremento en la complejidad de los sistemas.
- Calidad baja o muy baja en sistemas vitales.

La conclusión es que la Ingeniería del Software por sí sola no es la solución si no se considera la Planificación y Gestión de proyectos. Se necesita una metodología para planificar, organizar y controlar.

3.1.2. Claves del éxito

Dirigir un proyecto software requiere:

[1] Ojo, gestión de proyectos ⇒ planificación de proyectos, pero no viceversa.

- √ Organizar profesionales en buenos y equilibrados equipos.
- √ Procesos de ingeniería de software y de gestión bien hechos.
- √ Metodologías, técnicas y herramientas adecuadas y conjuntadas.
- √ Buena especificación de requisitos (por parte del usuario).
- √ Planificación adecuada.
- √ Presupuestos correctos, ajustados y proporcionados.
- √ Calendarios realistas (se admite presión aceptable).
- √ Control y coordinación (actividades de control de calidad y seguimiento del proyecto).
- √ Buena comunicación e información (entre los participantes del proyecto).
- √ Personal formado, motivado y adecuado.
- √ Gestión de riesgos.

3.2. CMM y Gestión de Proyectos

Si no hay *procesos repetibles* que emplee toda la organización, no existe una base para la *mejora continua* tanto de la productividad como de la calidad.

Organización inmadura (nivel 0 CMM)

- ◦ No hay un proceso definido y riguroso.
- ◦ No hay estimaciones realistas, y por tanto la planificación no se cumple.
- ◦ Si se impone un calendario, la calidad y la funcionalidad se resienten para lograrlo.

Organización madura (nivel 4-5 CMM)

- ◦ Hay habilidad generalizada para gestionar los procesos de desarrollo software.
- ◦ El proceso es conocido y todo se desarrolla de acuerdo con lo planificado.
- ◦ Los procesos son consistentes (responden a lo esperado).
- ◦ Las responsabilidades y papeles están claramente definidos.
- ◦ La planificación está basada en datos históricos y es realista.

Existe un camino evolutivo desde la situación caótica de procesos (nivel CMM Inicial) hasta alcanzar unos procesos software disciplinados (nivel CMM Optimizado), es decir, hay un camino evolutivo que incrementa la madurez por etapas.

3.2.1. CMM: Modelo de Madurez de la Capacidad del Software

Entendemos por **capacidad** de un proceso de desarrollo de software el conjunto de resultados que se pueden esperar siguiendo un proceso dado. Esta *capacidad* proporciona un medio de predecir los resultados del siguiente proyecto a abordar.

Entendemos por **madurez** del proceso de desarrollo software el grado de profundidad con que un proceso está explícitamente definido, gestionado, medido, controlado y es eficaz.

El modelo CMM pretende definir los procesos por los que tiene que pasar una organización para obtener un software de calidad:

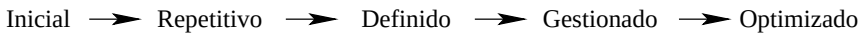


Figura 3.1: Niveles de madurez CMM.

Nivel inicial
: Lo primero es conseguir predicción rudimentaria de costes y tiempo. Hasta que el proceso no está bajo control estadístico no se puede tener un progreso en las mejoras. El proceso debe ser medible.

Nivel repetible
: Alcanzado un proceso estable con un nivel repetible de control estadístico, se inicia una gestión rigurosa del proyecto con compromisos en costes, tiempo y cambios.

A partir de aquí se pueden introducir tecnologías y metodologías avanzadas (Metrica v3, Merisse, SSADM) que sin una planificación y gestión eficaz son muy difíciles de introducir con buenos resultados.

3.2.2. Gestión de Proyectos

Un **proyecto** es una acción iniciada por la empresa en la que recursos humanos, financieros y materiales se organizan de una nueva forma para acometer un trabajo único, en el que dadas unas especificaciones y dentro de unas limitaciones en coste y tiempo, se intenta conseguir un cambio beneficioso definido por unos objetivos cualitativos y/o cuantitativos.

Un *proyecto* es algo novedoso, bajo el que hay que organizar multitud de aspectos, entre ellos incluso el esfuerzo de los recursos humanos. Se necesita un desarrollo de objetivos, que es lo primero a definir.

Llamamos **gestión de proyectos** al sistema de procedimientos, prácticas, tecnologías y conocimientos que facilitan la planificación, organización, gestión, dirección y control necesarios para que el proyecto termine con éxito.

3.2.3. Ciclo de Vida en la Gestión de Proyectos

La forma en que se aplican los procesos de gestión cambia conforme el proyecto avanza. Existen modelos de ciclo de vida, con variación en el número de estados:

Etapa	Nombre	Objetivos de Gestión
Germinación	Propuesta e Iniciación	Definición del proyecto Alcance y objetivos Diseño funcional Viabilidad Estimación inicial Decisión de continuar o no
Crecimiento	Diseño y Validación	Diseño del sistema para su validación Planes y recursos Estimación Producto base Validación
Madurez	Ejecución y Control	Formación y comunicación Planificación detallada y diseño Control estimación Asignación del trabajo Seguimiento del progreso Terminación de las previsiones Control y recuperación
Muerte	Finalización y Cierre	Terminación del trabajo Usos del producto Consecución de beneficios Disolución/recompensa del equipo Auditorías y revisión Registro histórico de los datos

Cuadro 3.1: Etapas del Ciclo de Vida de la Gestión de un Proyecto.

Cada etapa debe ser vista como un miniproyecto, repitiendo los procesos de gestión en cada una. El paso a una nueva etapa debe ser visto como un nuevo proyecto.

3.2.4. Informe de Definición del Proyecto

Debe incluir los siguientes puntos:

Definición
Qué pretende el proyecto en términos de resultados finales.

Justificación
Si el esfuerzo de desarrollo justifica la inversión a realizar.

Objetivos y metas
Respectivamente, resultados a largo plazo, estrategias y resultados tangibles. Servirán para determinar si el proyecto concluye con éxito.

Factores críticos del éxito
Son aquéllos que harán que un proyecto concluya con éxito o fracaso (en informática, generalmente es el usuario).

Riesgos
Identificar las posibles fuentes de problemas, ya sean de negocio, técnicas de usuarios, de implantación o de dirección del proyecto.

Alcance del proyecto
Fija las fronteras del proyecto: funciones principales, cantidad a producir, recursos requeridos, especificaciones de calidad, fecha de entrega, objetivo,...

Estructura del proyecto
El proyecto se subdividirá si la complejidad lo exige. Cada subproyecto debe incluir título, objetivos, alcance, producto final, hitos, riesgos/dependencias y responsables.

Fecha prevista de finalización

Fases, actividades, tareas e hitos

Organización
Refleja la estructura organizativa del proyecto y responsabilidad de cada recurso.

Sistema de control
Forma de realizar las revisiones del trabajo en el proyecto: Plan de garantía de calidad y Gestión de excepciones (actividades de acciones correctivas).

Supuestas dependencias y restricciones
Se indicarán los supuestos sobre el entorno de desarrollo, dependencias con respecto a otros proyectos, etc.

Presupuesto y gestión del presupuesto
Se indicará el presupuesto disponible, forma de gestionarlo y responsable (cash-flow).

Proyectos relacionados

3.2.5. Finalización del Proyecto

En la etapa de Finalización y Cierre, el equipo de trabajo debe asegurarse de que el trabajo se ha completado a tiempo y de forma eficiente. Para ello se debe dar término formal al proyecto:

1. ***Finalización del trabajo***.- Para comprobar si el trabajo se ha finalizado a tiempo se debe controlar:

- Listas de trabajos a realizar y realizados.
- Planificar y controlar los trabajos.
- Establecer reuniones de control.
- Planificar la disolución del equipo.
- Cerrar contratos con terceros (subcontratistas).

2. ***Transferencia del producto a los usuarios***.- Debe planificarse incluso la transición, aceptación del producto por parte de los usuarios, formación a los mismos y garantía del mantenimiento del producto y niveles de servicio.
3. ***Obtención de beneficios***.- Se ha de garantizar que se alcanzan los beneficios esperados. Para conseguirlo se debe:
   - Establecer una medida de referencia.
   - Calcular variaciones (comparando con la línea base) entre lo medido y lo esperado.
   - Tomar acciones correctoras para eliminar las variaciones.
4. ***Disolución del equipo***.- Se debe hacer de forma eficiente:
   - Planificar la disolución del equipo.
   - Devolver los recursos a su origen.
   - Mantener una reunión de final de proyecto.
   - Repartir premios, "castigos", evaluaciones y otras consideraciones.
5. ***Revisiones post-finalización***.- Incluyen:
   - Configuración final.
   - Comparar los costes y beneficios finales para retroalimentar el proceso de estimación (históricos, curva de productividad, interpolación).
   - Registrar los logros técnicos del proyecto para realimentar el proceso de desarrollo y la selección de futuros proyectos.
   - Revisar éxitos y fracasos del proyecto para el futuro.

Datos reales

[Lederer y Prasad, 1992][Gibbs, 1994][Standish Group, 1994]
Alrededor de dos tercios de todos los proyectos superan ampliamente (en más de un 50 %) sus estimaciones.

[Jones, 1994]
Los grandes proyectos se retrasan en su fecha de entrega entre un 25 % y un 50 % de su duración, y el retraso medio se incrementa con el tamaño del proyecto.

[Symons, 1991][McConnell, 1996]
Proponen, para evitar los problemas anteriores, desarrollo rápido y proyectos pequeños.

3.2.6. Gestión y Planificación de Proyectos: el camino

Es la ruta menos transitada, lo que puede parecer arriesgado, pero la ruta más concurrida es la que actualmente redunda a:

- ⋆ Costes masivos y retrasos
- ⋆ Baja calidad
- ⋆ Proyectos cancelados
- ⋆ Muchos cambios del personal
- ⋆ Fricciones entre personas
- ⋆ etc.

Los beneficios que se conseguirán con la otra alternativa hablan por sí solos:

- ▷ Reducir la sensación de incompetencia
- ▷ Controlar tiempo y costes del proyecto
- ▷ Incrementar la productividad al conocer las capacidades y sobrecargas
- ▷ Mejorar la imagen externa
- ▷ Seriedad
- ▷ Proyectos de alta calidad

La planificación es un remedio a muchos males, pero no a todos, y además debemos tener cuidado, pues:

- ↪ Se requiere tiempo (de 3 a 4 años)
- ↪ Se requiere esfuerzo (de la dirección y de los trabajadores)
- ↪ Se requiere formación
- ↪ Se requieren herramientas

Pero no hay otra solución. "Las soluciones simples tienden a funcionar sólo con problemas sencillos, y el desarrollo software no lo es" ([McConnell, 1996]).

Igual que músicos prestigiosos sin director, en software, equipos de desarrolladores inteligentes, de alta profesionalidad y dedicados emplean buenas y recientes técnicas y siguen sin alcanzar sus objetivos de planificación, si es que los tienen.

Se puede obtener un desarrollo rápido, económico y de calidad si nos apoyamos en una serie de pilares fundamentales (figura 3.2, [McConnell, 1996]):

Primer pilar El error clásico de descuidar la calidad del proyecto en sus fases iniciales supone desperdiciar tiempo y dinero más adelante corrigiendo defectos, justo lo más caro: retraso en tiempo (factor de $0'5$) y aumento en costes (factor de $1'5$).

Segundo pilar Si no analizamos, diseñamos y codificamos por este orden, nuestro sistema fallará cuando cambie la concepción del producto durante el desarrollo: retraso en tiempo y aumento en costes.

Tercer pilar Si no controlamos los riesgos podemos descubrir muy tarde que un subcontratista, por ejemplo, se ha retrasado: retraso en tiempo y aumento en costes.

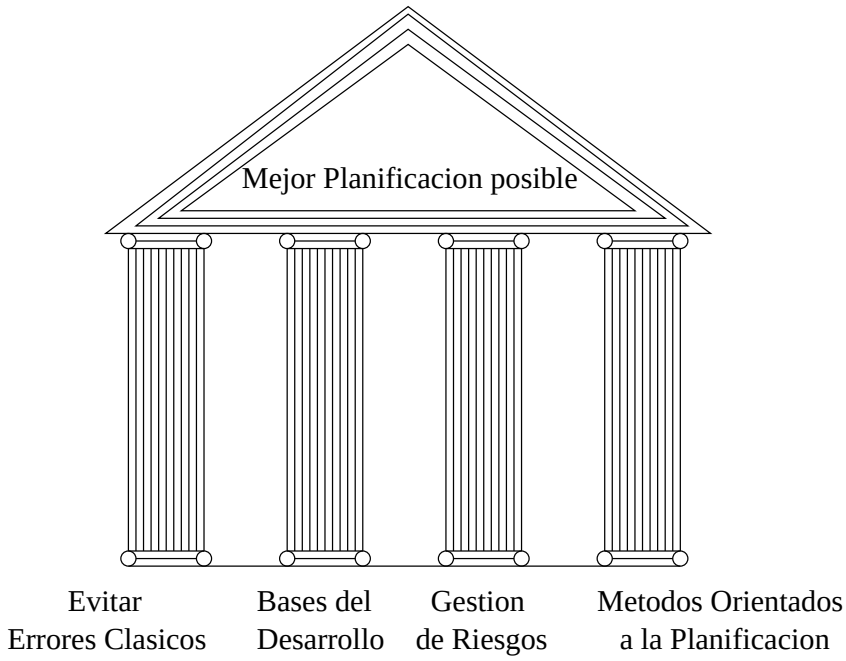


Figura 3.2: Pilares fundamentales de la Planificación.

Estos tres pilares son la mayor parte del soporte necesario para obtener el mejor plan posible. La planificación sola no sirve para nada, y sin seguimiento tampoco.

3.2.7. Cuatro dimensiones de un Proyecto Software

Las cuatro dimensiones de un proyecto software, son las **cuatro P**s que ya hemos mencionado en alguna ocasión (ver figura 3.3).

Potenciar estas cuatro dimensiones maximiza la velocidad de desarrollo y aumenta la calidad. Además, la planificación será más completa, creativa, efectiva y satisfará mejor a todos.

Personas

Los temas relacionados con personas tienen un gran impacto en la productividad del software y por tanto en su calidad.

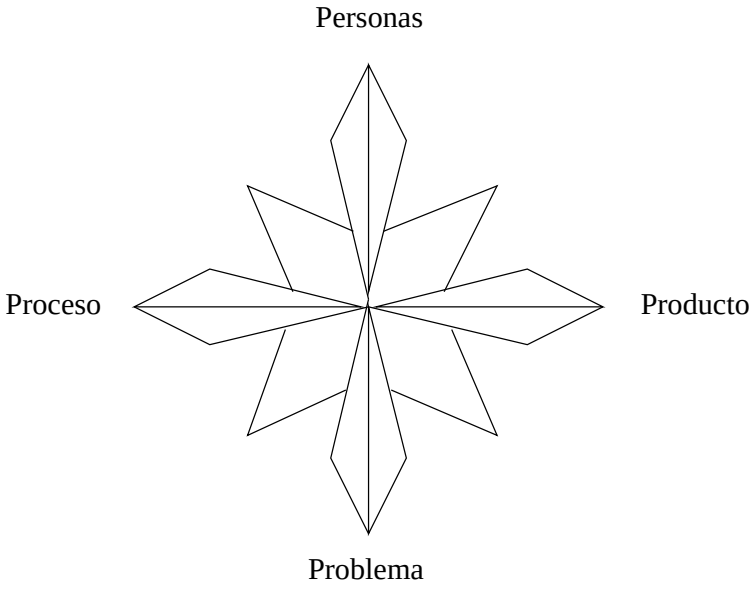


Figura 3.3: Las cuatro dimensiones de un Proyecto software.

La tecnología no es la respuesta; los métodos efectivos son aquéllos que sacan partido al potencial humano de los desarrolladores.

La mejora en la productividad conlleva ocuparse de temas relacionados con el personal: motivacion, equipo de trabajo, selección de personal, formación, gestión de personal...

Barry Boehm, en su libro *Software Engineering Economics* (1981) destaca los siguientes aspectos relacionados con las personas:

- *Máximo talento*: mejor tener un buen personal en general que unos pocos talentos en medio de los demás
- *Trabajo adecuado*: explotar las habilidades personales y motivar a la gente (no inculcar otras)
- *Progresión profesional*: ayuda a la actualización en vez del encajonamiento y la monotonicidad
- *Equilibrado del equipo*: seleccionar a la gente que se complemente y armonice con los demás
- *Eliminar inadaptación*: eliminar y reemplazar a los miembros problemáticos del equipo lo antes posible

Proceso

Se trata de la aplicación de metodologías de desarrollo (metodologías, estándares, procedimientos, técnicas y herramientas), planificación y gestión.

Como ejemplos de mejora en el proceso de desarrollo (citados por [Myers, 1992], [Gribbs, 1994]) se encuentran Hughes Aircraft, Lockheed, Motorola, Nasa o Xerox, que han reducido sus plazos de salida al mercado a la mitad y sus costes y errores (y por ende el mantenimiento) en un factor de 3 a 10.

Recomendaciones:

- Ocuparse del tamaño y características del producto da oportunidad de reducir la planificación.
- Intentar aplicar la regla 80/20: desarrollar el 80 % del producto (todo lo que no es algoritmia) en el 20 % del tiempo.
- Los productos más grandes conllevan más tiempo, y los más pequeños, menos.
- Más prestaciones implica más análisis, más diseño, más construcción, más pruebas, más mantenimiento, más tiempo y más costes.

Las consideraciones que se hacen a propósito del producto son:

- El esfuerzo para construir software se incrementa desproporcionalmente (exponencialmente) más rápido que el tamaño del software.
- La reducción del tamaño mejorará la velocidad del desarrollo también desproporcionadamente.
- Reducir a la mitad el tamaño de un programa intermedio normalmente supone una reducción de al menos dos tercios del esfuerzo.
- Objetivos ambiciosos de rendimiento, robustez y fiabilidad suponen más tiempo y más coste.

Estos puntos llevan a:

- √ Desarrollar sólo las prestaciones esenciales
- √ Desarrollar el producto por etapas, incrementalmente
- √ Emplear lenguajes de alto nivel
- √ Emplear herramientas, técnicas y metodología (no sólo en los aspectos propios de ingeniería del software)

En cuanto a la tecnología, se deben emplear herramientas lo más efectivas posible (herramientas CASE, de pruebas, lenguajes de cuarta generación,...)[2].

Neil Olsen extrajo las siguientes cifras de un estudio en 1995:

- Pasar de una inversión baja a una media en personal, formación y entorno de trabajo produce unas ganancias similares a la inversión.

[2]Es decir, Delphi vs. ensamblador, ERWIN y BPWIN vs. scripts o programar, reutilización vs. repetición, Requisite Pro vs. documentos.

- Inversiones adicionales se justifican con ganancias 1 a 1 aproximadamente.
- Pasar de una inversión media a alta dispara la productividad con ganancias 2 a 1 o 3 a 1.

Los métodos de ingeniería del software también ayudan:

- Estándares de codificación para reutilizar
- Revisiones de diseño y código
- Metodologías
- Tecnicas de desarrollo

3.2.8. Evitar errores clásicos en el desarrollo de software

El primer pilar de la planificación que veíamos (figura 3.2, página 62) nos hablaba de evitar errores clásicos. Veremos ahora un desglose de los posibles errores evitables, un tema clave porque en software un error puede dar al traste con todo un proyecto.

Errores clásicos relacionados con las personas

↪ *Motivación débil*: recompensas no adecuadas, no reconocimiento, horas extras,...

↪ *Personal mediocre*: contrataciones inadecuadas, personal problemático,...

↪ *Hitos*: sólo fijarse en que se cumplan los hitos y no cómo va el proyecto día a día; hay personas que se rigen por hitos, algo que no es adecuado.

↪ *Añadir más personal* a un proyecto retrasado, generalmente lo retrasará aún más.

↪ *Oficinas repletas y ruidosas*: se requiere silencio y privacidad.

↪ *Fricciones* entre clientes (usuarios) y desarrolladores.

↪ *Expectativas poco realistas*: los directivos suelen planificar el proyecto de forma muy optimista. Esto al final se percibe como un plan muy largo y muy malo.

↪ *Falta de participación* de los implicados.

↪ *Falta de participación* del usuario: cuando no se implica al usuario desde el principio, no se entienden ni se consideran sus requisitos. Esto originará retrasos y decepciones.

↪ *Ilusiones y confianza* en la bondad del destino: se cree que el destino deparará algo mejor de lo que se piensa y generalmente es justo lo contrario.

Errores clásicos relacionados con el proceso

↪ *Planificación excesivamente optimista*: optimista sí, pero los retos suelen no cumplirse. Se acaba minando la moral y la productividad.

↪ *Gestión de riesgos* insuficiente.

↪ *Planificación* insuficiente.

↪ *Abandono de la planificación* bajo las presiones: el fallo no es abandonar el plan, es no crear un plan alternativo.

↪ *Inicio* difuso.

↪ *Escatimar* en las actividades iniciales: análisis y diseño.

↪ *Control insuficiente* de la dirección.

↪ *Omitir tareas* necesarias en la estimación.

↪ *Programación a destajo* con métodos o técnicas estándares.

↪ *Fallos* en las subcontrataciones.

Errores relacionados con el producto

↪ *Exceso de requisitos*. Los requisitos deben ser los absolutamente necesarios.

↪ *Cambio de prestaciones*: como media hay un 20 % de cambios en los requisitos a lo largo del ciclo de vida de un proyecto, lo que lleva a un 25 % de aumento en el plan ([Jones, 1994]).

↪ *Desarrolladores meticulosos* y fascinados por la tecnología.

↪ *Desarrollo orientado a la investigación*: no intentar sobrepasar los límites de la ingeniería en más de dos áreas a la vez. El riesgo de fallo es demasiado alto ([Seymour Cray]).

Errores relacionados conn la tecnología

↪ *Síndrome de la panacea* ante una tecnología.

↪ *Sobreestimación de las ventajas* del empleo de nuevas herramientas o métodos (incluso ciclos de vida, gestión de calidad,...): hay curvas de aprendizaje, las mejoras son lentas, etc.

↪ *Cambiar de herramientas o filosofía* a mitad del proyecto.

3.2.9. Bases del desarrollo del software

Se puede planificar bien, pero no considerar los aspectos básicos de desarrollo supone no alcanzar las metas planificadas.

Los métodos fundamentales de la ingeniería del software deben emplearse porque reducen el coste y el tiempo de comercialización. El SEI (*Carnegie Mellon University*)

ha observado que implantar la disciplina de la ingeniería del software antes que la gestión de procesos es un fracaso ([Burlton, 1992]).

La gestión controla la planificación, el coste y el producto. Los fundamentos de gestión consisten en determinar el tamaño del producto, asignando los recursos apropiados a un producto de ese tamaño, creando un plan para aplicar esos recursos y luego controlando y dirigiendo los recursos para impedir que el proyecto se desvíe (es decir, planificar, gestionar y hacer seguimiento).

Los proyectos bien ejecutados pasan por tres etapas básicas para crear una planificación software:

- Estimación del tamaño del producto.
- Estimación del esfuerzo necesario para un producto de ese tamaño.
- Realización de una planificación, basándose en la estimación del esfuerzo.

Planificación y estimación son las bases del desarrollo (junto con los ciclos de vida), de forma que una mala estimación reduce eficiencia en el desarrollo, mientras que una estimación correcta es necesaria para una planificación efectiva y un desarrollo eficiente.

Para poder afrontar esto se dispone de diferentes métodos de estimación: *CO.CO.MO*, *Puntos de Función*, método de la *WBS*, etc.

En la revisión de un grupo de los "mejores proyectos" seleccionados por las empresas, se encontró que los mejores se caracterizaban por una fuerte planificación anticipada para definir las tareas y programaciones[3] ([Hetzel, 1993]).

La planificación sola no basta, pues. Una vez hecha debe seguirse el proyecto para comprobar que se está cumpliendo el plan previsto en objetivos de planificación, coste y calidad.

Para el seguimiento se harán:

- Reuniones periódicas
- Informes periódicos sobre el estado del proyecto
- Revisiones de hitos
- Informes de presupuestos

Hetzel en los "mejores proyectos" descubrió que se había hecho una estricta medición y seguimiento del estado del proyecto.

> **[Capers Jones, 1995]**
> "El control del proceso software es tan malo que muchos desastres software no se conocen hasta el mismo día del despliegue esperado".

[3] Programación = Planificación + Asignación de Recursos.

[**SEI, 1993**][**Kitson y Masters, 1993**]
"El 75 % de las empresas necesitan mejorar la supervisión y seguimiento de sus proyectos".

[**Baumert, 1995**]
"En las evaluaciones de las empresas, los principales problemas aparecen en las áreas de planificación, seguimiento y supervisión del proyecto".

Medidas o métricas. Históricos

Una forma de progresar, a largo (medio) plazo, es recoger datos numéricos para analizar la calidad y productividad del software.

Se deben recoger datos de: costes, tiempo, planificación, tamaño de los programas (en líneas de código, LOC, por ejemplo).

Si los datos son insuficientes, recoger más datos puede suponer mucho tiempo. Sin embargo, con los anteriores se tienen las bases para la planificación de proyectos futuros, que siempre es mejor que el instito para responder a preguntas como: *¿Podemos desarrollar este producto en 3 meses? Bueno, nunca hemos desarrollado un producto de ese tamaño en menos de 11 meses y el tiempo medio es de 13...*

Si un software tiene demasiados errores se empleará más tiempo corrigiéndolo que escribiéndolo. Todo funcionará mejor si no se cometen los errores en el primer paso.

La peor decisión es ahorrar tiempo reduciendo el poco tiempo que se le dedica ya al análisis, al diseño, a las revisiones de código, a la planificación y a las pruebas.

[**Capers Jones, 1991**]
"IBM fue la primera compañía en descubrir que la calidad y la planificación del software estaban relacionadas. También descubrieron que los productos con el menor número de defectos eran los que tenían mejor (mayor) tiempo de planificación (buena estructuración en tareas y buena asignación de recursos)."

[**Capers Jones, 1994**]
"El número de defectos (baja calidad) que se dan cuando los programadores lanzan productos bajo una excesiva presión en la planificación es hasta 4 veces mayor."

3.2.10. Gestión de riesgos

Los proyectos de software incluyen un amplio conjunto de riesgos:

- Cambios en los requisitos de usuario.
- Mala estimación de la planificación.
- Personal contratado poco fiable y efectivo.
- Falta de experiencia en la gestión.
- Problemas de personal.

- Problemas con la tecnología.
- Problemas con el desarrollo.
- Problemas con los proveedores.
- ...

Probabilidades en el mundo real

La probabilidad de finalizar un proyecto complejo en el tiempo estimado tiende a cero. La probabilidad de cancelar un proyecto complejo tiende a 0'5.

[**Peet Marwick, 1988**]
"De 600 empresas, el 35 % han tenido al menos un proyecto que se les fue de las manos.

Por ejemplo, AllState comenzó a automatizar en 1982 sus actividades administrativas; planificaron 5 años y 8 millones. Seis años más tarde y 15 millones de dólares después, AllState estableció una nueva fecha límite y estimó 100 millones en gastos."

[**Tom Gills**]
"Si no controlas los riesgos, ellos te controlarán a ti".

La función de la gestión de riesgos (del software) es identificar, estudiar y eliminar (en la medida de lo posible) las fuentes de riesgos antes de que empiecen a amenazar la finalización satisfactoria de un proyecto (software).

Se pueden controlar los riesgos a varios niveles ([Pressman, 1993]):

Nivel 1: Control de crisis
Controlar los riesgos sólo cuando se han convertido ya en problema.

Nivel 2: Arreglar cada error
Detectar y reaccionar rápidamente ante cualquier riesgo, pero sólo después de haberse producido.

Nivel 3: Mitigación de riesgos
Planificar con antelación el tiempo que se necesitaría para cubrir riesgos en el caso de que ocurran, pero no intentar eliminarlos inicialmente.

Nivel 4: Prevención
Crear y llevar a cabo un *plan* (sin ser formal y riguroso) como parte del proyecto software para identificar riesgos y evitar que se conviertan en problemas.

Nivel 5: Eliminación de las causas principales
Identificar y eliminar los factores que puedan hacer posible la presencia de algún tipo de riego.

Si se está en los niveles 1, 2 ó 3, ya se ha perdido la batalla de la planificación.

La **Gestión de Riesgos** se compone de ([Boëchm, 1989]):

1. *Estimación de Riesgos*
   a) *Identificación de Riesgos*

      Genera una lista de riesgos capaces de romper la planificación del proyecto.

      Los riesgos se parecen mucho a los errores clásicos ya mencionados. La diferencia es que los errores clásicos se comenten con mayor frecuencia que los riesgos. Los riesgos son menos comunes o pueden ser únicos para un proyecto determinado.

   b) *Análisis de Riesgos*

      Mide la probabilidad y el impacto de cada riesgo y los niveles de riesgo de los métodos alternativos[4].

      Una vez identificados para un proyecto, el paso siguiente es analizar cada riesgo para determinar su impacto. Para cada riesgo identificado se determina la exposición a riesgos (ER), esto es:

      $$\text{Probabilidad de pérdida no esperada} \times \text{Magnitud de pérdida}$$

      donde *pérdida no esperada = riesgo*.
      Ahora sólo queremos los riesgos de planificar, por lo que las pérdidas se expresan en unidades de tiempo. Las probabilidades y las magnitudes de pérdida hay que estimarlas, sin pretender ser exactos.
      Si se suman todas las ER se obtiene el retraso total del proyecto y se puede emplear para ajustar la planificación con un margen de retraso.

   c) *Priorización de Riesgos*

      Genera una lista de riesgos ordenados por su impacto.

      Los proyectos suelen gastar el 80 % de su presupuesto en arreglar el 20 % de sus problemas. Conclusión: centrarse, fundamentalmente, en ese 20 %.
      Se pueden ordenar los riesgos por su ER y así saber cuáles controlar para compensar esfuerzo y reducción en tiempo de los riesgos. Se puede decidir controlar riesgos especiales si la magnitud de la pérdida es considerable.

2. *Control de Riesgos*
   a) *Planificación de la Gestión de Riesgos*

      Genera un plan para tratar cada riesgo significativo. También asegura que los planes para la gestión de riesgos de cada uno de ellos son consistentes entre sí y con el plan del proyecto.

[4] No hay que tener en cuenta solamente que la probabilidad sea alta, porque si es baja pero el impacto es muy grande...

El plan de gestión de riesgos puede ser tan sencillo como un párrafo por cada riesgo, describiendo: qué hay que hacer, quién tiene que hacerlo, cuándo y cómo.

b) *Resolución de Riesgos*

Es la ejecución del plan para resolver cada uno de los riesgos significativos.

Depende del riesgo específico que se esté tratando. Posibles acciones son:

- Evitar el riesgo: no realizar actividades arriesgadas.
- Trasladar el riesgo de una parte del sistema a otra: que los expertos en una materia supervisen a los noveles y que el riesgo no esté en el camino crítico de la planificación.
- Informarse del riesgo: conocerlo mejor y ver si es o no posible.
- Asumir el riesgo: si las consecuencias son pequeñas y el esfuerzo es grande.
- Comunicar: difundir el posible impacto de un riesgo en caso de ocurrir.
- Controlar el riesgo: planificar acciones en caso de ocurrir.
- Recordar el riesgo: para proyectos futuros (gestión del conocimiento, buenas y malas prácticas, lecciones aprendidas).

c) *Monitorización de Riesgos*

Es la actividad del progreso (pueden aparecer nuevos riesgos, desaparecer algunos, cambiar su probabilidad e impacto,...) de la monitorización dirigido a la resolución de cada elemento de riesgo.

Los riesgos aparecen y desaparecen en el desarrollo del proyecto. Por esto se necesita una monitorización de riesgos, para comprobar cómo progresa el control de un riesgo, identificar cuándo aparecen/desaparecen nuevos riesgos,...

3.2.11. Estimación

Cualquier proyecto debe justificarse, bien por ahorro en costes de operación, de expansión de negocio, mejora del proceso,... Acompañado de una valoración de la viabilidad técnica y la inversión detallada, con el coste total y desglosado, y plazos en que se obtendrán los beneficios.

Es decir, es necesaria la estimación del nuevo proyecto.

La credibilidad de la estimación es baja. Podemos comprobarlo refiriéndonos a estudios:

- ⋆ El 5 % de los proyectos cumplen el presupuesto estimado
- ⋆ El 10 % lo exceden en un 10 %
- ⋆ El 20 % lo exceden en un 25 %
- ⋆ El 30 % lo exceden en un 50 %
- ⋆ El 35 % lo exceden en un 100 % o más

Según Tom Demarco, un 15 % de los proyectos no producen nada útil al finalizar.
Una buena regla es que los beneficios superen en al menos 5 veces los costes (salvo que el proyecto sea obligado).

La **estimación** es la actividad que permite obtener respuesta a *¿cuánto costará?*, *¿cuánto tiempo llevará hacerlo?*.
Las dificultades de una estimación son:

- No hay un modelo de cómo hacerlo en todas las organizaciones.
- Se necesitan diferentes estimaciones para las diferentes personales que hay en un proyecto (alta dirección, dirección del proyecto y cada recurso).
- La utilidad de una estimación depende de la etapa de desarrollo en que estamos: se precisa de diferente grado de exactitud a medida que avanza el proyecto.
- Se suele hacer muy superficialmente y bajo presión.
- Hay muchos cambios y la información de partida es muy vaga en ocasiones.
- Los rápidos cambios en las tecnologías de información son problemas para la estabilidad de un proceso de estimación.
- Falta de históricos y experiencia en estimar proyectos.
- Tendencia a subestimar, ignorando aspectos no lineales como coordinación y gestión.

En suma, la estimación es una tarea muy difícil pero imprescindible en planificación.

El proceso para planificar un desarrollo consta de tres pasos:

1. **Estimar el tamaño del producto**.
   Es el paso más difícil con diferencia. Se puede hacer a través del LOC o por Puntos de Función[5].
   Se puede estimar el tamaño de un proyecto de varias formas:
   - Utilizar un enfoque algorítmico (por ejemplo, puntos de función) para estimar el tamaño del programa a partir de las prestaciones.

[5]Miden la funcionalidad que se le da al usuario.

- Utilizar un software de estimación, a partir de la descripción de las prestaciones del programa (pantallas, informes, archivos, consultas, tablas de la BD,...).
- Emplear experiencias en proyectos similares.

2. **Estimar el esfuerzo**.
   Si hay una buena estimación del tamaño y un histórico de la organización en proyectos similares, es "fácil".
   Obtenida la estimación del tamaño, se puede derivar la estimación del esfuerzo. Esta estimación no es estrictamente necesaria para estimar una planificación, pero ayuda a:
   - Saber cuántas personas hay que incorporar al proyecto.
   - Estimar la planificación.

   Para convertir la estimación del tamaño en la del esfuerzo se emplea software de estimación, tablas de conversión, históricos e incluso métodos algorítmicos de aproximación como el CO.CO.MO. de Boëhm.
3. **Estimar la planificación**.
   Estimados tamaño y esfuerzo, es sencillo.
   Para calcular la estimación de la planificación se usa la fórmula:

$$\text{Planificación (meses)} = 3 \times \text{personas-mes}^{1/3}$$

Puntos de Función

Un **punto de función** es una medida sintética (y ficticia) del tamaño del programa.

Se suele emplear en los primeros estados del proyecto. Son más fáciles de determinar a partir de los requisitos que las LOC y dan una medida más exacta sobre el tamaño del programa (proyecto).

Existen diferentes métodos para contabilizar los puntos de función. Nosotros emplearemos uno basado en 1984 I.B.M. Method, según el cual el número de puntos de función de un programa se basa en el número y la complejidad de:

entradas pantallas, formularios, cuadros de diálogo, controles o mensajes para añadir, modificar o borrar datos del programa

salidas pantallas, informes, gráficos o mensajes que se generan para el usuario final u otro programa

consultas combinaciones de E/S en las que cada entrada genera una salida simple e inmediata. Las salidas pueden procesar, combinar o resumir datos complejos y presentar muchos formatos

archivos lógicos internos pueden ser BD o archivos planos. Son los principales grupos lógicos de datos de usuario finales, o información de control, que están completamente controlados por el programa

archivos de interfaz externos archivos controlados por otros programas con los que el programa va a interactuar

Este enfoque funciona bien para toda clase de software, pero si se están construyendo sistemas intensivos en BD tendrá que ajustarse al propio entorno, ya que, como se puede apreciar, no tiene en cuenta para nada la algoritmia de la lógica (problema).

	Complejidad baja	Complejidad media	Complejidad alta
Entradas[a]	2	4	6
Salidas	4	5	7
Consultas	3	4	6
Ficheros lógicos internos	2	10	15
Ficheros de interfaz externos	5	7	10

[a]Esto a su vez se suele dividir en varias filas: de 1 a 5 entradas, de 6 a 20, de 21 a 30, etc. (por ejemplo).

Para calcular el número de puntos función en un programa se toma el elemento y se multiplica por el factor indicado en la tabla. La suma de estos números da el "total de los puntos funcion sin ajustar" o "no ajustados".

Después se calcula un "multiplicador (o ajuste) de influencia basado en la influencia que tienen 14 factores sobre el programa. El intervalo de este multiplicador es de 0'65 a 1'35.

Se multiplican ambos valores para obtener la suma total de los puntos:

$$(PFNA = (\sum PFSA)) \times MA = PFA$$

Con el número anterior se puede comparar el tamaño y la planificación de proyectos anteriores y se podría estimar una planificación a partir de éstos. Otra posibilidad es emplear el *Método de Estimación de Primer Orden de Jones*, que veremos a continuación.

Método de Estimación de Primer Orden de Jones

Desarrollado por Capers Jones, el *método de primer orden de Jones* es un método de estimación que consiste en tomar el total de los puntos función y elevarlo a la potencia apropiada según una tabla. Dicha tabla de potencias surge del análisis de su BD de miles de proyectos.

En la tabla se cruza el tipo de software desarrollado con el tipo de empresa (media, muy organizada y poco organizada) y el resultado son los meses estimados.

No es la mejor forma de estimar la planificación pero da una aproximación que es mucho mejor que hacerlo a ojo.

Modelo CO.CO.MO. de Boëhm

Este modelo, desarrollado por Barry W. Boëhm en 1981, toma su nombre de las siglas de **CO***nstructive* **CO***st* **MO***del*.

El modelo CO.CO.MO. proporcina dos "versiones" de estimación, que se pueden aplicar según la exactitud: *básico* e *intermedio*. Asimismo, el modelo proporciona tres "modos", según las características de la aplicación: *orgánico*, *embebido* e *híbrido*.

Las ecuaciones de esfuerzo (se proporcionan también unas para el tiempo de desarrollo) son del tipo:

$$E = a \times S^b \times m$$

donde

E	esfuerzo de desarrollo en meses/hombre
a y b	constantes para cada modo y nivel del modelo
S	LOC del producto software terminado (estimación)
m	factor de ajuste determinado a partir de 15 atributos (factores) que inciden en el coste

Según Boëhm se pueden hacer estimaciones de un 20 % de error en el 70 % de los casos (si se estiman bien las LOC es una buena estimación).

El **CO.CO.MO. básico** permite dar una estimación durante la fase de análisis previo, cuando la mayoría de los factores que incidirán en el proyecto son todavía desconocidos, apoyándose tan sólo en el número de líneas "fuente" previstas para el proyecto.

Por su parte, el **CO.CO.MO intermedio** toma en consideración además, con los 15 atributos como modificadores de los resultados obtenidos, aspectos tales como:

- capacidad de programadores y analistas
- la seguridad y la fiabilidad requerida del software que se va a desarrollar
- tipo de herramientas empleadas en su desarrollo y la eficacia de las mismas
- ...

Los modos de operación en CO.CO.MO. (tanto para CO.CO.MO. básico como intermedio —distintos valores de las constantes a y b—) se especifican:

Modo orgánico para proyectos pequeños y poco complejos

Modo embebido para proyectos de mucha envergadura

Modo híbrido representa un compromiso entre los dos anteriores

Los supuestos que Boëhm implica son:

- ▷ La base de la estimación es el número de instrucciones "fuente" que nos será necesario escribir para desarrollar el proyecto (no se consideran las instrucciones para realizar las pruebas, ni para puesta a

punto, llamadas a rutinas ya hechas, comentarios, programas de apoyo a utilidades ya existentes y sí el número de líenas de los J.C.L.[6].

- ▷ Las estimaciones abarcan desde la fase de diseño hasta la implantación y pruebas. Las demás fases se deberán estimar por separado (no entran dentro del modelo CO.CO.MO.).
- ▷ La unidad de medida de las estimaciones del esfuerzo es meses/hombre (152 horas —1 mes-hombre— de tiempo laborable real).
- ▷ Presupone que se hace una correcta planificación y gestión; se sigue una metodología,... Si no es así, las estimaciones sufrirán desviaciones importantes (escasa utilidad).
- ▷ Pocos cambios en los requisitos (el número de LOC/LDC no cambia).

Además, hay una serie de consideraciones a tener en cuenta:

- √ Este modelo no es válido cuando hablamos de software de sistema, es decir, no incluye software en tiempo real, control de procesos o cosas similares. El motivo es que la productividad de estos sistemas es mucho más baja (software de base).
- √ Las planificaciones se obtienen en meses e incluyen diseño, construcción y pruebas, pero no incluyen el tiempo necesario de especificación de requisitos.
- √ Cuando hablamos de LDC (LOC), nos referimos a sentencias fuente sin comentarios y sin espacios en blanco. En C y Pascal, una LOC sería aproximadamente el número de ";", pudiendo tener más de una línea física para una sentencia es una LOC. Además, no son lo mismo para ensamblador que para Delphi.

[6] Job Control Language.

Capítulo 4

Gestión y Planificación de Proyectos

4.1. Objetivos de un Sistema de Planificación de Proyectos

Los objetivos de un **sistema de planificación de proyectos** son:

- Satisfacer las necesidades de información de gestión a todos los niveles y en sus diferentes vertientes.
- Fomentar una cultura de gestión que contribuya al aumento de la productividad (Dirección por objetivos y Control de gestión).
- Contribuir a la estructuración y homogenización de los métodos de trabajo con objeto de:
  - Facilitar las labores de formación.
  - Permitir la reasingación eficaz de los recursos.
  - Permitir un seguimiento compartido de la gestión.
  - Dotar de utilidad a la información histórica.
- Mejorar la comunicación entre el personal involucrado.

A partir de la planificación de un proyecto, un gestor debe poder responder a preguntas del tipo:

√ ¿Cuál es el grado de realización de un proyecto X?

√ ¿Cuál es la calidad de la planificación del jefe de proyecto Y?

√ ¿Qué había pasado con aquel proyecto similar? ¿Cuál fue la proporción de esfuerzo y plazos en sus distintas fases?

√ ¿Los miembros del equipo de trabajo y los usuarios conocen sus responsabilidades y las actividades asignadas?

√ ¿Y las de las demás personas involucradas en el proyecto?

- √ ¿Cuándo va a finalizar la programación del susbistema X del sistema Y? ¿Cuándo se prevee finalizar la fase de implementación del proyecto Z?
- √ ¿Podemos retrasar, sin más, la realización de la actividad X en el proyecto Y? ¿Qué ocurre con otras actividades? ¿Y con el producto final? ¿Y con otros proyectos relacionados?
- √ ¿Qué repercusión produce en el proyecto y en los recursos necesarios la incorporación de esta modificación solicitada por el usuario?
- √ ¿Cuándo finaliza el analista X su participación en el proyecto?
- √ ¿Existen los recursos suficientes y adecuados para abordar un nuevo proyecto?
- √ ¿De qué proyecto puedo extraer recursos minimizando su impacto?
- √ ¿Qué carga de trabajo requeriremos de la unidad usuario X durante el período Y para atender a los proyectos en que está involucrado?
- √ ¿En qué estado de situación están los proyectos del área económico-financiera? ¿Están coordinados?
- √ ¿Qué actividades tengo que realizar y cuándo tienen que estar terminadas? (para cada miembro del equipo)
- √ ¿Cómo se ha comportado el analista X duante este proyecto? ¿Ha desempeñado cometidos por debajo o por encima de su nivel? ¿Ha cumplido con plazos y especificaciones?
- √ ¿Cuánto nos estamos gastando en desarrollo de productos?
- √ ¿Se encuentran en plazo los proyectos realmente estratégicos?
- √ ¿Cuál es la carga de análisis, programación, control de calidad e implantación en los proyectos de desarrollo?
- √ ¿Cuál es el valor del inmovilizado producido por este proyecto? ¿Qué criterios de amortización podemos aplicar?
- √ ¿Puede irse el analista X de vacaciones el mes de agosto?
- √ ¿Cuánto nos ha costado realmente este proyecto? ¿Cuáles han sido las desviaciones en plazo y coste?
- √ ¿Fue rentable el proyecto X? ¿Había otras posibilidades más rentables para su realización?
- √ Etc.

En resumen, los objetivos son:

- ▶ Facilitar la planificación, control y seguimiento de las actividades.
- ▶ Fomentar una cultura de gestión.
- ▶ Mejorar la comunicación y la coordinación.

- Mejorar la homogeneización y estructuración de los métodos de trabajo.
- Optimizar la disponibilidad de los recursos (humanos y no humanos).
- Obtener el estado consolidado de la distribución de esfuerzos.

4.2. Elementos de un Sistema de Planificación de Proyectos

Los elementos de la planificación se representan en la siguiente figura:

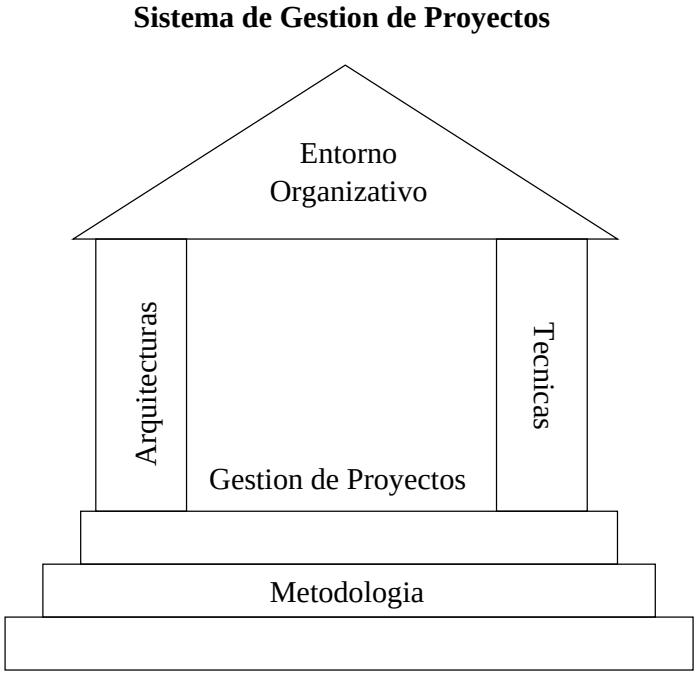


Figura 4.1: Elementos de un Sistema de Planificación.

4.3. Metodología de Planificación de Proyectos

Los elementos relevantes de la metodología son:

- Definición del proyecto.
- Modelo genérico del ciclo de vida de los proyectos.
- Metodologías específicas por cada tipo de proyecto.

Los veremos en detalle en las secciones siguientes.

4.3.1. Definición de un proyecto

La definición del proyecto debe adaptarse a las necesidades de cada organización.

Genéricamente, un **proyecto** es un conjunto de:

- Unos recursos humanos, materiales, financieros,...
- Una organización (relación entre actividades y recursos)
- Una planificación (relación entre actividades, recursos y tiempo)
- Unos productos específicos a obtener con los elementos anteriores
- Unos objetivos a alcanzar con los productos anteriores
- Un entorno de riesgo en el que se desenvuelven dichos elementos

Ámbito de un proyecto

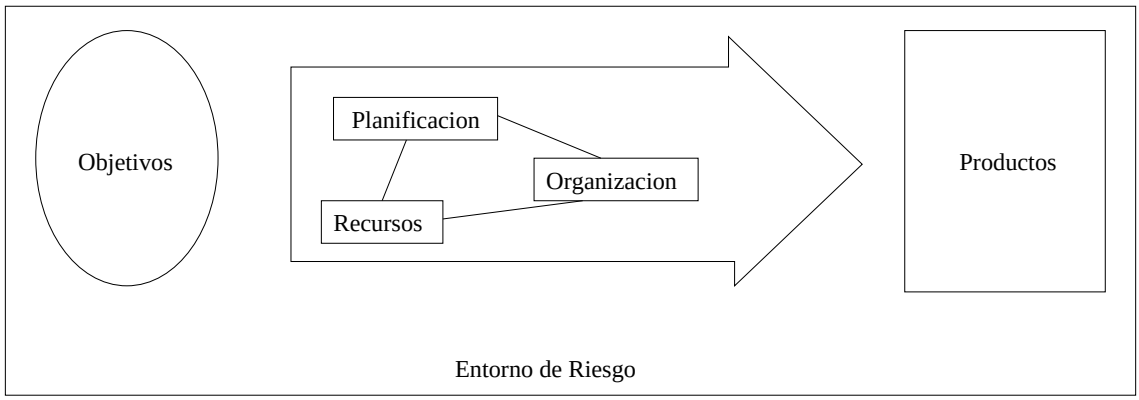


Figura 4.2: Ámbito de un proyecto.

Características de un proyecto

Un proyecto es algo:

Discreto: con un único inicio y final definidos, y con un producto final establecido.

Complejo: con un conjunto de diferentes tareas interrelacionadas.

Único: tanto en relación con su producto final como con su entorno de desarrollo.

Es decir, un proyecto no es un *proceso cotidiano* o *rutinario*, no es una de las "funciones normales" de la organización.

Gestión de un proyecto

La gestión de un proyecto tiene por objetivo la entrega de los productos finales del proyecto:

- √ en plazo
- √ dentro del presupuesto
- √ de acuerdo con las especificaciones
- √ con los niveles de calidad correspondientes a los estándares profesionales y a las expectativas de la dirección

Es decir, supone acabarlo en tiempo, en coste, esfuerzo y con los recursos estimados en un principio.

Las ventajas de la gestión de proyectos es que obliga a definir objetivos, programas, presupuesto, calidad y unicidad. No obstante, también tiene sus inconvenientes, ya que consume recursos, los programas elaborados pueden no ser realistas, así como el presupuesto o los objetivos, dando todo ello lugar a conflictos.

No obstante, la gestión se ve restringida por:

- peticiones de trabajo limitadas (no se pueden aceptar todas las propuestas de proyectos)
- fechas objetivo de entrega predeterminadas
- todas las peticiones con alta prioridad
- limitado número de recursos
- disponibilidad de recursos limitada
- cambios en los planes de los proyectos
- necesidad de recursos no planificados
- cambio en los equipos de trabajo
- ...

4.3.2. Modelo genérico de Ciclo de Vida de un Proyecto

Podemos hablar del ciclo de vida de un proyecto en términos de fases estándares (según la metodología *ITPM* de la consultora *KPMG*):

Fases propias

- Evaluación y aprobación del proyecto
- Planificación y puesta en marcha del proyecto
- Ejecución del proyecto

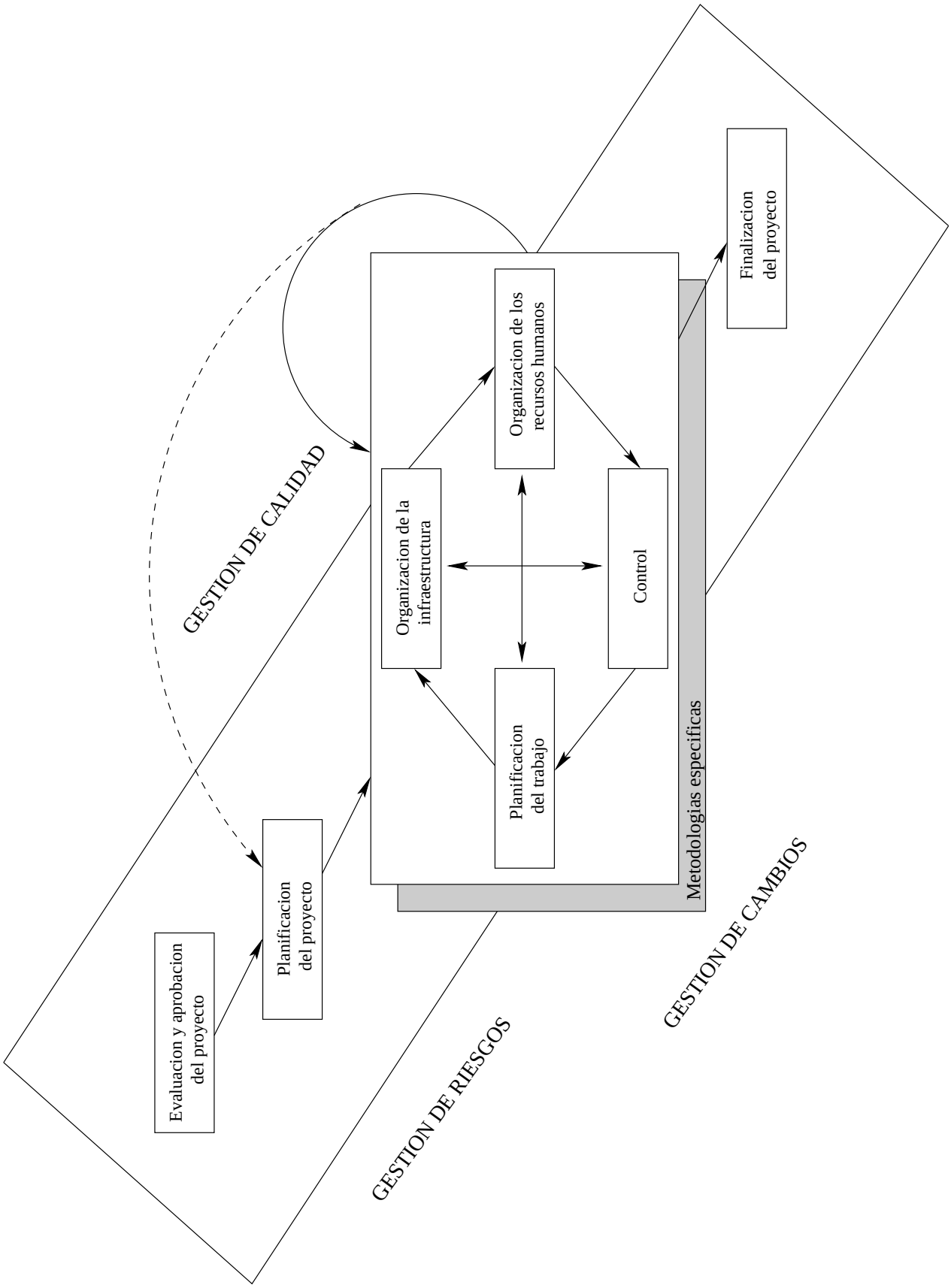


Figura 4.3: Ciclo de Vida de un Proyecto.

    - Planificación detallada del trabajo
    - Organización de la infraestructura
    - Organizacion de los recursos humanos
    - Control
  - Finalización del proyecto

Fasees adyacentes

  - Gestión de calidad
  - Gestión de cambios
  - Gestión de requisitos

Es decir, a grandes rasgos la gestión de proyectos se descompone en:

- Planificación del proyecto
  - Identificar los requisitos de trabajo
  - Cuantificar los requisitos de trabajo
  - Identificar los requisitos de recursos
- Seguimiento del proyecto
  - Realizar seguimiento del proyecto respecto al plan
  - Realizar los ajustes necesarios
  - Analizar el impacto

Una efectiva gestión del proyecto sucede si se produce:

- √ Una planificación disciplinada (primero en papel, luego en herramientas)
- √ Realización del trabajo de acuerdo con estándares preestablecidos
- √ Mediciones adecuadas y evaluación de los resultados
- √ Acciones correctivas adecuadas
- √ Liderazgo en los equipos

La máxima clave es que planificación y seguimiento no pueden consumir más esfuerzo que el propio proyecto.

4.3.3. Metodologías específicas

Por cada tipo de proyecto hay metodologías específicas (ver figura 4.4), y para cada planificación realizada hay que seguir siempre las mismas fases, independientemente de la metodología empleada.

Entre las características generales de la gestión de proyectos se encuentran objetivos, elementos y metodologías.

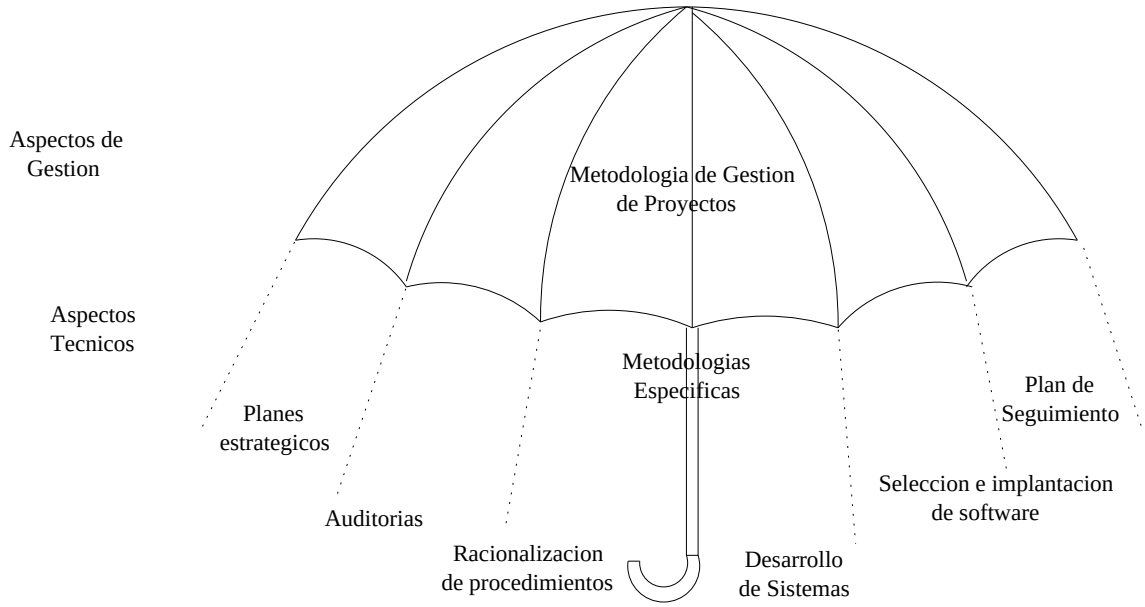


Figura 4.4: Metodologías de Gestión de Proyectos.

4.4. Conceptos de Planificación y Seguimiento de Proyectos

4.4.1. Técnicas de Gestión más usuales

Podemos clasificar las técnicas de gestión más usuales según la siguiente taxonomía:

- Técnicas de Representación
  - Grafo PERT[1]
  - Grafo Gantt
  - Histogramas
- Técnicas de Estructuración
  - Work Breakdown Structure (WBS)
  - Organization Breakdown Structure (OBS)
- Técnicas de Programación
  - CPM
  - PERT
  - Nivelación de recursos
- Técnicas (o métricas) de Seguimiento
  - Progreso
  - Esfuerzo
  - Coste
- Técnicas de Análisis de Riesgos
  - Montecarlo

4.4.2. Técnicas de Representación

Diagrama de Gantt

Es una representación simplificada de la red PERT, donde normalmente se obvian las relaciones de precedencias (cuando se incluyen se habla de *Diagrama de Gantt anotado*).

Consiste básicamente en representar en una escala de tiempos cada una de las actividades mediante barras (*barras de Gantt*, *barras de Gantt de seguimiento*), que representan su duración en "fechas reales".

[1] Técnicas de Revisión y Evaluación de Programaciones.

4.4.3. Técnicas de Estructuración

Work Breakdown Structure (WBS)

Técnica que consiste en estructurar las tareas de un proyecto por tipos, por niveles de agregación, por centros de costes,...

Es una estructura de descomposición/desglose de un proyecto en actividades y con diferentes niveles de detalle.

Organization Breakdown Structure (OBS)

Técnica que consiste en estructurar las tareas de un proyecto por las unidades, e incluso personas, que poseen responsabilidad sobre la realización de las mismas (estructura jerárquica típica de una empresa).

Refleja cómo están organizadas las diferentes áreas de una organización en términos de responsabilidad funcional.

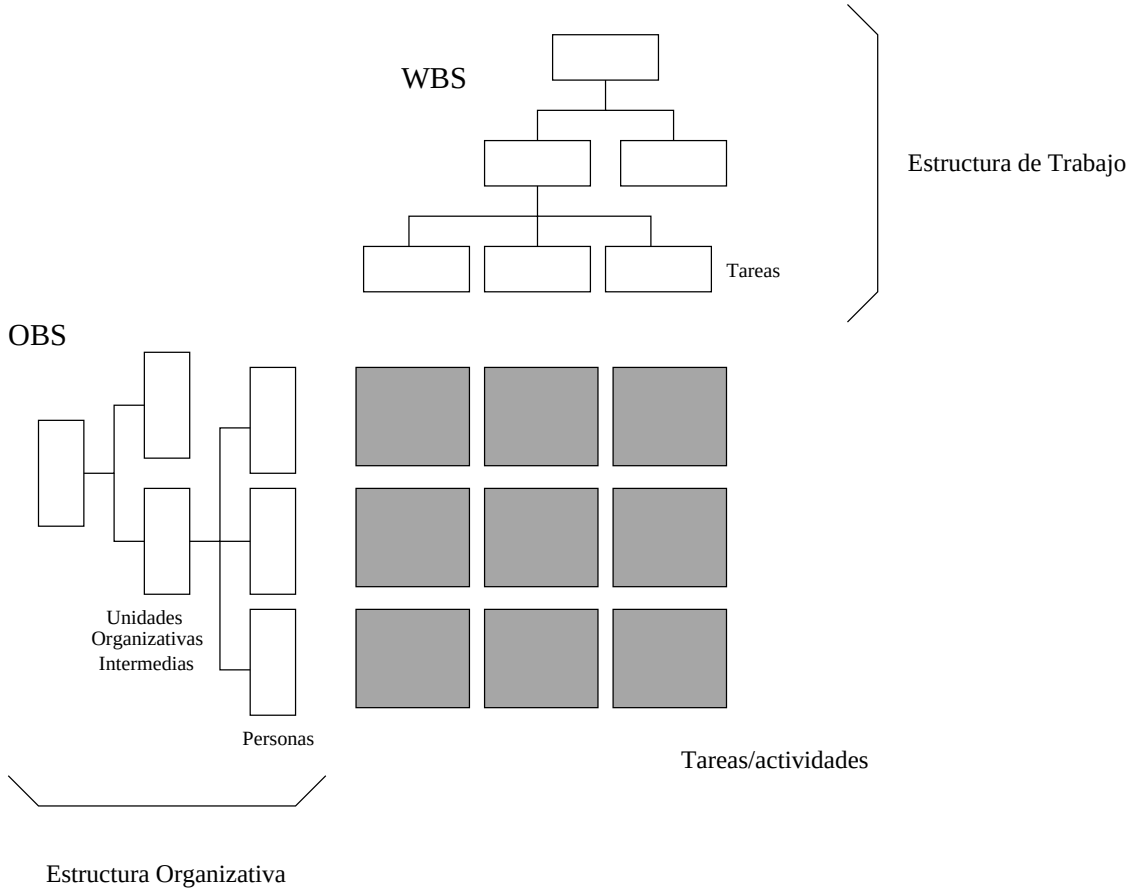


Figura 4.5: WBS y OBS.

Tareas/Actividades

Una tarea/actividad[2] es la unidad elemental del nivel de planificación.

Una tarea/actividad puede ser identificada por su duración (inicio+fin), consumo de recursos (asignación y dedicación) o ambas cosas.

Eventos (Milestones/Hitos)

El evento es un tipo de actividad que no tiene duración y sirve para indicar un punto particular en un proyecto.

Normalmente se utiliza para describir hitos o milestones (puntos de control), si no hay una oficina de control y gestión de proyectos, para imponer un seguimiento sencillo (o en caso de que el proyecto sea pequeño).

Suelen ser base de seguimiento y control especial. Aparecen con frecuencia ante eventos de terceras partes (subcontratistas).

Algunos expertos nunca los usan.

4.4.4. Técnicas de Programación

CPM

Permite obtener las fechas mínimas esperadas y las fechas máximas permitidas de comienzo y terminación de las actividades y tareas a partir de las duraciones e interdependencias de las mismas.

Al conjunto de actividades o tareas que poseen holgura total libre mínima (diferencia entre las fechas más tardías y las fechas más tempranas) se le denomina **camino crítico**.

El camino crítico nos interesa para prestar más atención, hacer un seguimiento más cuidadoso, asignar recursos de forma adecuada, etc.

PERT

PERT son las siglas de *Program Evaluation and Review Technique*.

Es una técnica similar al CPM donde las duraciones no son magnitudes deterministas, sino que son calculadas dentro de una probabilidad dada, a partir de las estimaciones optimistas, pesimistas y media de las mismas.

Se obtienen diferentes caminos críticos con una probabilidad asociada a cada uno.

Las herramientas suelen seguir CPM, no PERT.

Llamamos **grafo PERT** a una red que representa todas las actividades a realizar en el proyecto y sus interdependencias (restricciones lógicas). Existen dos notaciones:

- √ PDM: las actividades son nodos y las relaciones de interdependencia son vectores que las unen.

[2] Algunos autores y herramientas utilizan el término *actividad* para agrupar varias tareas.

√ ADM: los hitos/milestones son nodos y las actividades o tareas son vectores, cuyos módulos indican la desviación de las mismas.
El problema de esta notación es que obliga a que existan hitos, por lo que no suele emplearse (salvo en pequeños proyectos en los que nos podamos permitir seguir simplemente si se cumplen los hitos).

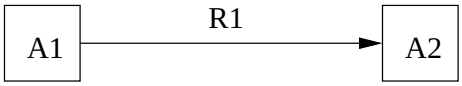


Figura 4.6: Notación PDM del grafo PERT.

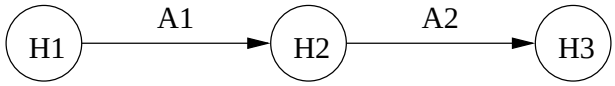


Figura 4.7: Notación ADM del grafo PERT.

Un elemento se considera como **recurso** si va a estar sujeto a una compartición, posiblemente originando conflictos de uso.

Existen varias clasificaciones a la hora de hablar de tipos de recursos:

Clasificación 1

- Recursos Humanos (individuos, grupos de recursos humanos homogéneos —igual nivel de eficiencia—).
- Materiales.
- Maquinaria.

Clasificación 2

- Consumibles (se emplean en la realización de la actividad y no se reutilizan —materiales—).
- Recurrentes (tienen capacidad de reutilización —personas, maquinaria—).

De ahora en adelante manejaremos los siguientes conceptos:

▷ Programación = Planificación + Asignación de Recursos

▷ Planificación = Establecimiento del Plan

▷ Calendario = Días naturales, días laborables (distinto en verano que en invierno)

▷ Duraciones = Día, semana

▷ Esfuerzo = Horas×Hombre, Número de Personas (depende de los recursos)

- ▷ Pool de RR.HH. = Identificador, nombre, OBS a la que pertenece, coste hora estándar, coste hora extra, disponibilidad por unidad de duración...
- ▷ Elementos de Coste Adicionales = Consumibles, materiales

Técnica CPM: Pasos

1. **Definición de actividades**
   - *a*) Definir las actividades de acuerdo a un estándar metodológico en función de la tipología del proyecto (desarrollo, estudio viabilidad, implantación,...).
   - *b*) Determinar las restricciones lógicas entre las actividades.
   - *c*) Asignar a cada actividad su duración en función del trabajo a realizar en cada una de ellas, así como la fecha de comienzo/fin objetivo (si las hubiese).
   - *d*) Obtener el camino crítico.
   - *e*) Afinar la planificación.

   La primera actividad *siempre* será planificar/gestionar/coordinar. Esta actividad "de acumulación de costes" también forma parte del proyecto, pero tiene la peculiaridad de que cuando éste empieza, ya ha acabado, de manera que no forma parte del camino crítico. Es por eso que se suele tratar como una "función normal" (sólo para justificar horas).
2. **Definición de restricciones lógicas**
   - *a*) Definir las interrelaciones entre las actividades de acuerdo con la tipología del proyecto y la metodología específica a utilizar (por ejemplo, en un proyecto de desarrollo en cascada, en espiral,...).
   - *b*) Tipos de restricciones:
     1) **Start to Start**
        La actividad B no *puede* comenzar hasta que la actividad A haya comenzado (tampoco obliga a que lo haga inmediatamente).

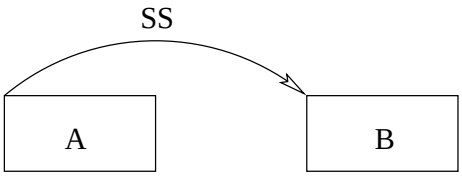


Figura 4.8: Restricción lógica *Start to Start*.

     2) **Start to Finish**
        La actividad B no puede terminar hasta que la actividad A haya comenzado. Es una restricción poco usual.

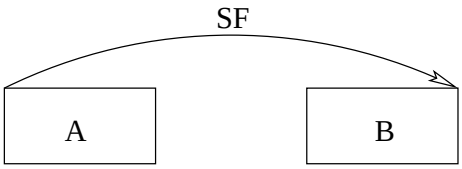


Figura 4.9: Restricción lógica *Start to Finish*.

3) **Finish to Start**
La actividad B no puede comenzar hasta que la actividad A haya terminado (aunque podría comenzar mucho después).

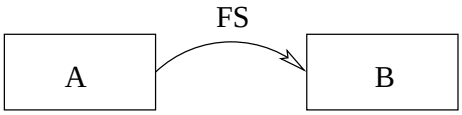


Figura 4.10: Restricción lógica *Finish to Start*.

4) **Finish to Finish**
La actividad B no puede terminar hasta que la actividad A haya terminado (lo que no quiere decir que tenga que acabar cuando termine ésta).

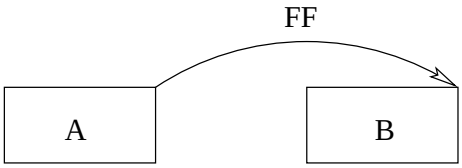


Figura 4.11: Restricción lógica *Finish to Finish*.

Se llama **tarea sucesora** (B) a aquélla cuyo comienzo o finalización depende de otra tarea, que se denomina a su vez **tarea antecesora** (A). Para saber qué va antes y qué va después, es necesario saber lo que se hace en cada actividad.

Hablamos de **demora positiva** cuando queremos indicar que desde el momento en que debería empezar la restricción ha de pasar cierto tiempo. También se maneja el concepto de **demora negativa**.

Una **hamaca** o **tarea resumen** es un tipo especial de actividad que mide el tiempo transcurrido entre dos puntos de la red (entre dos actividades, dos hitos, etc). Las relaciones lógicas se establecen siempre entre tareas elementales, nunca entre hamacas (no tiene sentido).
La duración de una hamaca no es una mera suma de tareas.

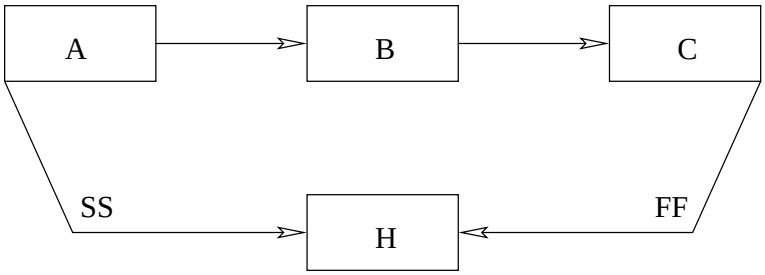


Figura 4.12: Esquema de una *hamaca*.

Técnica CPM: Cálculos

1. **Análisis de tiempos**
   Llamamos **camino crítico** al conjunto de actividades que determinan la duración total del proyecto. Todas las tareas del camino crítico deben necesariamente realizarse en las fechas previstas.
   Los datos necesarios para los cálculos son las duraciones estimadas de cada actividad y las restricciones lógicas entre las actividades (relaciones).
   El análisis de tiempos no tiene en cuenta los recursos necesarios ni su disponibilidad para el cálculo del camino crítico. Esto es así "técnicamente", pero no lógicamente, pues para el cálculo de duraciones estimadas se han tenido que tener en cuenta los recursos.

2. **Cálculo de fechas más tempranas (early)**
   Se ha de calcular la *fecha más temprana de comienzo de la actividad* (early start) y la *fecha más temprana de terminación de la actividad* (early finish).

3. **Cálculo de fechas más tardías (late)**
   Se ha de calcular la *fecha más tardía de comienzo de actividad* (late start) y la *fecha más tardía de terminación de la actividad* (late finish).

4. **Cálculo de la holgura total**
   Resulta de la diferencia entre la fecha más tardía de terminación y la fecha más temprana de finalización.
   Todas aquellas actividades que no tengan holgura (holgura de valor 0) forman parte del camino crítico.

5. **Método del camino crítico**
   Consta de dos pasos:

   a) Cálculo hacia delante: fechas tempranas (calculamos lo más pronto que podría estar terminado el proyecto). También se denomina cálculo desde el principio.

   b) Cálculo hacia atrás: fechas tardías (calculamos lo más tarde que podría empezar el proyecto). También se denomina cálculo desde el final.

Se obtienen de esta forma las actividades críticas vs. las actividades con holgura.

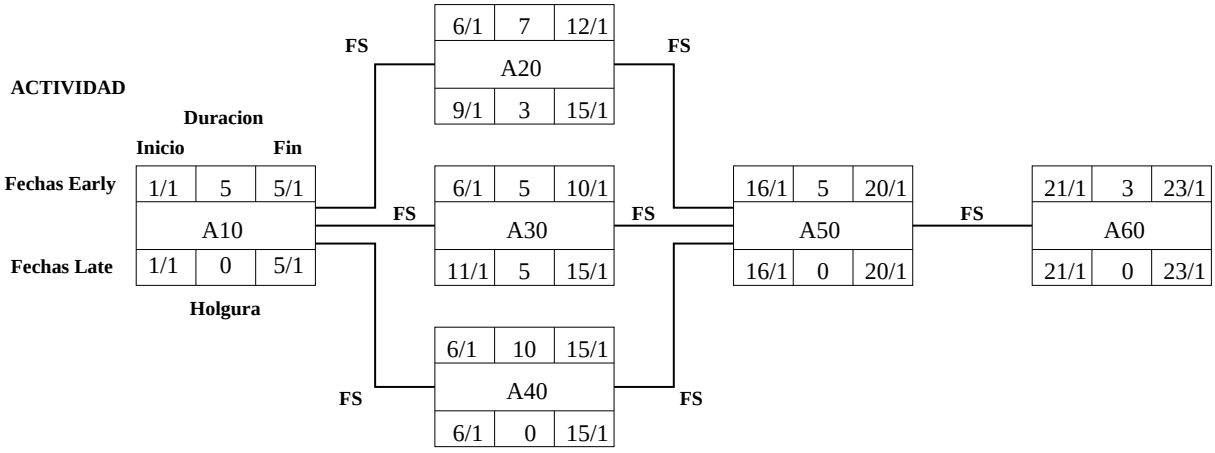


Figura 4.13: Gráfico de Red con cálculos asociados (PERT).

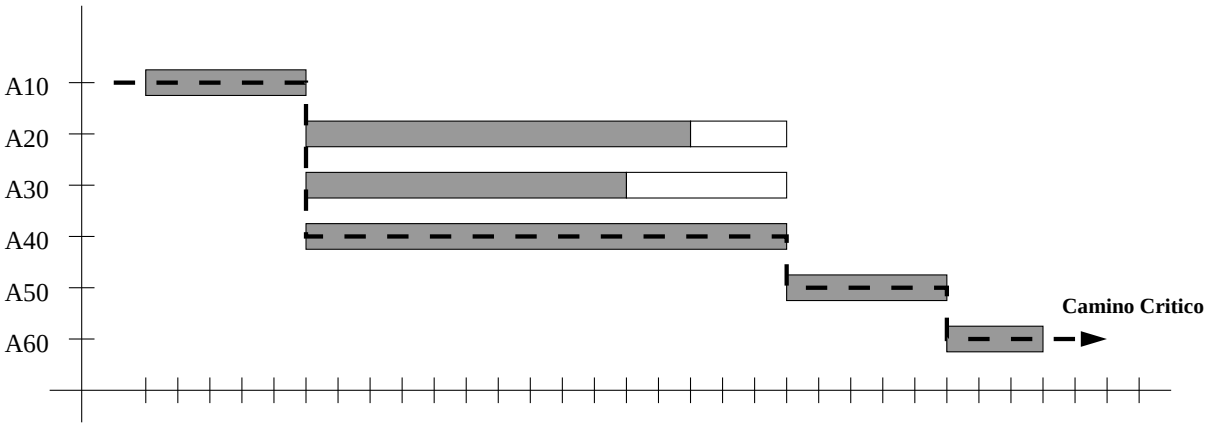


Figura 4.14: Gráfico de barras de Gantt.

Nivelación de recursos

Tiene lugar en dos pasos:

1. **Asignación de recursos**:
   - *a*) Definir qué perfiles se necesitan en cada actividad del proyecto.
   - *b*) Asignar recursos (individuos, materiales, perfiles) a cada actividad, en qué período se necesitan y cuánto esfuerzo se requerirá de ellos para la realización de cada una de ellas.
   - *c*) Nivel recursos para obtener las dedicaciones/cargas de los mismos y analizar posibles alargamientos de las actividades.

2. **Afinar la planificación**.

Los cálculos que se realizan para la nivelación de recursos son:

1. Ajustes en la planificación CPM en función de la disponibilidad de los recursos o de la máxima carga soportable en el tiempo.
2. El ajuste se realiza eliminando recursos de la realización de las actividades, con el consiguiente alargamiento en la duración de las mismas, por retraso en la realización de las mismas, por segmentación de las actividades, etc.

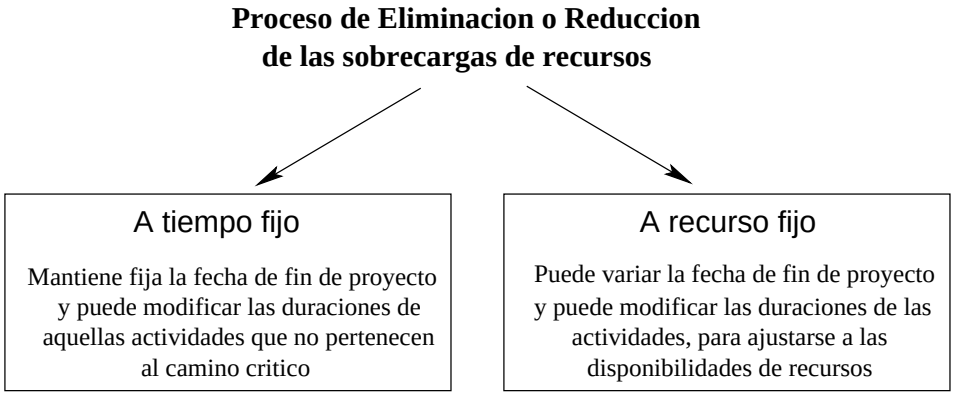


Figura 4.15: Formas de nivelar recursos en planificación.

4.4.5. Técnicas de Seguimiento

Los pasos más usuales a la hora de realizar el seguimiento del progreso de un proyecto son:

- **Establecer una línea base/original**
  - Es el proceso de almacenar los datos de análisis y nivelación que constituyen la información de comparación en sucesivos controles de avance o seguimiento del proyecto.
  - En el caso de que la planificación sea correcta se establecerá la línea base (foto fija de la planificación a efectos de comparación).
- **Progreso/Seguimiento del proyecto**
  - A nivel de actividad se indicará: fecha real de comienzo (si ha comenzado), fecha fin real (si ha finalizado) y duración pendiente a la fecha de progreso (time now) o porcentaje de completitud.
  - A nivel de cada actividad en que se trabaja y para cada recurso, se indicará el esfuerzo pendiente y los períodos en los que se requerirá, y el esfuerzo real.

- Se asignarán nuevos recursos necesarios y se eliminarán los no necesarios.
- Se analiza y nivela (a partir de la *baseline*) hasta tener una replanificación idónea (según la empresa, según lo que varíe el tiempo, coste y esfuerzo con respecto a lo previsto).
- Otra posible información de progreso es la referente a costes no correspondientes a recursos humanos (costes de informes técnicos, libros, licencias de software, etc.).

Es necesario definir una normativa estándar para el seguimiento de proyectos (mensualmente, trimestralmente, anualmente, al cambiar un proyecto significativamente).

Normalmente el esfuerzo real o dedicación a cada actividad de cada proyecto se implementa a través de hojas de trabajo con una periodicidad concreta (semanal, menual) para automatizar la recogida de datos y aumentar la fiabilidad de los mismos.

[Lord Kelvin]
Cuando puedas medir aquello sobre lo que hablas y expresarlo en números, sabes algo de ello; si no puedes medirlo, es decir, si no puedes expresarlo en números, tu conocimiento es escaso y no satisfactorio.

[Tom Demarco]
No puedes controlar lo que no puedes medir.

Métricas de Seguimiento

Hay fundamentalmente dos tipos de métricas de seguimiento:

De Producto
Son aquéllas que cuantifican algunos atributos del producto (tamaño, trazabilidad de los diseños, fiabilidad, complejidad...).

De Proceso
Son aquéllas que cuantifican algunos atributos del proceso de desarrollo, mantenimiento y de todo el entorno de desarrollo y mantenimiento (productividad de las herramientas, capacidad de las personas, ratio de defectos...).

La planificación y seguimiento de proyectos son dos áreas clave del nivel 2 de CMM. Este nivel sólo busca la repetitividad.

El objetivo es obtener medidas de los proyectos con procedimientos similares para conseguir la repetitividad ("Repetir los éxitos y evitar los fracasos").

¿Qué se quiere medir?

√ El progreso/seguimiento o cómo se van realizando las actividades conforme a la planificación de las mismas.

- √ El esfuerzo o cuántas unidades de dedicación se necesitan para la realización de las actividades.
- √ El coste o cuánto dinero necesitamos para obtener los productos/subproductos definidos.

El conocimiento se obtiene a través de un conjunto de métricas o medidas también llamadas **indicadores CMM**:

- ↪ Métricas que miden el grado de realización de las actividades de acuerdo con la planificación (o programación si se asignan recursos) de las mismas.
- ↪ La diferencia entre lo real y lo planificado es una indicación de la adherencia del proyecto al plan.
- ↪ Desviaciones significativas indican problemas.
- ↪ Se aplican a todas las etapas del ciclo de vida.
- ↪ La información de progreso/seguimiento es empleada por el jefe de proyecto y los diferentes niveles de la estructura de gestión de proyectos.

Métricas de Esfuerzo

- ↪ Métricas que miden las dedicaciones de los recursos humanos (horas×hombre, número de personas, etc.) para la realización de las actividades de los proyectos.
- ↪ La diferencia entre lo real y lo planificado es una indicación de la adherencia del proyecto al plan.
- ↪ Desviaciones significativas indican problemas.
- ↪ Se aplican a todas las etapas del ciclo de vida.
- ↪ La información de progreso/seguimiento es empleada por el jefe de proyecto y los diferentes niveles de la estructura de gestión de proyectos para seguir el consumo (gasto) real de dedicaciones de los recursos respecto al plan.

Una situación prolongada de muy pocos recursos trabajando en un proyecto puede ser el resultado de:

- ▷ Una sobreestimación del tamaño del software.
- ▷ Un progreso inadecuado.
- ▷ Un número elevado de conflictos abiertos.
- ▷ Incomprensión de los requisitos.
- ▷ Un equipo muy productivo.

- ▷ Un producto de mala calidad.
- ▷ etc.

Una situación prolongada de exceso de recursos trabajando en un proyecto puede ser el resultado de:

- ▷ Un problema más complejo de lo esperado.
- ▷ Recursos con insuficiente calidad para el trabajo a desempeñar.
- ▷ Un progreso inadecuado.
- ▷ Un número elevado de conflictos.
- ▷ Subestimación del tamaño del software.
- ▷ etc.

El jefe de proyecto debe utilizar los indicadores de una manera conjunta. Consideremos los siguientes casos:

- ○ Horas reales incurridas $>$ Horas planificadas $\Rightarrow$ se SE GASTA MÁS + Progreso real $>$ Progreso planificado $\Rightarrow$ se ACABA ANTES. Si continúa esta tendencia se acabará el proyecto antes de lo previsto y se excederá el esfuerzo de recursos humanos, es decir, se gastará más.
- ○ Horas reales incurridas $<$ Horas planificadas $\Rightarrow$ se SE GASTA MENOS + Progreso real $>$ Progreso planificado $\Rightarrow$ se ACABA ANTES. Si continúa esta tendencia se acabará el proyecto antes de lo previsto y se gastará menos en consumo de recursos humanos.
- ○ Horas reales incurridas $>$ Horas planificadas $\Rightarrow$ se SE GASTA MÁS + Progreso real $<$ Progreso planificado $\Rightarrow$ se AVANZA MENOS. Si continúa esta tendencia el proyecto acabará más tarde y se gastará más en consumo de recursos humanos.
- ○ Horas reales incurridas $<$ Horas planificadas $\Rightarrow$ se SE GASTA MENOS + Progreso real $<$ Progreso planificado $\Rightarrow$ se AVANZA MENOS. Si continúa esta tendencia el proyecto acabará más tarde y se gastará menos en consumo de recursos humanos.

Estas reglas son generales, pero dependen de cada proyecto.

Métricas de Coste

- ↪ Métricas que miden los costes reales del proyecto con respecto al plan.
- ↪ La diferencia entre lo real y lo planificado es una indicación de la adherencia del proyecto al plan.
- ↪ Desviaciones significativas indican problemas.
- ↪ Se aplican a todas las etapas del ciclo de vida.

↪ La información de progreso/seguimiento es empleada por el jefe de proyecto y los diferentes niveles de la estructura de gestión de proyectos para seguir el consumo (gasto) real en costes respecto al plan.

Los indicadores más comunes son:

- Variación de coste:

$$CV = BCWP - ACWP$$

donde

CV	Variación de coste
$BCWP$	Coste presupuestado del trabajo realizado
$ACWP$	Coste real del trabajo realizado

- Variación del progreso (en estimación con respecto al coste):

$$SV = BCWP - BCWS$$

donde

SV	Variación del progreso
$BCWP$	Coste presupuestado del trabajo realizado
$BCWS$	Coste presupuestado del trabajo planificado

- Coste estimado a la finalización:

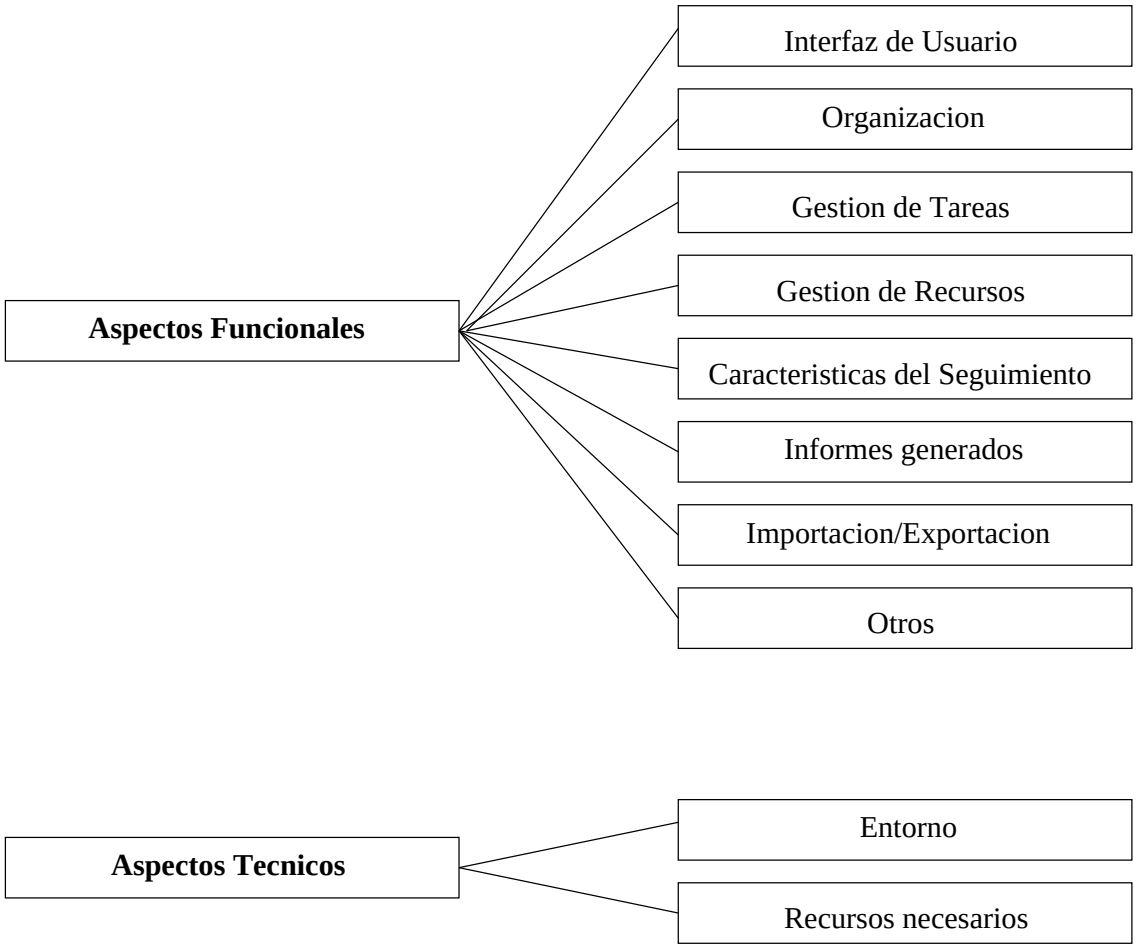
$$TCE = ACWP - ETC$$

donde

TCE	Coste estimado a la finalización
$ACWP$	Coste real del trabajo realizado
ETC	Coste estimado pendiente hasta la finalización

4.5. Arquitectura de un Sistema de Planificación de Proyectos

4.5.1. Aspectos a tener en cuenta en un Software de Gestión de Proyectos



Aspectos funcionales de la interfaz con el usuario

Prestación	Necesidad a cubrir
Diagrama de Gantt	Visualizar de una manera rápida la progresión de un proyecto o un conjunto de proyectos.
Grafo PERT (o de Red)	Visualizar de una manera rápida las interdependencias entre las diferentes actividades de un proyecto.
Histogramas	Visualizar de forma interactiva y gráfica las desviaciones en costes a través del tiempo, disponibilidad de recursos, etc.
Presentación	Interfaz con el usuario propiamente dicho.
WBS	Visualizar la estructura de descomposición de las actividades a realizar.
OBS	Visualizar la estructura organizativa del proyecto.

Aspectos funcionales de la organización del trabajo

Prestación	Necesidad a cubrir
Estructura de descomposición del trabajo	Posibilidad de estructuración del trabajo (WBS, OBS).
Número de códigos de descomposición	Posibilidad de varias estructuras a cubrir.
Subproyecto	Interdependencias entre las distintas actividades de los diferentes subproyectos.
Posibilidad de multiproyecto	Planificación de proyectos independientes, con la misma bolsa de recursos, posibilidad de respetar planificaciones anteriores o planificar todos en conjunto.

Aspectos funcionales de la gestión de tareas

Prestación	Necesidad a cubrir
Número máximo de tareas por proyecto	Complejidad del proyecto y detalle de las tareas a realizar en él.
Técnica de programación	Técnica que ayude a realizar una programación lo más detallada posible indicando holguras, tareas críticas,... (CPM, PERT, etc).
Duración de tareas	Diferentes medidas de tiempo, turnos.
Tipo de dependencias	FS, FF, SF, SS.
Prioridades de tareas	Asignación prioritaria de recursos a tareas en paralelo.
Calendarios	Aproximación total o parcial a las fechas reales.
Nivelación automática de recursos	Histograma de ayuda para ajustar la programación CPM en función de los recursos.
Histograma de recursos	Visualizar la distribución de la carga del proyecto en función del tiempo.
Nivelación de recursos entre proyectos	Distribución de recursos entre proyectos en base a su disponibilidad u otras razones.

Aspectos funcionales de la gestión de recursos

Prestación	Necesidad a cubrir
Número máximo de recursos por proyecto	Volumen de recursos a gestionar.
Número máximo de recursos por tarea	Volumen de recursos a gestionar y detalle de la estructuración de recursos.
Asignación parcial de recursos	Dedicación discontinua de un recurso a una actividad.
Tipos de recursos	Seguimiento detallado del coste de un proyecto en recursos.

Aspectos funcionales de seguimiento

Prestación	Necesidad a cubrir
Real frente a planificado	Análisis de las desviaciones.
Tanto por ciento realizado	Porcentaje completado.
Costes	Situación real frente a prevista.
Técnica de valor ganado	

Aspectos funcionales de informes generados

Prestación	Necesidad a cubrir
Diferentes tipos de informes	Con distintas finalidades.
Personalización	Poder indicar los aspectos más relevantes que nos interesan en cada caso particular.
Generador	Automático.
Gráficos	Para apreciar mejor visualmente los resultados.

Aspectos funcionales de importación/exportación

Prestación	Necesidad a cubrir
Importación	Poder incluir datos provenientes de otras fuentes.
Exportación	A distintos formatos portables.

Otros aspectos funcionales

Prestación	Necesidad a cubrir
Cálculo de riesgos	Herramientas de ayuda al análisis de riesgos.
Simulación	Previsión de resultados a partir de eventos distintos.
Macros	Posibilidad de estandarizar procesos.
Acceso a BBDD corporativas	Posibilidad de acceder a datos corporativos desde el propio sistema.

Aspectos técnicos de los recursos necesarios

Prestación	Necesidad a cubrir
Entorno hardware	Mainframe, LAN (distribuido),...
Software de base	Sistema operativo, gestor de BD, software de gráficos, sistema de proceso de textos, etc.

Aspectos técnicos del entorno

El entorno organizativo debe ser el objeto de un *Manual de Organización de Proyectos*, conteniendo:

- Estructura de órganos y responsabilidades.

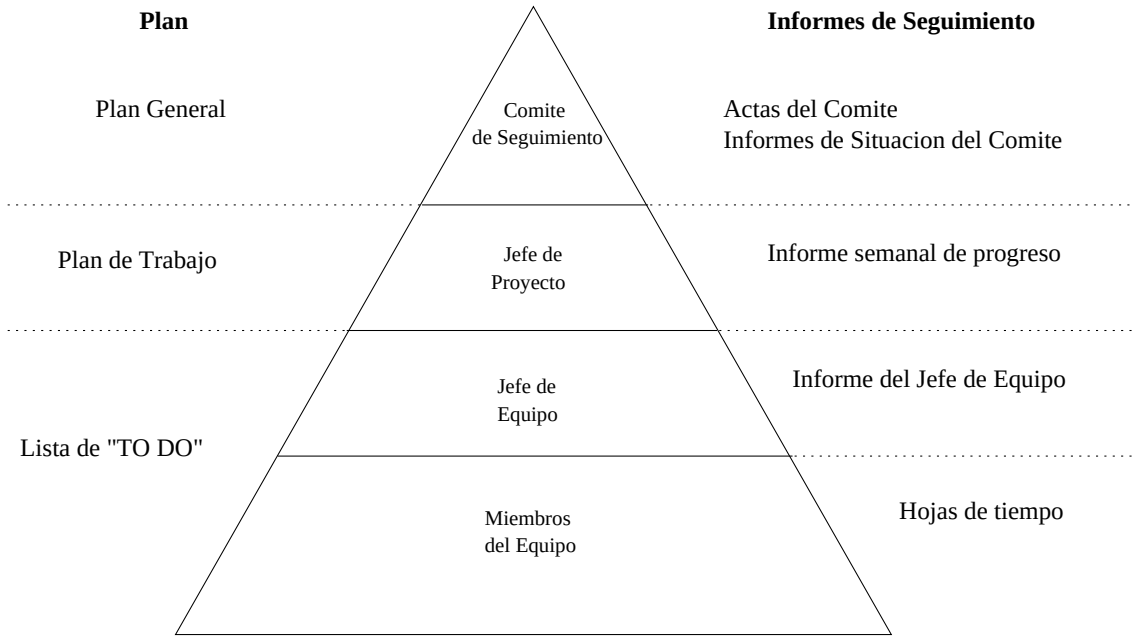


Figura 4.16: Estructura de un entorno organizativo clásico.

- Unas normas y políticas de actuación.
- Conjunto de procedimientos administrativos (hojas de tiempo, distribución de informes,...).
- El escenario organizativo del sistema de software (herramientas de planificación[3]).

La implantación o simplemente la elección de una herramienta de planificación (herramienta CASE), puede afrontarse en realidad como un proyecto más.

[3]Por ejemplo, según las normas ISO, en el manual de calidad se indicaría cómo hacerlo y quizás, en un apéndice, cómo se hace en la herramienta escogida o en uso actualmente.

Capítulo 5

Gestión de Riesgos

La **Gestión de Riesgos** no es una actividad que deba realizarse desde el principio de un proyecto, sino que se introduce cuando, tras haber realizado gestión de proyectos, éste se halle asentado[1].

5.1. Introducción a la Gestión de Riesgos

Los **riesgos** son algo inevitable, son inherentes a los proyectos y siempre existirán, pero con algunos podremos vivir y frente a otros tendremos que ejercer acciones para controlarlos y/o evitarlos.

Un **riesgo** es un evento que podría aminorar la capacidad para conseguir los objetivos definidos, es decir, cualquier cosa que pueda impedir que se cumpla lo previsto.

¿Por qué debemos tratar los riesgos? Hay que reconocer que donde hay una oportunidad hay beneficios, y donde hay beneficios hay riesgos. Para lograr los objetivos/beneficios marcados hay que controlar esos riesgos.

Las dos alternativas son:

1. **Gestión de problemas** (actuar de "bombero")
2. **Gestión de riesgos**

5.2. Fases de la Gestión de Riesgos

La **Gestión de Riesgos** se organiza en cuatro fases diferenciadas:

1. Identificación

- Enumeración de los riesgos para el proyecto.
- Clasificación y agrupación (por ejemplo, por causas).

[1] En un proyecto de 300 días×hombre se identificarán 5-6 ó 7 riesgos.

U1	Gestion de Problemas (bombero)	*Nos olvidamos de los riesgos y si aparecen (problemas) se intentan solucionar entonces.*
U2	Riesgos poco importantes	*Contingencia: Atendemos el riesgo cuando se presenta, pero tenemos las actividades planificadas (previstas).*
U3	Riesgos mas importantes	*Actividades preventivas: identificar alternativas y acciones de contingencia*
U4	Riesgos muy importantes	*Contingencia y mitigacion*

Figura 5.1: Umbrales de Gestión de Riesgos.

2. Valoración

Priorizar riesgos y umbrales de actuación.

- Cuantificación de los riesgos.
- Decisión del nivel aceptable de un riesgo.
- Priorización de los riesgos.

3. Análisis

Definir qué se va a hacer en cada umbral.

- Estudio de alternativas.
- Contención, mitigación, prevención.

4. Control y Seguimiento

- Implantación de estrategias de mitigación.
- Seguimiento de los riesgos clave.
- Labor de contingencia.

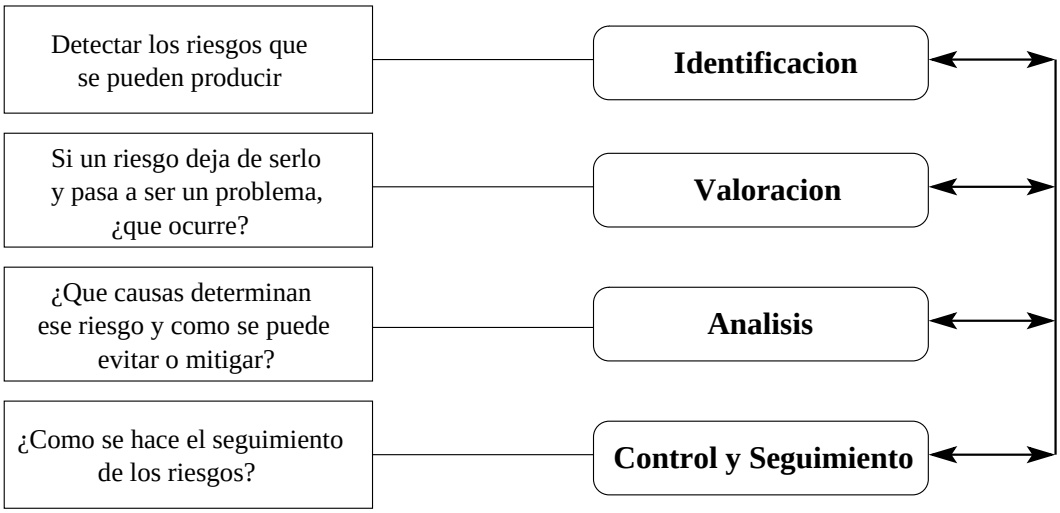


Figura 5.2: Fases de la Gestión de Riesgos.

5.2.1. Clasificación de riesgos

Llamamos **riesgo en el proyecto** a la incertidumbre que afecta a un proyecto por problemas técnicos, de coste y planificación, de recursos, etc.

Llamamos **riesgo en el negocio** a la incertidumbre con el que los sistemas de información proporcionarán los resultados esperados para el negocio, así como a la posibilidad de amenazas y vulnerabilidades a interferencias hostiles (a veces no intencionadas).

Llamamos **riesgo en Gestión de Riesgos** a la incertidumbre en la práctica de Gestión de Riesgos en la Gestión de Proyectos.

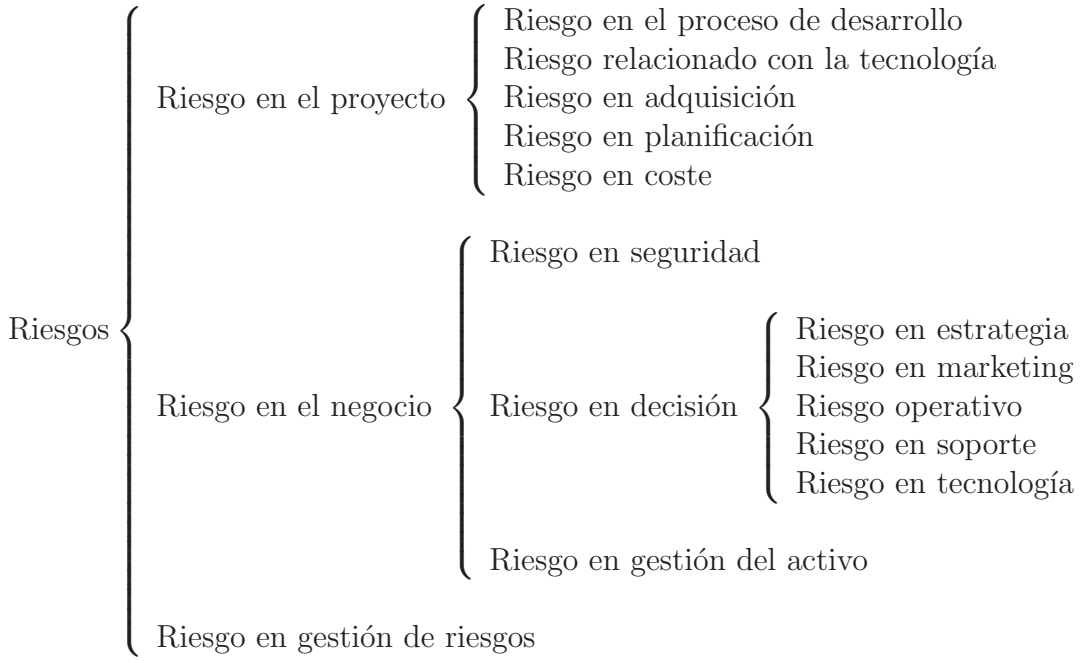


Figura 5.3: Marco de Gestión de Riesgos.

5.2.2. Primera fase: Identificación de riesgos

En la fase de identificación debemos:

- Definir los *riesgos generales* tomando el proyecto como un todo (¿cuál es nuestro enemigo?).
- Subdividir la situación de riesgo en un conjunto de productos o tareas principales.
- Estudiar cada producto o tarea principal definiendo *riesgos específicos*.

Pero ¿cómo se identifican los riesgos? El riesgo es a menudo definido como la representación de un resultado insatisfactorio. Pensar en estos términos es útil para ayudar a identificar riesgos potenciales.

Identificar resultados insatisfactorios permitirá identificar las directrices clave para lograr la identificación final de los riesgos.

La mayoría de los riesgos tienen una o más causas origen, por ejemplo:

- Pobre gestión de requisitos.
- Falta de la definición de un ciclo de vida.
- Pobre planificación y seguimiento.
- Malas relaciones entre el personal.
- Mala gestión de las relaciones interpersonales.
- Mala gestión en las compras.
- Tecnología inmadura y/o desconocida.
- etc.

El proceso de búsqueda de las causas origen puede ser considerado en cada área presentada con anterioridad: riesgos en el proyecto, riesgos en el negocio,... Esto proporciona un modo sistemático de presentar los riesgos.

Para proyectos que no sean muy pequeños, se puede examinar individualmente cada tarea principal del plan de proyecto y cada área de producto, identificando aproximadamente 6 riesgos principales por área, para evitar complejidad en el proceso. Así se va confeccionando una lista que se debe evitar que sea una enumeración exhaustiva de pesimismos.

Evento vs. Causa

Otra posibilidad de identificar riesgos es a través del estudio de eventos y sus resultados:

¿Que lo ha disparado? *¿Que produce el evento al darse?*

De manera que tenemos cuatro aproximaciones de identificación de riesgos:

1. Examinar posibles resultados insatisfactorios y sus causas de origen.
2. Usar el marco definido para identificar riesgos.
3. Utilizar el resultado del plan (WBS) o las tareas analizadas para identificar sus riesgos.
4. Estudiar los eventos que se pueden dar y sus efectos (atacar la causa si estamos en el tercer umbral de riesgos).

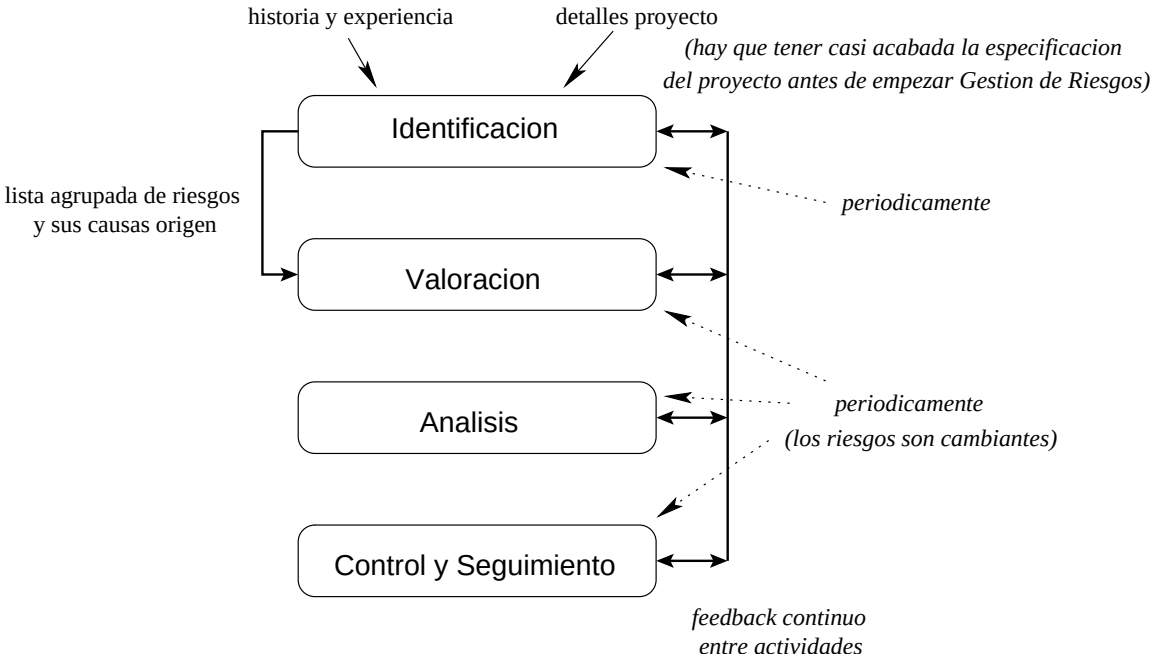
Una vez identificados, los riesgos podrían clasificarse de acuerdo a las causas origen que los provocan. Esto ayuda en la planificación de Gestión de Riesgos a determinar las causas que deben ser controladas: la agrupación ayuda a saber sobre qué hay que poner especial atención (actuaremos sobre la causa, no sobre la consecuencia).

Existe la necesidad de recoger datos de identificación, valoración y control de riesgos. Tener una base de datos de información de riesgos —tipos de proyecto frente a esfuerzo estimado o resultante— (y otra de medidas de desarrollo de software) es una práctica clave en la gestión de la capacidad de desarrollo.

La segmentación de la tipología de proyectos se hace en base al tipo de proyecto, su contexto y tres valores: *exposición*, *probabilidad* e *impacto*:

$$ER = PR \times IR$$

Puede utilizarse para mejorar la ayuda en la identificación de riesgos conocidos.



De cara a la siguiente fase, ya sabemos algunas formas de "identificar al enemigo". Una vez identificado, hay que atender a los temas que pudieran tener el mayor potencial para dañar nuestras intenciones. Ése es el objetivo: valorar los riesgos (¿cuánto daño nos puede hacer el enemigo? ¿cómo es de grave?) y cuantificarlos de alguna manera (ya veremos que a veces cualitativa y otras cuantitativamente).

5.2.3. Segunda fase: Valoración de riesgos

Una vez se han identificado los riesgos en la fase anterior, es fundamental su priorización, que se hará en esta fase. Para llevar a cabo dicha priorización es necesario un método de valoración o cuantificación (que no tiene por qué ser numérico).

Una vez cuantificado un riesgo, es comparable con otros para estudiar la prioridad que posee dentro de todo el proyecto.

Se ha definido *riesgo* como la probabilidad de obtener un resultado no satisfactorio. La **exposición** a un riesgo se define como el producto de la **probabilidad** de que ese riesgo ocurra multiplicada por el **impacto** (en tiempo, esfuerzo, en coste o en los tres) que puede producir la ocurrencia del riesgo:

$$\mathbf{ER} = \mathbf{PR} \times \mathbf{IR}$$

La *probabilidad* de un riesgo puede valorarse como una medida matemática ($n \in [0,1]$[2]) o bien como una medida subjetiva (alta, media, baja[3]).

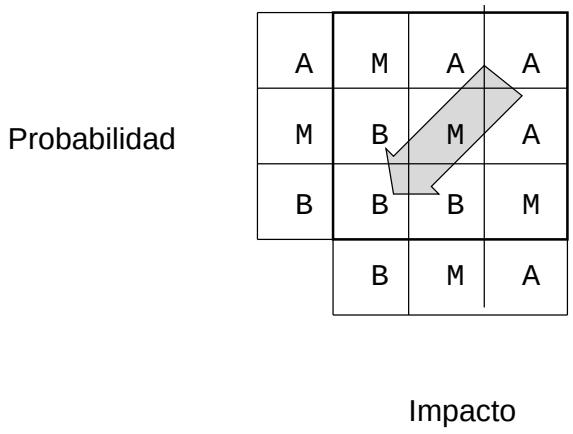


Figura 5.4: Tabla de análisis de exposición a riesgos.

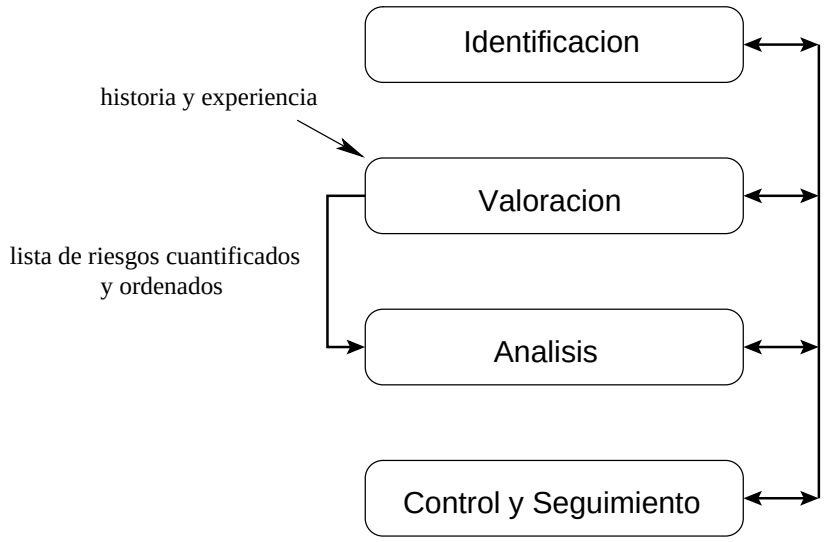
Con este método (tan) subjetivo (tabla 5.4) se pueden definir los *umbrales* de forma sencilla, por ejemplo: aquellos riesgos cuya exposición sea B (baja) se vive con ellos (gestión de problemas), para los que tengan nivel M (medio) se de define un plan de contingencia (pasos a seguir si se dan) y para los A (alta) se definirán acciones de prevención, alternativas y también un plan de contingencia.

En cuanto a la cuantificación del impacto, si el riesgo es económico el impacto puede ser la pérdida expresada en unidades matemáticas (€); si el riesgo afecta a la planificación puede expresarse en unidades temporales (días o semanas) y si el riesgo afecta a la vida, deberán contemplarse aspectos como la viabilidad legal, la gravedad de las incapacidades, el número de personas afectadas,... Una tarea mucho más difícil es decir cómo afecta un riesgo al esfuerzo (esfuerzo que hay que emplear para subsanarlo).

[2]En realidad, $n \in (0,1)$, pues un valor 0 implica que realmente no hay riesgo y un valor 1 es certeza.
[3]En caso de utilizar una medida subjetiva es mejor que sea cualitativa y no cuantitativa.

Ya hemos visto que los riesgos pueden presentarse como valores ($n \in [0, 1]$). Normalmente se necesitan datos históricos para proporcionar un valor matemático de este tipo (hay todo un rango) para la probabilidad de la ocurrencia de un riesgo.

La exposición al riesgo permite priorizar los riesgos identificados y extraer un conjunto de riesgos clave de la lista completa de riesgos. Se pueden definir niveles de aceptabilidad de los riesgos para dirigir las medidas a poner en práctica. De esta forma, se proporciona una actuación dirigida a los riesgos importantes, aunque está claro que no se debe actuar a rajatabla (puede haber casos especiales...).



5.2.4. Tercera fase: Análisis de riesgos

Esta fase involucra aspectos como:

- Estudio de alternativas.
- Definición de estrategias de mitigación.
- Planificación del control del riesgo.

Existen tres posibles clasificaciones para un riesgo desde el punto de vista del análisis:

Gestionable y en el alcance: Riesgo que requiere acciones para gestionarlo, pero sus costes e impacto están dentro de lo asumible en el proyecto.

Gestionable pero fuera del alcance: Riesgo cuyas acciones requeridas tienen en sí mismas un serio impacto en el coste, planificación, etc. del proyecto.

Inevitable: Riesgo cuyo tratamiento requiere un cambio fundamental de la naturaleza del proyecto.

Asimismo, hay cuatro estrategias básicas de Gestión de Riesgos:

Prevenir: Desarrollar el proyecto de forma que el riesgo no pueda progresar. Considerar caminos alternativos.

Controlar: Aceptar el riesgo, planificar las acciones de contención o contingencia, y hacer seguimiento frecuente del riesgo.

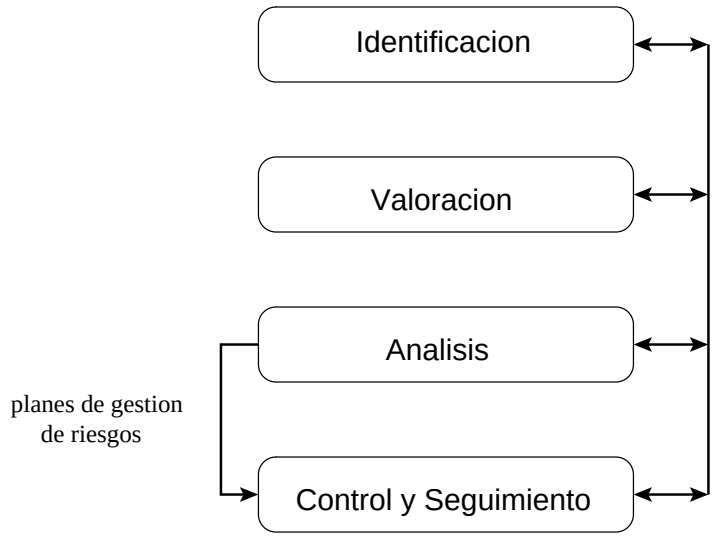
Transferir: Sacar el riesgo fuera: subcontratistas.

Asumir: Actuar de "bombero".

Una vez identificados los riesgos, valorados y elegido un plan para su gestión, los riesgos clave (exposición alta) deberían ser acometidos. Se recomienda crear un plan para cada riesgo clave (Planificación de Gestión de Riesgos). Después, los planes individuales deben ser integrados para el empleo óptimo de recursos.

Se recomienda seguir para cada plan individual de Gestión de Riesgos un estándar de planificación, que debería incluir[4]:

- √ Objetivos
- √ Entregables e hitos
- √ Responsabilidades (¡muy importante!)
- √ Estimación
- √ Costes (recursos)



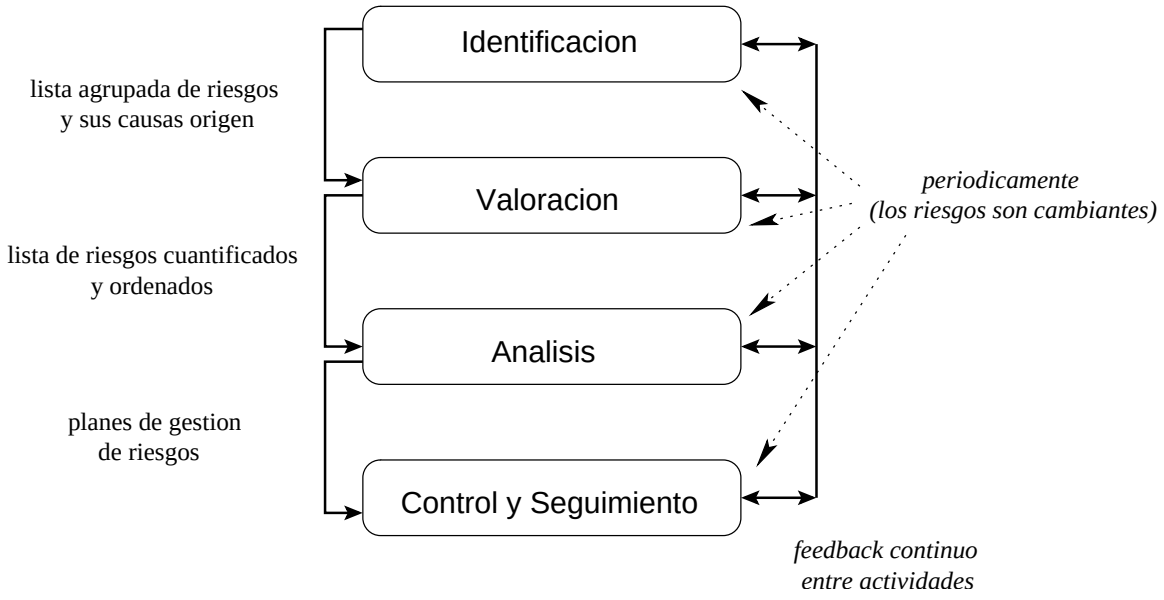
[4]Similar a la hoja de cálculo que mencionábamos para planificación de proyectos de menos de 20 días×hombre.

5.2.5. Última fase: Control y Seguimiento de riesgos

Las actividades de **control y seguimiento** se centran en asegurar que los planes de gestión de los riesgos se llevan a cabo correctamente. Verifican que los indicadores de los riesgos (por ejemplo, la exposición) no sufren variaciones (cambio en el valor de la exposición). Para ello hay que llevar el seguimiento y recoger las variaciones en el estado del riesgo, documentando los cambios (que se incluirán en el informe de seguimiento del proyecto).

Es necesario determinar los puntos de chequeo en los que los riesgos serán revisados y controlados (normalmente, cuando se haga también revisión/seguimiento del proyecto). Esto depende del ciclo de vida adoptado.

Como regla general, pues, se hará seguimiento de riesgos cada vez que se haga el informe de seguimiento de un proyecto, tal y como se habrá indicado en su plan de proyecto.



Capítulo 6

Gestión de la Configuración del Software

La integridad de un producto software depende de la acción combinada de dos cosas: su desarrollo y su gestión y control (y los criterios de calidad aplicados durante las mismas).

El objetivo de la **Gestión de la Configuración del Software** (en adelante, GCS) o *Software Configuration Management* (SCM) es lograr, mantener y asegurar la integridad del software, evaluando y controlando los cambios que afecten al producto software. Es decir, es algo más que controlar el código y sus versiones: no sólo se va a controlar lo que se genera, sino lo que se usa para generarlo. Además, es algo delicado, pues los errores debidos a la GCS son complicados de detectar y carísimos.

Por tanto, la GCS se encuadra en las actividades de control relativas a un proyecto.

6.1. Conceptos básicos de la GCS

6.1.1. ¿Qué es la GCS?

Según Babich, la Gestión de la Configuración del Software es "*el arte*[1] *de coordinar el desarrollo de software para minimizar la confusión. El arte de identificar, organizar y controlar las modificaciones que sigue el software que construye un equipo de programación*". Y su objetivo es "*maximizar la productividad minimizando los errores*".

La GCS se realiza, por tanto, durante todo el ciclo de vida del producto y su misión se resume en:

- minimizar la confusión
- minimizar los errores
- maximizar la productividad

[1] Aunque nosotros lo veremos desde una perspectiva ingenieril.

La definición de la IEEE es algo más formal: "*La GCS cubre todas las actividades utilizadas para identificar y definir los* ***elementos de configuración***[2]*, así como sus relaciones; permite controlar cambios y modificaciones durante todo el ciclo de vida, conociendo los sucesivos estados del software que se archivan (o producen) y la verificación de la completitud y consistencia de cada uno de estos estados*".

Es decir, según IEEE, la GCS es algo similar a una máquina de estados, que durante todo el ciclo de vida:

- identifica elementos de configuración
- identifica componentes de elementos
- identifica relaciones

Los ECS son todo aquello que se usa para generar todo lo que ha de ser controlado: código fuente, código objeto, código ejecutable, documentos técnicos de desarrollo, herramientas CASE, compiladores, prototipos, manuales de usuario, planes de prueba, especificación del sistema, plan del sistema, plan del proyecto, diseño preliminar, diseño detallado, resultados de los planes de prueba, plan de calidad, productos hardware y software usados, diseño y contenidos de las bases de datos...

6.1.2. Objetivos de la GCS

Son fundamentalmente dos:

1. **Facilitar la visibilidad:**
   - √ sobre el estado del producto: ESTADO
   - √ sobre su historia: EVOLUCIÓN[3]
2. **Mantener la integridad del producto:**
   - √ establecer y mantener la integridad de los productos generados durante un proyecto y a lo largo de todo su ciclo de vida (incluye, por tanto, mantenimiento)

¿Qué significa **integridad** de un producto (IPS)? ¿Cuándo diríamos que un producto es *íntegro*? Se considera que debe cumplir tres requisitos:

1. Satisfacer los requisitos o necesidades del usuario, pues en otro caso al usuario no le sirve. Esto incluye tanto requisitos *explícitos* como *implícitos* (desde que no falle hasta que tenga ventanas bonitas).
2. Cumplir los requisitos de rendimiento.
3. Ser traceable[4], es decir, debe poderse trazar toda su evolución desde que empieza a concebirse hasta el final, a través de todas las fases de su ciclo de vida.

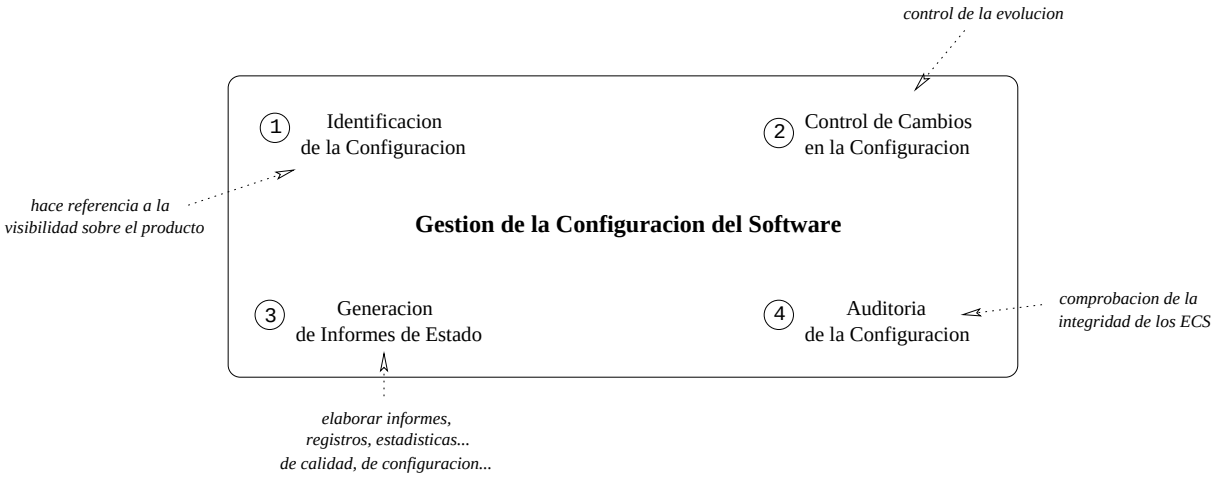


Figura 6.1: Etapas de la Gestión de la Configuración del Software.

Esto se consigue a través de la GCS:

GCS-1: Identificación de la Configuración
Identificar la estructura (ECS) del producto, sus componentes, y los componentes de cada componente, asignándoles un código, haciéndolos visibles e identificando cómo se agrupan para formar otro tipo de componentes.

GCS-2: Control de Cambios en la Configuración
Controlar las versiones y entregas del producto (software) y cualquier cambio que se produzca a lo largo de todo su ciclo de vida (y, por tanto, sobre cualquier entrega ya realizada).

GCS-3: Generación de Informes de Estado
Informar acerca del estado de los componentes de un producto (ECS), cambios y/o solicitudes de la GCS, etc.

GCS-4: Auditoría de la Configuración
Validar la completitud y consistencia de y entre los componentes de un producto, asegurando que éste es lo que el usuario quiere.

Así pues, entenderemos por **configuración del software** el conjunto de todos los ECS de un proyecto, el conjunto de toda la información y productos utilizados o generados como resultado del proceso de Ingeniería del Software.

Por su parte, entenderemos por **elemento de configuración del software** (ECS) cada uno de los componentes de la configuración del software. Los ECS son la unidad de trabajo para la GCS (por separado, tienen entidad por sí mismos).

La configuración del software genera todos los ECS, que deben ser definidos para cada proyecto.

[2]En adelante, ECS.
[3]Se evalúan y controlan los cambios.
[4]Recordemos que la trazabilidad era un requisito ISO.

6.1.3. Línea base

El concepto de **línea base** o **baseline** se usa para facilitar el control de cambios. Se puede definir:

↪ Desde el punto de vista del proceso,

> Es un *punto de referencia* en el proceso de desarrollo que queda marcado por la aprobación de uno o más ECS, mediante una revisión técnica formal.

↪ Desde el punto de vista del producto,

> Es un conjunto de ECS revisados y aceptados (aprobados) que sirven como base para el desarrollo posterior, y sólo pueden cambiarse a través de un proceso formal de control de cambios.

A medida que avancemos en el desarrollo del producto aumentará el número de ECS. La línea base sirve para evitar los cambios indiscriminados, controla los cambios sin impedir los justificados.

6.1.4. Versiones, revisiones, variantes y releases

Una **versión** es una instancia de un ECS en un momento dado del proceso de desarrollo, que es almacenada en una biblioteca o repositorio y que puede ser recuperada en cualquier momento para su uso o modificación.

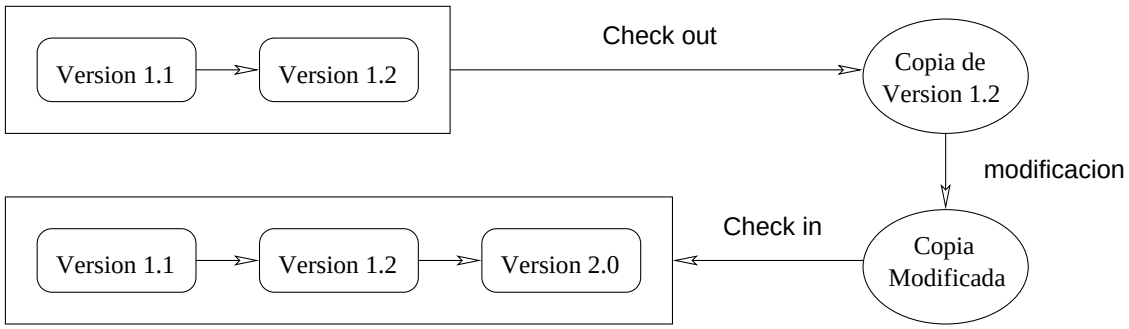
A las distintas *versiones* que aparecen en el tiempo, según se va avanzando en el desarrollo de un elemento, se les suele llamar también **revisiones**[5].

Cada una de las versiones/revisiones de un ECS se debe poder identificar de manera única, siendo común utilizar un esquema numérico, donde cada nueva versión recibe un número sucesivo.

Se suele utilizar un **grafo de evolución** para seguir la traza de las revisiones: es una representación para las distintas versiones de un elemento y sus relaciones de sucesión. La tónica general es tomar la última versión y modificarla, exigiendo que todas las versiones, actuales y anteriores, estén disponibles.

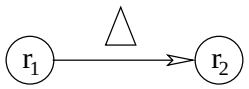
El que elabora el elemento es el que decide cuándo se hace una nueva versión. Se tiene que hacer cuando el elemento cambia significativamente (por ejemplo, modificación de diagramas de análisis).

[5]En general, se suele usar el término *versión* cuando el cambio frente a la anterior es grande, y *revisión* cuando es de menor categoría, aunque frecuentemente se utilizar el término *versión* para todo, por extensión.



Las herramientas de control de versiones, por eficiencia (y otras razones), no almacenan físicamente todas las revisiones, sino sólo una de ellas, que puede ser la primera o la última. Sin embargo, nos permiten recuperar cualquier otra versión (trazabilidad). Para ello guardan también toda la historia de cambios que han ocurrido sobre el elemento y que lo han hecho pasar de una versión a otra.

Siendo r_1 y r_2 dos revisiones consecutivas en el grafo de evolución, llamamos **delta** a una secuencia de operaciones que, aplicadas sobre la revisión r_1, dan como resultado la revisión r_2.



Existen varios tipos de *deltas*:

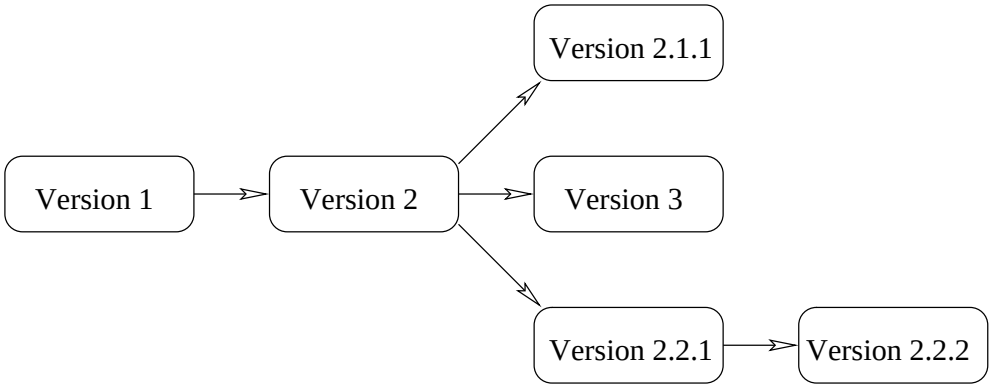
- ▷ Según su dirección,
  - ○ Deltas directos: almacenan los cambios que hacen pasar de una versión a la siguiente (se almacena la primera revisión).
  - ○ Deltas indirectos o inversos: almacenan los cambios que hacen pasar de una versión a la anterior (almacenan la última revisión)[6].
- ▷ Según su localización de almacenamiento:
  - ○ Deltas mezclados: se manejan las operaciones en ficheros propios del ECS (Δ y ECS se almacenan en el mismo fichero).
  - ○ Deltas separados: Δ y ECS están en documentos aparte.

Las **variantes** son versiones (o revisiones) de un ECS que coexisten en un determinado momento y que se diferencian entre sí en ciertas características.

Representan la necesidad de que un objeto satisfaga distintos requisitos al mismo tiempo. Puede haber varias variantes sobre las que se esté trabajando simultáneamente, a diferencia de las revisiones (que sólo se trabaja con la última).

Una variante no reemplaza a otra, como ocurre con las revisiones, sino que abre un nuevo camino de desarrollo. Las variantes se reconocen fácilmente en el grafo de evolución como una ramificación de éste:

[6]Está claro que son más rápidos, pero también son más complejos.



Las variantes pueden ser:

- Temporales: se acabarán mezclando con otra variante en algún momento del desarrollo.

  Es necesario que varias personas trabajen simultáneamente sobre la misma versión de un objeto, y para que no ocurran conflictos entre ellos, se crea una variante para cada persona, para que puedan trabajar en paralelo.
  El momento de la reunión es delicado, por ello sólo deben usarse cuando sean imprescindibles y hacer que sean lo más cortas en el tiempo posible.

- De usar y tirar:

  Se usan para explotar distintas soluciones alternativas y quedarnos con la mejor. Un caso típico son las variantes de pruebas, que son aquéllas sobre las que se introducen elementos especiales para facilitarnos las pruebas.

- Permanentes:

  - Variantes de requisitos de usuario: el caso más típico es el idioma en las aplicaciones.
  - Variantes de plataforma: una variante por cada sistema operativo o plataforma hardware sobre la que deseemos que funcione nuestra aplicación.

Al abrir ramificaciones en el grafo de evolución, en vez de una única configuración, vamos a tener un conjunto de *configuraciones alternativas*. Cada una va a satisfacer las necesidades de un entorno particular o usuario.

Cada configuración alternativa se especifica mediante los ECS que la componen y la versión adecuada de cada uno de ellos. Esto se puede conseguir de la siguiente forma:

√ Se asocian atributos a cada versión de un ECS.

√ Se crea una especificación de configuración que describe el conjunto de atributos deseados[7] y se recuperan los ECS adecuados para construir la configuración.

Una **release** es una configuración del sistema que se va a comercializar o entregar al cliente. Debe identificarse y almacenarse para poder recuperarla en cualquier momento. La GCS también se encarga de controlar la gestión e instalación de releases.

6.1.5. Actividades relacionadas

√ Control de Versiones (facilita la Gestión de la Configuración).

Consiste en saber, para cada elemento, cuál es la última versión, la relación entre distintas versiones (grafo de evolución de control de versiones) y dónde están (librerías de trabajo o desarrollo).

Esto facilita el control de los cambios porque ayuda a saber sobre qué version(es) hacer un cambio y cómo impactará (y dónde).

√ Herramientas CASE (automatizan la Gestión de la Configuración).

También permiten:

- ⋆ Control de versiones (mantener registro histórico de las diferentes versiones de un ECS, permitiendo recuperar cualquiera de ellas).
- ⋆ Facilidad de construcción o *building* (gestionar la compilación y linkado[8], el alcance de los distintos componentes del producto software, etc.; facilitada por la Gestión de la Configuración). Es necesario saber:
  - qué componentes enlazar
  - en qué versión
  - dónde están

  así como cómo está la información de identificación de la configuración y control de versiones.
- ⋆ Control del trabajo en equipo (CVS; facilitado por la Gestión de la Configuración). La compartición de elementos de trabajo genera problemas de sobreescritura de cambios, pero es necesaria porque es la única forma de realizar desarrollo en paralelo (problema de la integración del trabajo realizado o *merge*).
- ⋆ Gestión de Problemas (seguimiento de la evolución de los problemas que afectan al producto; facilitada por la Gestión de la Configuración). El cambio de un producto puede venir dado por cambio en los requisitos/necesidades o por un problema. Hay que:

[7] Si dichos atributos pueden ser desde simplemente el número de versión y/o revisión —SCCS, RCS— o bien una serie de pares *(atributo, valor)* —NSE, DSEE—.

[8] Tipo `make` y `Makefile`.

- recopilar información acerca del problema
- definir un proceso y forma de resolución del problema

Los pasos a seguir por la GCS ante un problema son:

- admisión y registros de informe de incidencia
- identificación de un responsable
- asignación de actividades correctivas
- monitorización del estado del problema
- registro de actividades correctivas del problema
- generación de informes correctivos del problema

La información que es necesario recoger se resume en:

- descripción del problema
- prioridad asignada
- causas estimadas (al menos en principio)
- ECS afectados
- persona o cliente que lo notificó
- responsable asignado
- actividades a llevar a cabo
- etc.

√ Mantenimiento (facilitado por la Gestión de la Configuración).

Intentar controlar el cambio para facilitar el mantenimiento.

√ Otros aspectos relacionados:

⋆ Metodologías para la integración de las actividades de GCS (determinan los productos que se van a generar y que tienen que ser controlados).

⋆ Entorno de desarrollo (uso de herramientas de GCS e integración con otras herramientas del entorno de desarrollo).

⋆ Organización (aparecen nuevos roles y responsabilidad, que hay que integrar en la organización del proyecto —interrelaciones—).

6.2. Identificación de la configuración

La **identificación de la configuración** es la primera de las actividades fundamentales de la GCS según IEEE.

Su objetivo es proporcionar visibilidad sobre el producto software, definiendo sus componentes (ECS) y la estructura de éstos. Para ello, hay que proporcionar servicios de identificación y accesibilidad.

La accesibilidad se refiere a su ubicación, mientras que la identificación consiste en señalar y asignar nombres significativos y consistentes a todos y cada uno de los eCS en cada fase de desarrollo. Para ello se siguen unos pasos:

1. Identificación de la estructura y componentes del producto software.
2. Selección de los ECS.
3. Definición de las relaciones en la configuración.
4. Definición de un esquema de identificación.
5. Definición y establecimiento de líneas de base.
6. Definición y establecimiento de bibliotecas de software.

Estas actividades se reparten durante el ciclo de vida del proyecto de la siguiente forma:

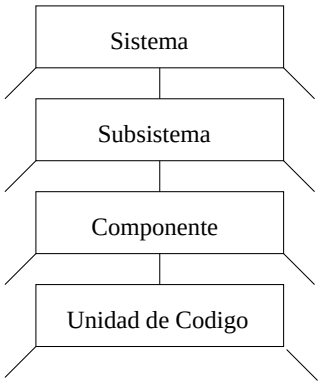
Actividades al inicio del proyecto

- ▷ Identificación:
  - ○ Respecto a los ECS:
    - · Identificación de la estructura y componentes del producto software. Nivel de visibilidad.
    - · Selección de los ECS.
    - · Definición del esquema de identificación.
    - · Selección de relaciones en la configuración a mantener.
  - ○ Respecto a las líneas base:
    - · Definición de líneas base.
- ▷ Accesibilidad:
  - ○ Definición de bibliotecas de software.

Actividades durante el proyecto

- ▷ Identificacar/etiquetar ECS generados.
- ▷ Establecer líneas base.
- ▷ Mantener relaciones en la configuración.
- ▷ Mantener bibliotecas software.

La identificación de estructura y componentes consiste en esbozar la jerarquía del producto (preliminar, se irá concretando, depende mucho del caso concreto: muy complejo, muy sencillo). Lo importante es decidir con qué nivel de visibilidad se va a controlar.



En la selección de los ECS hay que considerar dos tipos de productos:

Productos generados

- ∗ Indicados por la metodología seguida en el desarrollo.

  - Seleccionar los que van a estar bajo el control de la GCS (no todos tienen por qué y no siempre).
  - Decidir en qué forma se van a descomponer en otros.

Productos utilizados durante el desarrollo

  - Hardware.
  - Software.
  - Manuales y documentación.
  - Estándares.
  - Normativas.

Al buscar ECS se trata de separar en elementos distintos funcionalidades diferentes, para tratar de minimizar el impacto de posibles cambios. Además de diferenciar, se separa porque cuanta más funcionalidad asignemos a un elemento, más modificaciones supondrán un cambio.

Se tienden a definir/agrupar los elementos por distintos criterios:

- √ utilización múltiple (número de elementos de su mismo nivel o niveles superiores que utilizan el elemento —se trata de minimizar—)
- √ criticidad (gravedad del impacto de un fallo en ese componente)
- √ número de personas implicadas en su mantenimiento
- √ complejidad de su interfaz (las interfaces deberían ser simples —minimizar acoplamiento—)
- √ singularidad del componente con respecto al resto
- √ reutilización

En cuanto al número adecuado de ECS a identificar, como siempre, debe llegarse a un compromiso, pues pocos serán muestra de falta de visibilidad y demasiado sería inmanejable y costoso.

Como hemos visto, en GCS es necesario identificar:

- ECS
- versiones
- variantes
- configuraciones alternativas
- releases

Para ello es necesario definir un esquema de identificación:

- Identificación unívoca de ECS:

  - número o código del ECS
  - nombre del ECS
  - descripción del ECS
  - proyecto al que pertenece
  - fase/subfase de creación
  - fecha de creación
  - autor(es) del ECS (y aprobador-es-)
  - tipo de ECS (documentos, programa, elemento físico,...)
  - localización
  - línea base a la que pertenece
- Identificación unívoca de cada versión de un ECS:
  - número de versión
  - fecha de la versión

Podemos hablar de varios tipos de códigos:

No significativos
: Son códigos de fácil asignación pero difícil localización: debe mantenerse un registro de asignación de códigos.

Significativos
: Son de fácil localización pero sólo son útiles si son fáciles de recordar (puede complicar su interpretación).

La identificación de un ECS puede guardarse en etiquetas sobre el mismo ECS, en fichas o listados, e incluso en una base de datos específica.

Se puede considerar que los ECS son objetos y que están conectados con otros ECS mediante relaciones. Esta información nos ayudará a saber dónde se sitúa un ECS con respecto al resto, posiblemente mediante un grafo, que asimismo nos ayudará a ver la delimitación del impacto de un cambio.

Las relaciones interesantes a mantener son:

Composición (entre ECS)
: Puede darse en:
  - El software.
  - La documentación.

  En ella se identifica un ECS *contenedor* y un ECS *contenido* (por ejemplo, el diseño de un sistema y el diseño de sus bases de datos).

Derivación (entre ECS)
: Es el tipo de relación existente entre el código fuente y el código objeto, por ejemplo, o entre los casos de prueba y las trazas de ejecución de las pruebas, el diseño de un módulo y su código fuente, etc. Se identifica un ECS *origen* y un ECS *originado*.

Dependencia (de cualquier tipo, entre ECS)
: Como la que puede haber entre un modelo de datos y un DFD, o el código objeto y la versión del compildor.

Sucesión (entre versiones de un ECS)
: Puede estar *explícita*, en una tabla de sucesión que contenga código del ECS, versión antecesora y versión sucesora (trazabilidad), o bien *implícita* en la numeración de las propias versiones.

Equivalencia (entre copias de una versión de un ECS)
: Por ejemplo, una copia en disco será equivalente a la copia en papel o a la copia en cinta. Del mismo modo que en la sucesión, puede tenerse una tabla de copias que contenga el código del ECS, la versión, el número de copia[9], el tipo (documento en papel, manual, fichero, programa, máquina, cinta,...), la localización (máquina+directorio, armario+caja, edificio+habitación,...), etc.

En cuanto a la definición de líneas base, deben decidirse los *hitos* (*milestones*) en los que establecerlas. Lo normal es al final de determinadas fases, para identificar los resultados de las tareas de la fase y asegurar que ésta se ha completado.

En cada proyecto, pues, debe decirdirse qué líneas base se van a establecer, cuándo y su composición. ¿Cómo se establece una línea base? Físicamente, mediante etiquetado y almacenamiento, y lógicamente mediante un documento de identificación de la configuración.

	¿Cuándo se establece?	*¿Qué contiene?*
FUNCIONAL	Al finalizar el Análisis y Especificación de requisitos del sistema	Definición del problema Costes/tiempos Requisitos del sistema
DISTRIBUCIÓN DE FUNCIONES	Al finalizar el Análisis de Requisitos Software	ERS de cada componente software
DISEÑO PRELIMINAR	Tras el Diseño de la Arquitectura	Arquitectura Plan de pruebas
DISEÑO	Tras el Diseño Detallado	Diseño detallado Plan de implementación Diseño de las pruebas
PRODUCTO	Tras las Pruebas	Programas Informes de las pruebas
OPERACIÓN	Tras la Implementación	Manual de usuario Manual de instalación Manual de operación

Cuadro 6.1: Líneas base más comunes.

[9] De manera similar a como tratábamos las listas de distribución en trazabilidad ISO.

Una biblioteca de software es una colección controlada de software y/o documentación relacionados cuyo objetivo es ayudar en el desarrollo, uso o mantenimiento del software.

Para cada proyecto hay que decidir qué bibliotecas se van a usar, quién es el bibliotecario y toda una serie de procedimientos: procedimiento de introducción de elementos, procedimiento de acceso,...

BIBLIOTECA DE PRODUCCIÓN
- Una por proyecto
- ÁREA DE TRABAJO
  - Se establece al inicio del proyecto
  - Elementos de desarrollo (incompletos, incorrectos,...)
  - Cambio informal
  - Su ubicación suele ser el propio PC del desarrollador
- ÁREA DE INTEGRACIÓN
  - Donde se juntan las partes (ECS) antes de ser entregadas
  - No hay cambios
  - Suele ser un servidor (con repositorio)
- BIBLIOTECA DE PROYECTO O DE SOPORTE DEL PROYECTO
  - Almacena los ECS una vez aprobada la línea base
  - Control de cambios interno semiformal

BIBLIOTECA MAESTRA
- Fin de proyecto y releases (entregas al cliente)
- Cambio formal
- Una por proyecto
- Pocos permisos de escritura, muchos de lectura

REPOSITORIO SOFTWARE
- Almacena todo al dar por finalizado el proyecto
- Una vez retirado el producto
- No hay cambios
- Global a toda la empresa

REPOSITORIO DE COMPONENTES REUTILIZABLES
- Elementos del repositorio software reutilizables (GCS)
- Se supone almacenamiento a largo plazo (backup)

Cuadro 6.2: Bibliotecas software.

6.3. Control de cambios en la configuración

El **control de cambios** es la actividad de GCS más importante, la segunda tarea básica propuesta por IEEE en este contexto. Su objetivo es proporcionar un mecanismo riguroso para controlar los cambios. Normalmente combina procedimientos humanos y el uso de herramientas automáticas.

Se consideran dos tipos de cambios fundamentalmente:

- corrección de defectos
- mejora del sistema[10]

Se definen asimismo varios *niveles* de control:

Informal: Antes de que el ECS pase a formar parte de una línea base, aquél que lo ha desarrollado podrá realizar cualquier cambio justificado sobre él.

Semiformal o a nivel de proyecto: Una vez que el ECS pasa la revisión técnica formal y se convierte en una línea base, para que el encargado del desarrollo pueda realizar un cambio debe recibir la aprobación del jefe de proyecto (si el cambio es local —sólo afecta a ese proyecto—) o del comité de control de cambios (si tiene impacto sobre otros ECS).

Formal: Se suele adoptar una vez que se empieza a comercializar el producto, cuando se transfieren los ECS a la biblioteca maestra. Todo cambio deberá ser aprobado por el comité de control de cambios.

El **Comité de Control de Cambios** puede ser una persona o un grupo de personas que tienen la autoridad sobre la aprobación, denegación y priorización de un cambio en un proceso formal o semiformal. Es necesario que posean una visión global del producto para evaluar el impacto del cambio y poder tomar una decisión.

No obstante, la responsabilidad en los cambios no es sólo de este comité, sino que también tienen responsabilidades todos los miembros del proyecto, el jefe del mismo y el bibliotecario.

Es necesario establecer de forma precisa, al comienzo de cada proyecto, cuál será el proceso de gestión de cambios que se va a utilizar. Generalmente, cualquier miembro de un proyecto puede solicitar un cambio, cualquier miembro puede ser el encargado de realizarlo, de informar sobre el estado real de los cambios que se están realizando,... El encargado de testear que los procedimientos de control de software se siguen es el jefe de proyecto, que también puede informar del estado de los cambios, y por supuesto participa en la evaluación del impacto y coste de los mismos. En cuanto al bibliotecario, controla la realización de los cambios, es el responsable de que no se modifique la biblioteca maestra hasta que el cambio sea revisado y aprobado, etc.

Un proceso formal de control tendría la siguiente estructura:

[10] Los clientes tienden a clasificar todos los cambios en la primera categoría, mientras que los programadores lo hacen en la segunda.

1. Iniciación del cambio (solicitud).
   - Si el cambio es para mejorar algo: formulario de solicitud de cambios.
     - Motivo del cambio (descripción del problema o cambio de requisitos).
     - Datos del solicitante.
     - Qué hay que cambiar.
     - Otros cambios relacionados o ECS afectados.
     - Alternativas.
   - Si el cambio se realiza por algún problema: informe de incidencia.
     - Fecha y hora de la incidencia.
     - Descripción.
     - Efectos.
     - Cómo replicar la incidencia.
     - Tipo de prueba que se realizaba.
     - Volcado de datos.
     - Datos del informante.
2. Clasificación y registro.
3. Aprobación o rechazo inicial (comité de control de cambios).
4. Evaluación de la solicitud de cambio (si se ha aprobado):
   - Informe de cambio:
     - Estimación del esfuerzo necesario.
     - Coste.
     - Impacto.
5. Aprobación o rechazo: orden de cambio.
   - Cambio a realizar y restricciones a respetar.
   - Criterios de revisión y auditoría.
   - Baja en la biblioteca.
6. Realización del cambio, seguimiento y control (gestión de problemas si se debe a un problema).
7. Verificación del cambio:
   - Se ha corregido el problema o se han satisfecho los requisitos modificados.
   - No se han introducido nuevos defectos.
   - Se añade a la biblioteca de soporte del proyecto.
8. Notificación al originador del cambio.

Si el cambio es sólo "semiformal", se hará al menos evaluación y seguimiento del cambio.

6.4. Generación de informes de estado de la configuración

La **generación de informes de estado** es la tercera actividad de la GCS según IEEE. Su objetivo es mantener a desarrolladores, gestores y usuarios/clientes al tanto del estado de la configuración y su evolución.

Su importancia radica en que, gracias a ella, es posible:

- √ la *continuidad* del proyecto (se trata de lograr que el proyecto siga adelante cuando, por ejemplo, el jefe de proyecto deja la empresa)
- √ se evita la *duplicidad* (ya que si no se guarda información acerca de lo que se ha hecho, se puede estar repitiendo trabajo ya realizado)
- √ se evita repetir los errores
- √ se consigue repetir lo que se hizo bien
- √ se consigue encontrar las causas de problemas

Las actividades que a su vez componen la generación de informes son, a grandes rasgos, tres:

1. Captura de información:

   La información proviene de otras actividades de GCS.
   La cantidad y tipo de información a capturar depende de las características del proyecto, de su tamaño y complejidad. El *Plan de Gestión de la Configuración* deberá indicar la información mínima a capturar.

2. Almacenamiento:

   Lo mejor suele ser almacenar esta información en una base de datos y utilizar herramientas automatizadas para gestionarla.

3. Generación de informes:

   Pertinentes.
   De análisis post-mortem del proyecto (también suelen hacerse del producto, fundamentalmente para históricos).
   Para estimaciones para proyectos futuros.

En cuanto a los "productos" (*deliverables*) de esta actividad, son fundamentalmente de dos categorías: *registros* e *informes*. Para cada proyecto en particular, a través del mencionado Plan de Gestión de la Configuración (ver sección 6.6), se decide qué es un registro y qué es un informe. Si la empresa está conforme ISO 9000, tendremos ya determinados qué registros hay que obtener y qué informes hay que hacer.

- Ejemplos de registros:
  - De identificación de la configuración:
    - De ECS
    - De versiones y variantes
    - De configuraciones alternativas
    - De relaciones entre ECS (referencia a otros ECS)
    - De líneas base (referencia a los ECS que la componen)
    - De releases:
      - Fecha de liberación
      - Composición: ECS+versión
      - Diferencias con la release anterior
    - De instalaciones:
      - Lugar
      - Fecha
      - Release
  - De control de cambios:
    - De incidencias:
      - Datos
      - Resultado
      - Historia
    - De solicitudes de cambio (referencia a incidencias)
    - De cambios (referencia a solicitud de cambio)
      - Evaluación e impacto
      - Disposición
      - Plan de implementación
      - Restricciones y criterios de revisión
      - Historia del cambio: solicitud, aprobación/denegación,...
    - De cambios (referencia a solicitud de cambio ó notificación de cambio)
      - Sobre código
      - Sobre documentación
      - Sobre bases de datos
  - Auditoría de la configuración
    - Fecha
    - Problemas detectados y recomendaciones
  - Actas de las reuniones del Comité de Control de Cambios
    - Asistentes
    - Propósito
    - Disposiciones

- Ejemplos de informes:
  - Tipos según la previsión:
    - Planificados
    - Bajo demanda
  - Tipos según la complejidad:
    - Directos (contenido de los registros)
      - Inventario de ECS
      - Estado de los cambios
    - Indirectos (diferencias entre versiones)

6.5. Auditoría de la configuración

La **auditoría de la configuración** es la última y más costosa de las actividades en GCS, ya que, por ser auditoría, requiere de personal experimentado ajeno al proyecto.

La diferencia entre una *auditoría general* y una *auditoría de la configuración* reside en que la primera es una verificación independiente de un trabajo o del resultado de un trabajo o grupo de trabajos que se hace con el objetivo de evaluar su conformidad respecto de especificaciones, estándares y/o aspectos contractuales u otro tipo de criterios, mientras que la segunda es la forma de comprobar que, efectivamente, el producto que se está construyendo es lo que pretende ser.

Esta función a veces se considera fuera de la Gestión de la Configuración y dentro de la garantía de calidad, sin embargo, también tiene relación con las actividades de Verificación y Validación... en realidad es un punto de intersección de todas ellas.

Hay varios tipos de actividades de control de calidad:

Revisiones de fase: Se realizan al final de cada fase y su objetivo es examinar los productos de dicha fase. Las revisiones propias de la GCS son aquéllas en que se establecerán las líneas base. El objetivo de esta revisión es descubrir problemas, no ver que todo está bien.

Revisiones de cambios: Deben verificar que se han realizado correctamente los cambios aprobados sobre una línea base.

Auditorías: Se pueden realizar al final del proceso de desarrollo de software, y su objetivo es examinar el producto en su conjunto, o bien pueden ser periódicas y entonces comprueban lo que está hecho del producto hasta el momento.

La tarea de revisión implica tres tipos de funciones:

√ *Verificar la configuración actual del software con respecto a la línea base anterior.* Debe haber correspondencia y trazabilidad entre los elementos de configuración (ECS) que aparecen en una línea base y los que aparecen en las líneas base que la preceden y la siguen.

√ *Validar que la configuración actual del software satisface la función que se esperaba del producto* en cada hito del proceso de desarrollo (se realiza contra los requisitos).

√ *Valorar si una línea base es aceptable o no*, teniendo en cuenta los resultados de la verificacón y validación y otro tipo de comprobaciones.

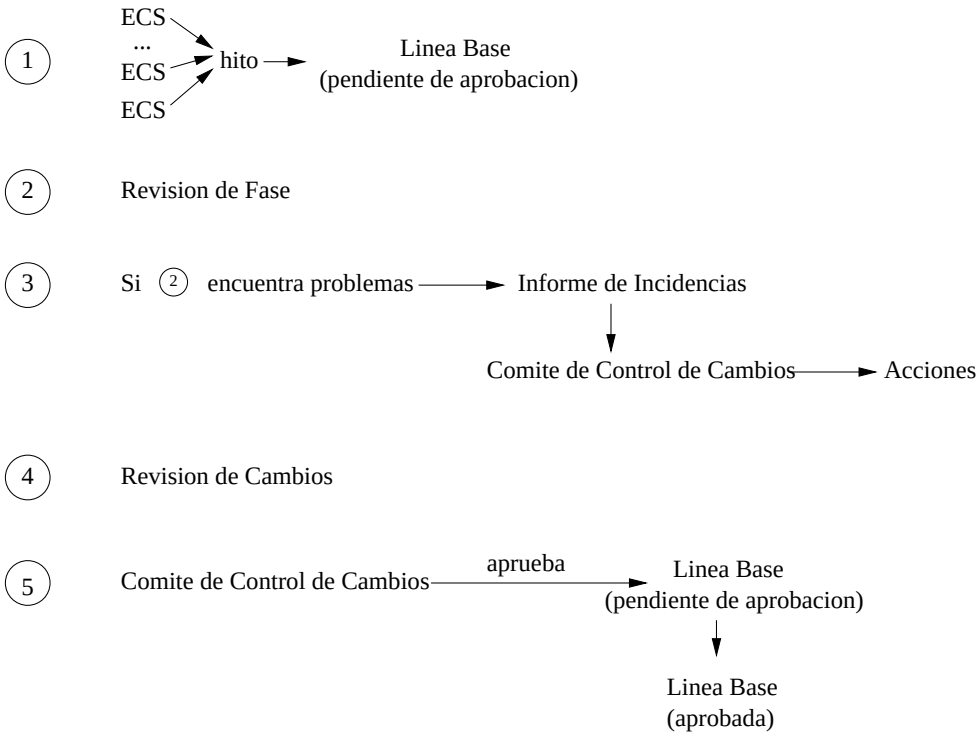


Figura 6.2: Esquema de una tarea de revisión.

Tipos de auditorías:

- Funcional.- Sus entradas son los informes de las pruebas y los ECS, y su objetivo, comprobar que se han completado las todos los tests pruebas para cada ECS auditado y que, de acuerdo con sus resultados, satisfacen los requisitos impuestos sobre él.
- Física.- Su objetivo es verificar la adecuación, completitud y precisión de la documentación que constituye las líneas base de diseño y de producto (en general de cualquier línea base). Se trata de asegurar que representa el software que se ha codificado y probado. Tras esta auditoría se establece la línea base de producto (cuando el producto ya está listo para ser implantado) y tiene lugar inmediatamente después de haberse superado la auditoría funcional.
- Revisión formal de Certificación (en su entorno operativo). Este último tipo de auditoría pretende certificar que el ECS se comporta correctamente una vez que está instalado y se encuentra en su entorno operativo.

6.6. Plan de Gestión de la Configuración

El **Plan de Gestión de la Configuración** es un documento que se debe establecer/productir antes del proyecto, en su comienzo. Define políticas, estándares y procedimientos que se van a utilizar en la gestión de la configuración del proyecto (quién debe aprobar qué cosas, qué informes se realizarán, sobre qué y cuándo,...).

Aunque ISO indica que debe hacerse, no es usual confeccionar un Plan de Gestión de la Configuración por proyecto. Lo que propone IEEE es que en las PYMES se elabore un único plan general (que funcionará a modo de plantilla) y añadir acciones preventivas o correctivas para cada proyecto.

6.6.1. Estructura del Plan

Según IEEE, la estructura básica de un Plan de Gestión de la Configuración será:

- Introducción[11].
  - Propósito del plan y a quién va dirigido.
  - Alcance: proyectos a los que se aplica, ECS bajo control, supuestos que podrían tener un impacto sobre la GCS, etc.
  - Definiciones y acrónimos, normalmente sólo del proyecto tratado.
  - Referencias a estándares IEEE, otros planes de proyecto, etc.
  - Definición de alto nivel del proceso de GCS.
- Especificaciones de gestión para actividades GCS.
  - Organización: contexto organizativo, relaciones de dependencia y autoridad, etc.
  - Responsabilidades: comité de control, bibliotecario, etc.
  - Implantación del plan: establecimiento del comité, líneas base, revisiones y auditorías, etc.
- Políticas, directivas y procedimientos aplicables a la GCS.
  - Niveles del software en un árbol jerárquico.
  - Nombrado de programas y módulos.
  - Designación de versiones.
  - Identificación de productos software.
  - Identificación de documentación.
  - Identificación de medios y ficheros.
  - Proceso de liberación de documentos.
  - Proceso de liberación de productos software.
  - Procedimientos de informes de incidencias, solicitudes de cambio, órdenes de cambio,...

[11] Los requisitos mínimos en la introducción son: qué líneas base se establecen, cuándo, qué ECS forman parte de ellas, qué ciclos de aprobación se aplican a los cambios, personal,...

- Estructura y forma de operación de los comités de control.

- Actividades de GCS.
  - Identificación de la configuración.
    - Descripción del esquema de identificación para los ECS.
    - Enumeración de las líneas base y para cada una de ellas se describe:
      - momento de establecimiento
      - ECS a incluir
    - Bibliotecas y repositorios a utilizar, describiendo procedimientos de inserción, recuperación, protección, etc.
  - Control de la configuración.
    - Mecanismos de iniciación de cambios (formularios, procedimientos).
    - Mecanismos de evaluación de cambios (procedimientos, criterios).
    - Mecanismos para aprobación/rechazo de cambios (procedimientos, autoridad).
    - Mecanismos para verificación de cambios aprobados.
    - Mecanismos para Gestión de Problemas que se utilizarán.
    - Mecanismos para Control de Versiones.
  - Informes de estado de la configuración.
    - Registros a mantener.
    - Informes a generar.
    - Procedimientos de captura, almacenamiento y procesamiento de la información.
  - Auditoría de la Configuración: descripción de cada una de las auditorías y revisiones que se van a realizar, indicando
    - Objetivos
    - ECS a auditar o revisar
    - Participantes
    - Procedimiento de realización
    - Procedimiento para registrar deficiencias
    - Listas de comprobación y cuestionarios a utilizar
    - Criterios de aprobación del ECS
  - Control de suministradores y subcontratistas: forma en que se va a controlar que los subcontratistas y suministradores satisfacen los requisitos de gestión de configuración establecidos[12].
  - Recogida y retención de registros.

[12]Concepto de *calidad concertada*: alguien sirve algo y el servido no hace auditorías periódicas si no tiene ningún problema con el producto.

- Qué documentación de GCS retener.
- Métodos y recursos para recopilación, salvaguarda y mantenimiento de esta documentación, incluyendo cualquier tipo de backup.
- Período de retención.

Apéndices

Apéndice A

Ejemplo de implantación de ISO 9000-3 en una empresa

Lo principal de un Sistema de Calidad es que debe estar documentado:

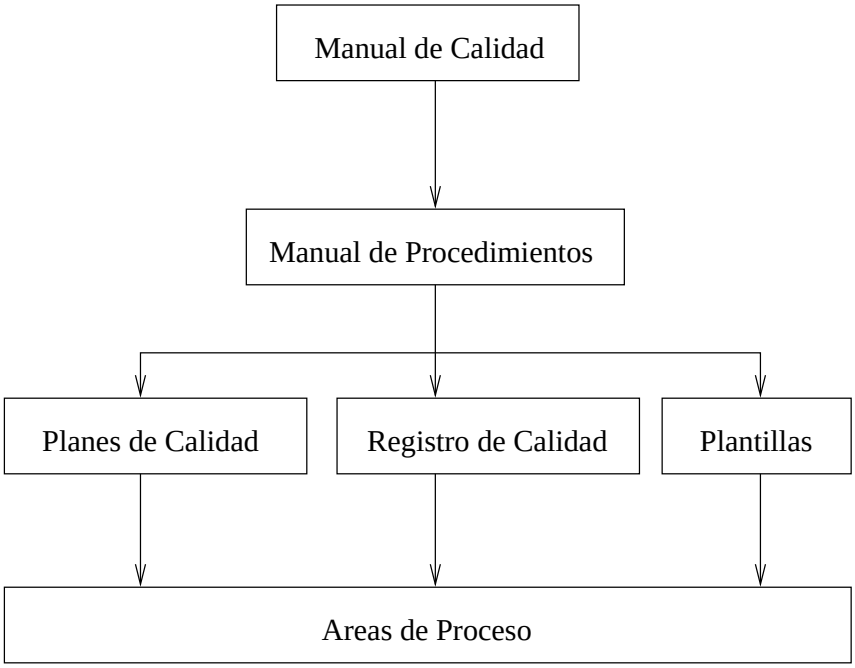


Figura A.1: Estructura documental de un Sistema de Calidad.

El *Manual de Calidad* es algo particular de la empresa, único para ella su área o departamento que se certifique. Es el marco de desarrollo de proyectos que se apoya en un conjunto de procedimientos de referencia. Debe dar una visión global (de alto nivel) de la empresa, sus objetivos y principios de operación, así como una descripción más o menos detallada de cada una de sus áreas.

El *Manual de Procedimientos* entra más en detalle en lo que respecta al conjunto de procedimientos que forman la empresa, es más técnico y hará referencia, tal y como se indica, a tres elementos:

- *Planes de Calidad* (o Planes de Realización), aplicaciones particulares de las directrices del Sistema de Calidad a cada proyecto en particular.
- *Registro de Calidad*, evidencia documental de que se ha realizado algo. Los datos que contiene no son modificables (la especificación de requisitos software sí lo es).
- *Plantillas*.

Por último, las *Áreas de proceso* son, por ejemplo: Gestión de Proyectos, Gestión de Requisitos, Soporte, Post-Venta,...

A.1. ¿Por dónde empezar?

El primer paso es establecer cómo será el *Sistema de Información Documental* de la empresa (formato electrónico —un servidor, una intranet...—, formato papel, etc). Sea cual sea la solución adoptada, se tendrá en cuenta para establecer las directrices en el Procedimiento de Control de la Documentación (quién la puede modificar, disponibilidad y cuestiones similares). La filosofía de ISO es que todo debe estar perfectamente documentado, distribuido, permanentemente actualizado...

Lo primero que debe hacerse es, pues, redactar el **Manual de Calidad**, un documento breve que remitirá a otros:

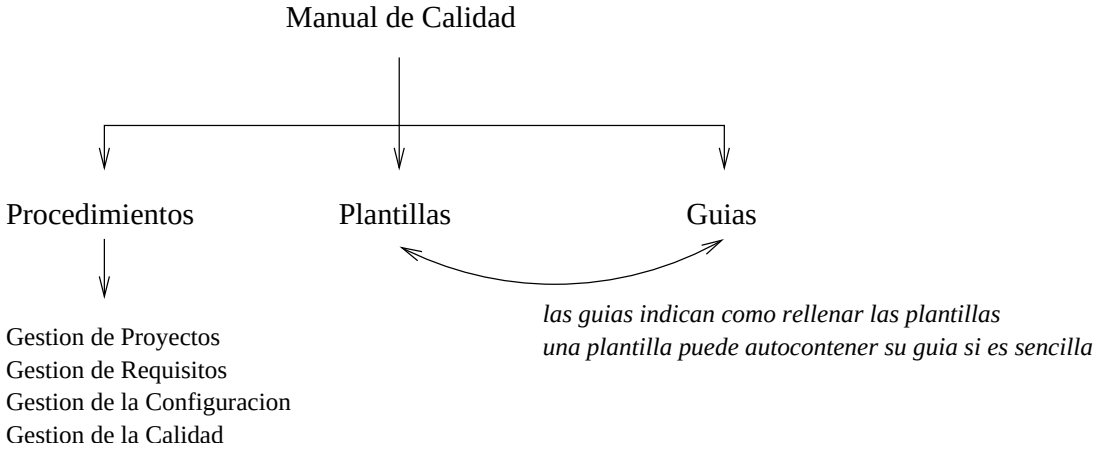


Figura A.2: Referencias del Manual de Calidad.

A.1.1. Cómo es un documento en un Sistema de Calidad

Cada documento relevante de un Sistema de Calidad debe ajustarse a una serie de directrices:

√ Cada versión debe ser aprobada por el mismo cargo (aunque no sea la misma persona física).

- √ La primera vez, en el registro de cambios, se indica que es la versión inicial y la fecha. En versiones siguientes, se debe indicar la fecha y la descripción de cambios realizados.
- √ En la lista de distribución debe aparecer quién tiene copia del documento para conseguir una actualización permanente ante posibles modificaciones (el primero de la lista será, obviamente, Dirección)
- √ Puede haber otras directrices que rijan los contenidos específicos de los distintos documentos.
- √ Se incluirá al final de cada documento la plantilla del plan de proyectos (según el plan se incluirán registro de cambios y lista de distribución o no), seguimiento de proyectos e informe de cierre.

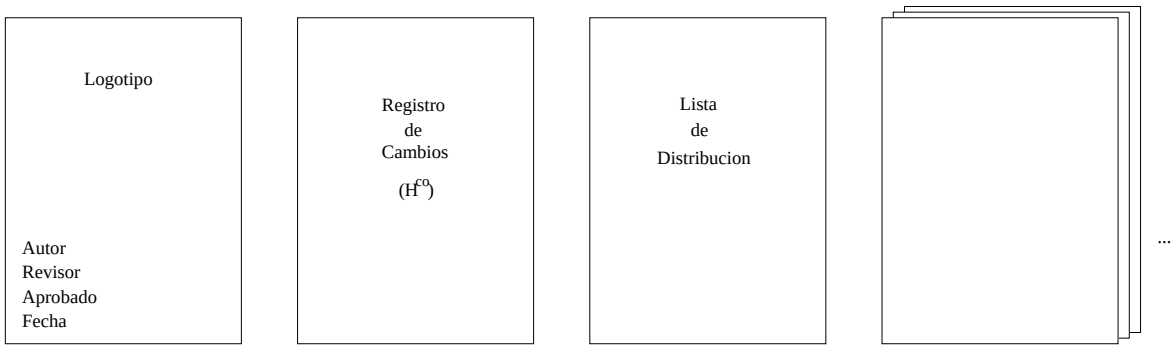


Figura A.3: Formato de un documento en un Sistema de Calidad.

La distribución de nuevas versiones depende de si la empresa trabaja con una intranet o con papel. En una intranet se sobreescribirán los ficheros (o se eliminarán los antiguos), gestionando el sistema de permisos y la actualización será inmediata. En papel, se suele enviar el nuevo registro de cambios a quienes corresponda, las hojas modificadas y una guía que indique dónde se debe colocar cada hoja sustituida (para no tener que enviar el documento completo).

A.2. Aspectos Principales del Sistema de Calidad

A.2.1. El Manual de Calidad

Es el eje principal del Sistema de Calidad:

- ↪ Describe la empresa y su estructura organizativa
- ↪ Introduce el Sistema de Calidad de la empresa
- ↪ Introduce las áreas de proceso consideradas en el Sistema de Calidad
- ↪ Para cada área de proceso
  - ⋆ describe el área de proceso

- enumera los procedimientos organizados (como plantillas[1] del plan de proyectos, informe de cierre,...)

Cada área puede tener distintos procedimientos.

A.2.2. El Manual de Procedimientos

Cada área de proceso se especificará por una serie de procedimientos que deberán ser minuciosamente descritos en el **Manual de Procedimientos**.

A.2.3. Revisión de Ofertas y Contratos

[1]Cualquier tipo de *plantilla* no es conveniente añadirlo porque si cambia hay que modificar el manual, dejar constancia de cuándo y por qué se ha modificado,...

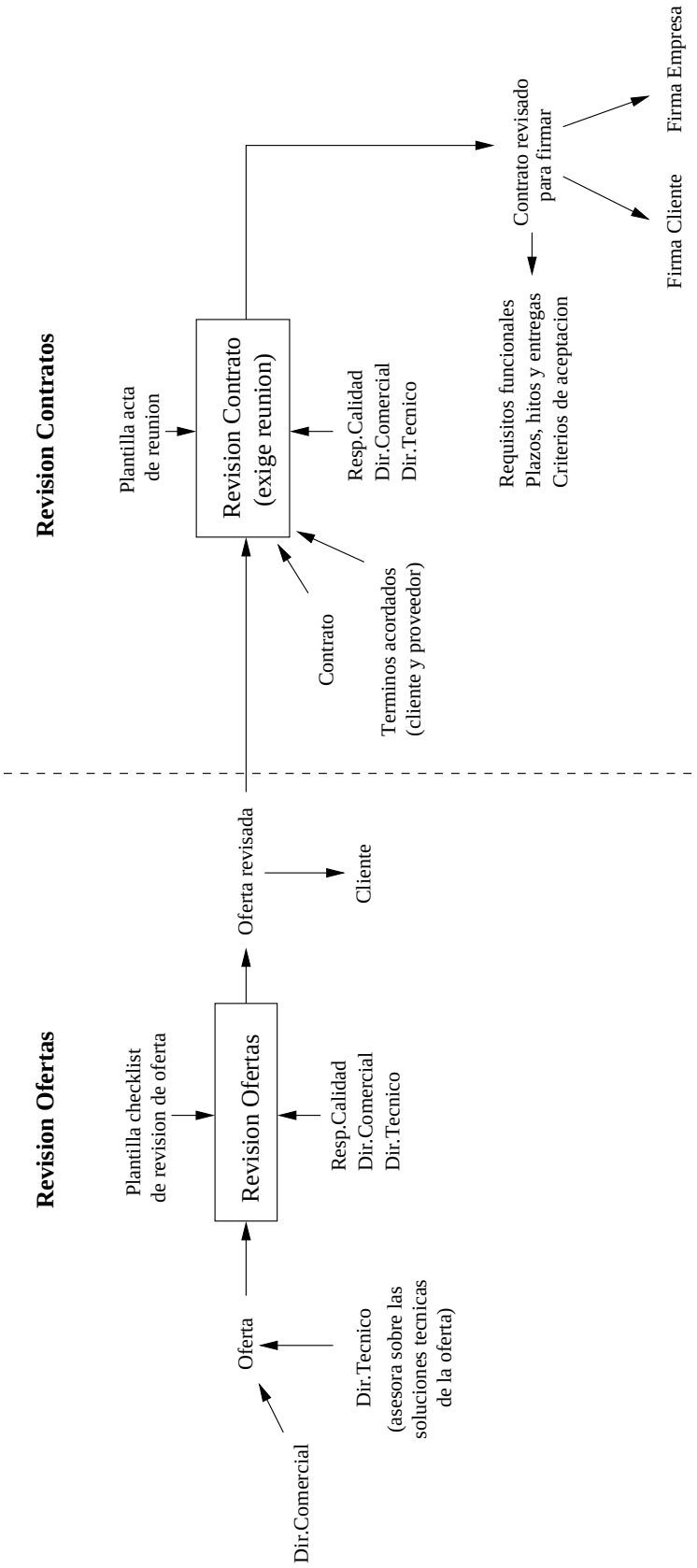


Figura A.4: Revisión de Ofertas y Contratos.

A.2.4. Ciclo de Vida

Responde a la pregunta: *¿cómo lo hago?*

El **Ciclo de Vida** permite flexibilidad dentro de un Sistema de Calidad:

- En el Plan de Proyectos se deciden qué cosas se hacen
- Se puede dar de alta una acción preventiva, que si es acertada tendríamos un Ciclo de Vida alternativo

Para desarrollar un Ciclo de Vida:

```
                    | Fase |
        | Productos de Entrada | Productos de Salida |
                | Responsable de Producto |
  | Criterios de entrada de fase | Criterios de salida de fase |
| Revisión (Jefe de Desarrollo) | Aprobación (Jefe de Proyecto) |
        | Verificación de fase (Responsable de Calidad) |
```

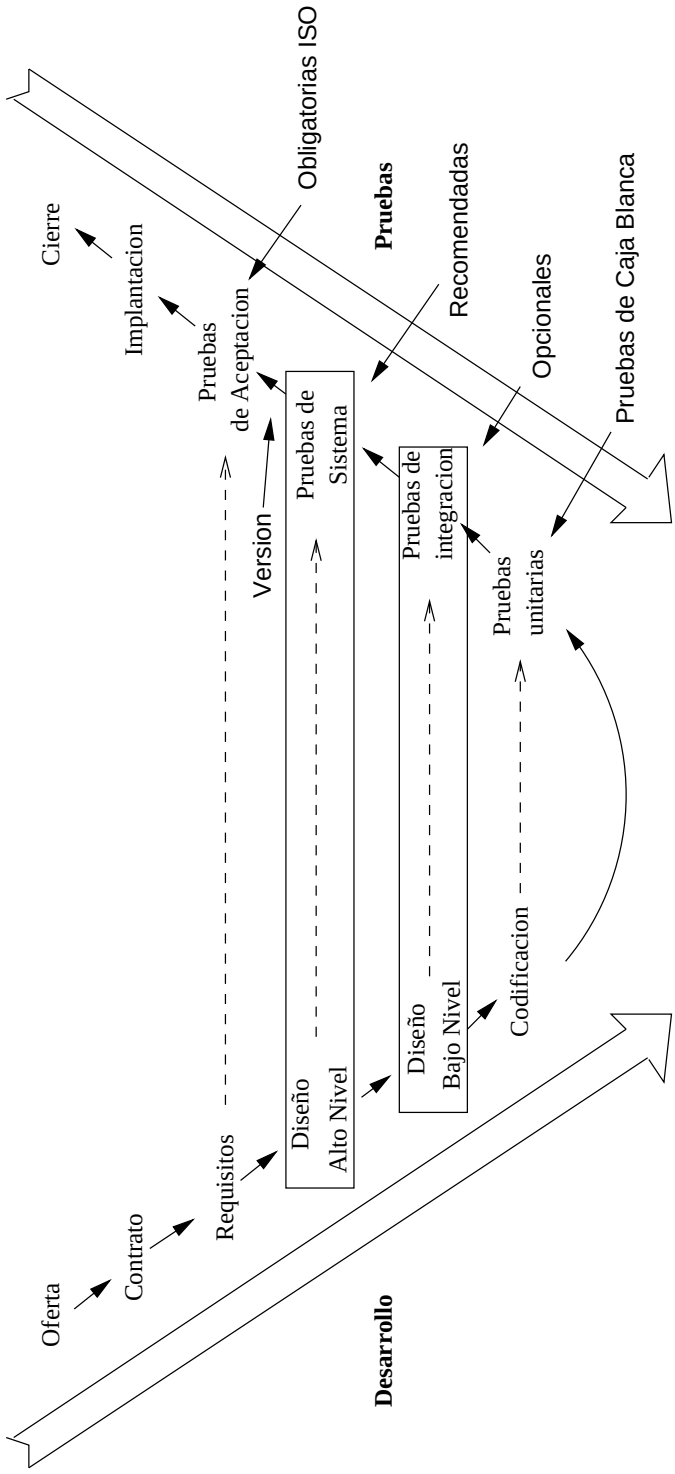


Figura A.5: Ciclo de Vida.

A.2.5. Diseño

Entendemos por **Diseño** el de Alto Nivel (análisis).

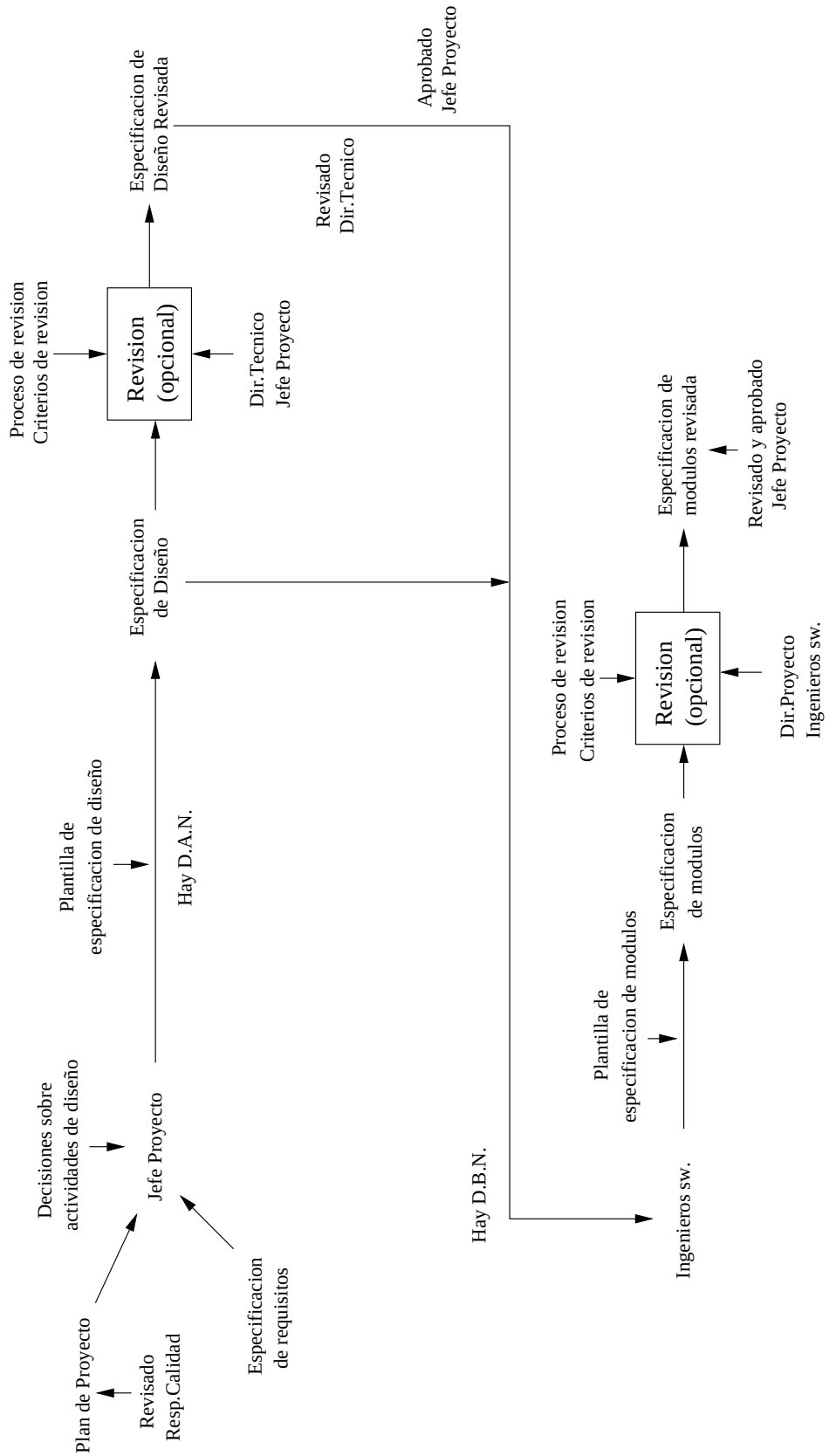


Figura A.6: Diseño de Alto Nivel (análisis).

A.2.6. Programación

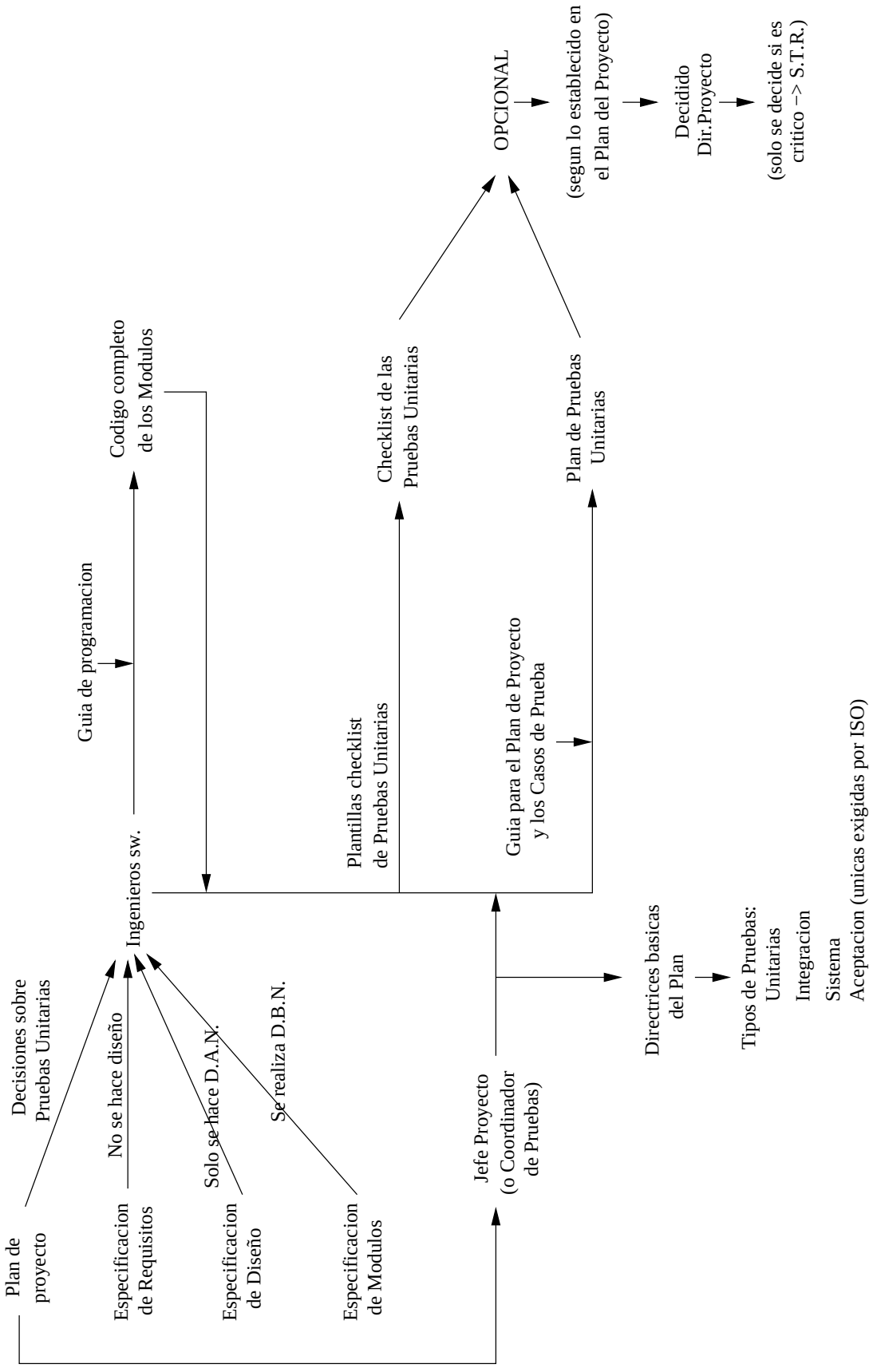


Figura A.7: Programación.

A.2.7. Pruebas

Repasaremos los distintos tipos de pruebas susceptibles de ser realizadas en el seno del Ciclo de Vida de un Proyecto.

Pruebas Unitarias

Este tipo de pruebas se realiza normalmente sólo si el proyecto es crítico.

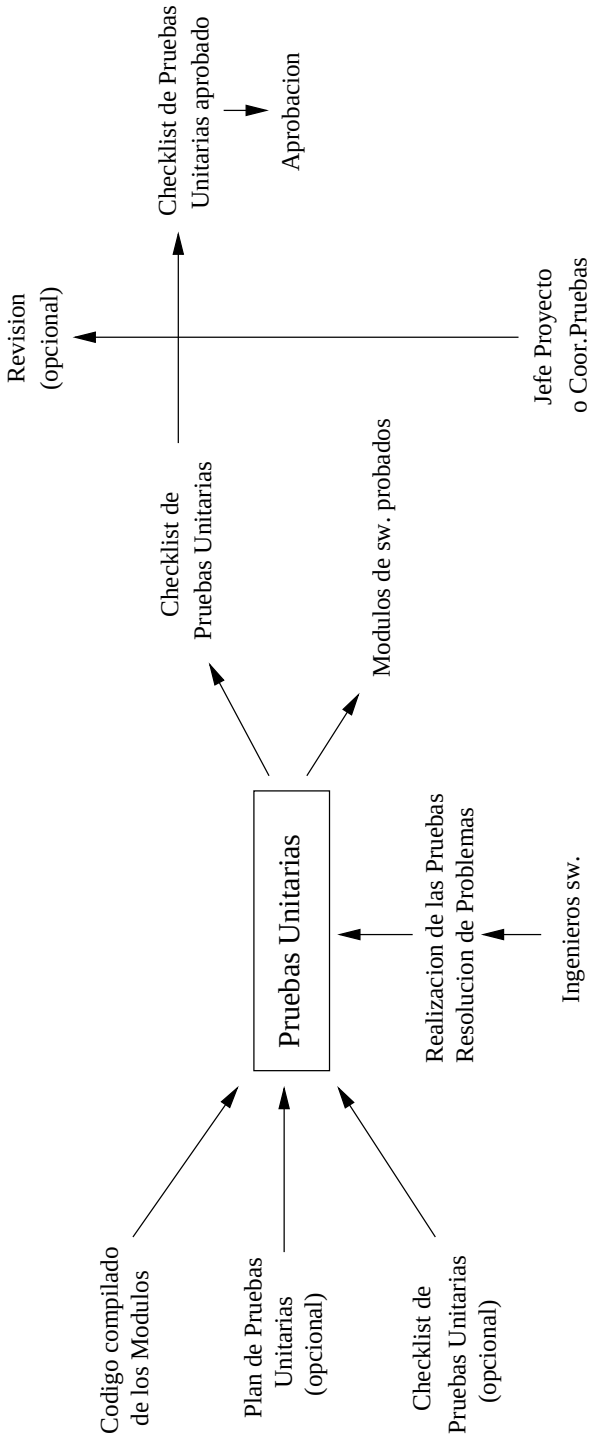


Figura A.8: Pruebas Unitarias.

Pruebas de Integración

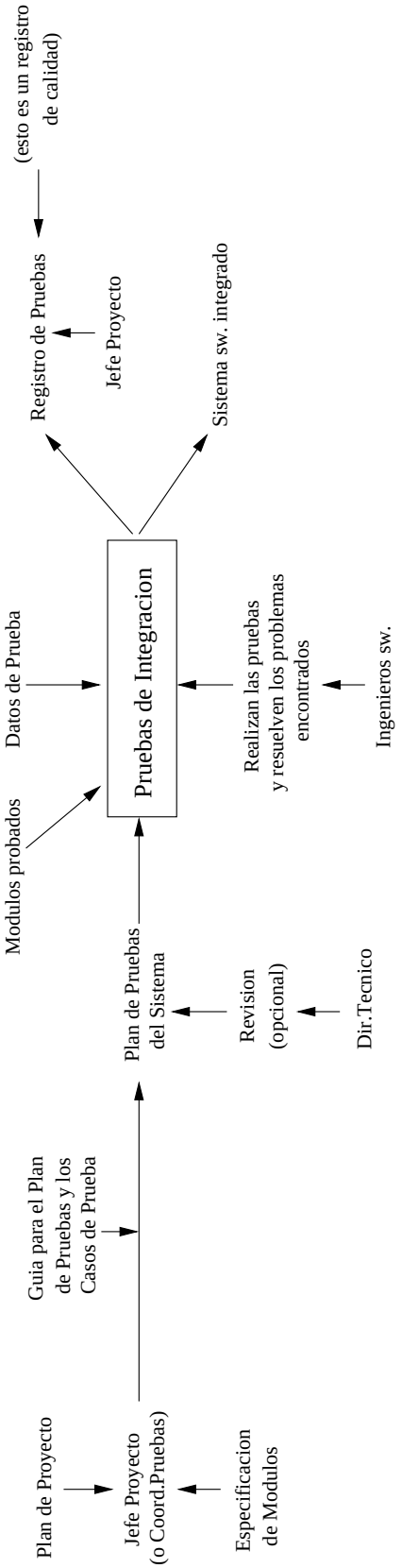


Figura A.9: Pruebas de Integración.

Pruebas de Sistema

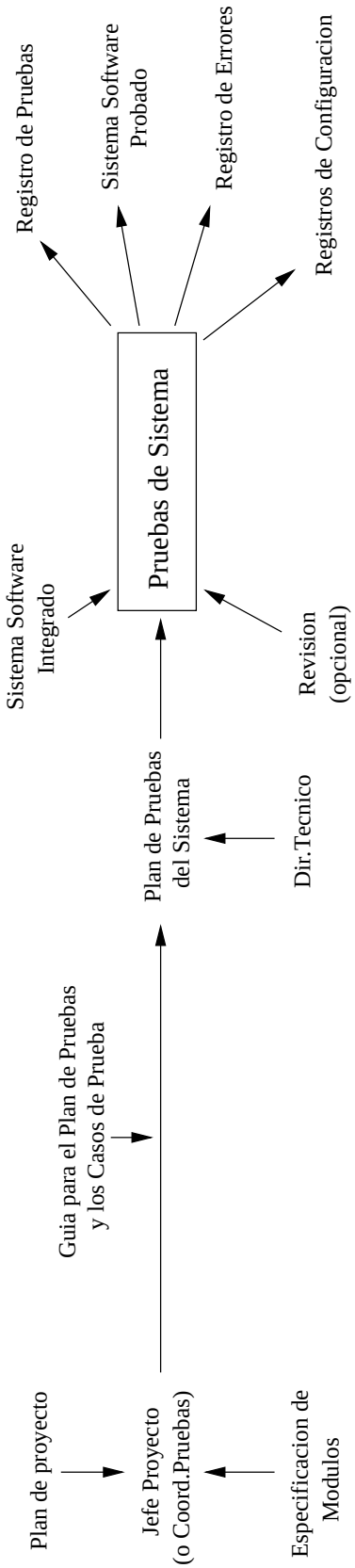


Figura A.10: Pruebas de Sistema.

Pruebas de Aceptación

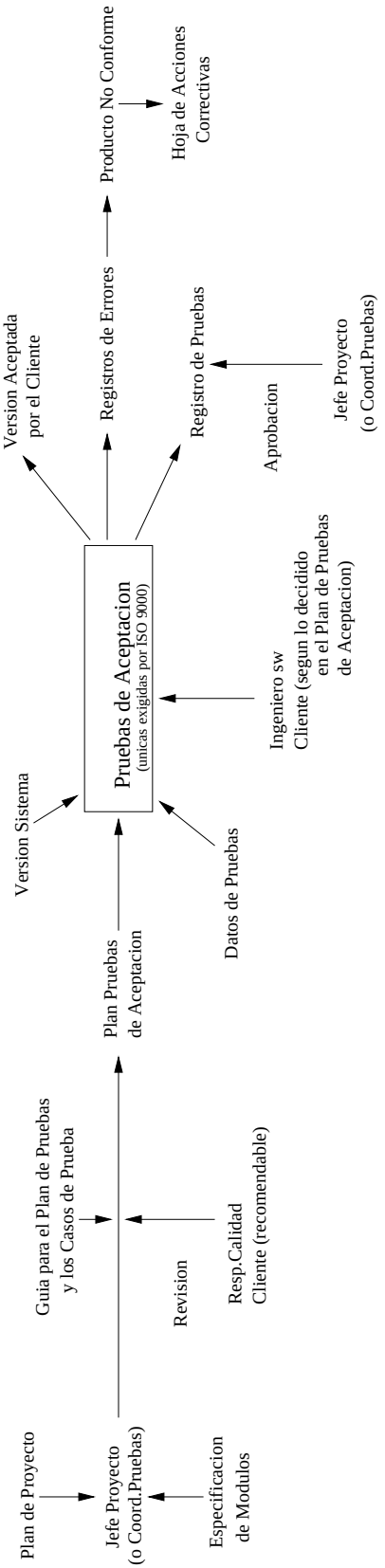


Figura A.11: Pruebas de Aceptación.

A.2.8. Notificación de Errores

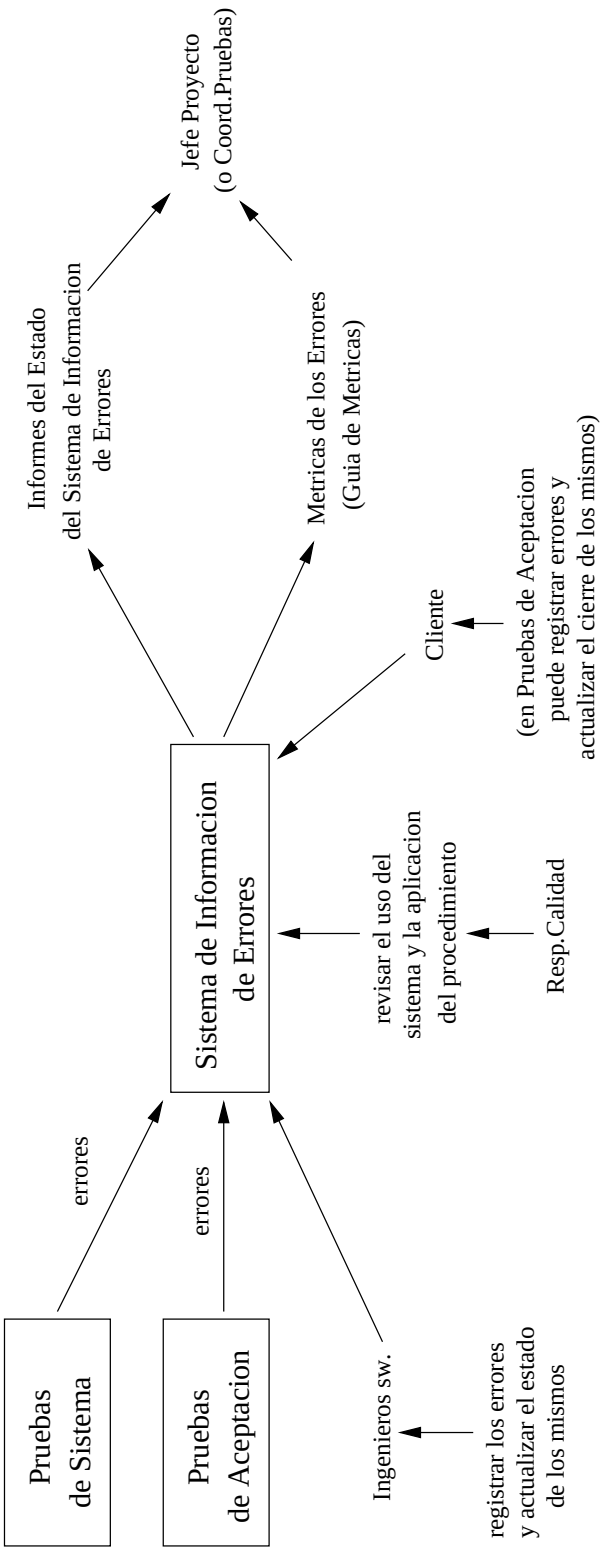


Figura A.12: Notificación de Errores.

A.2.9. Atención al Cliente

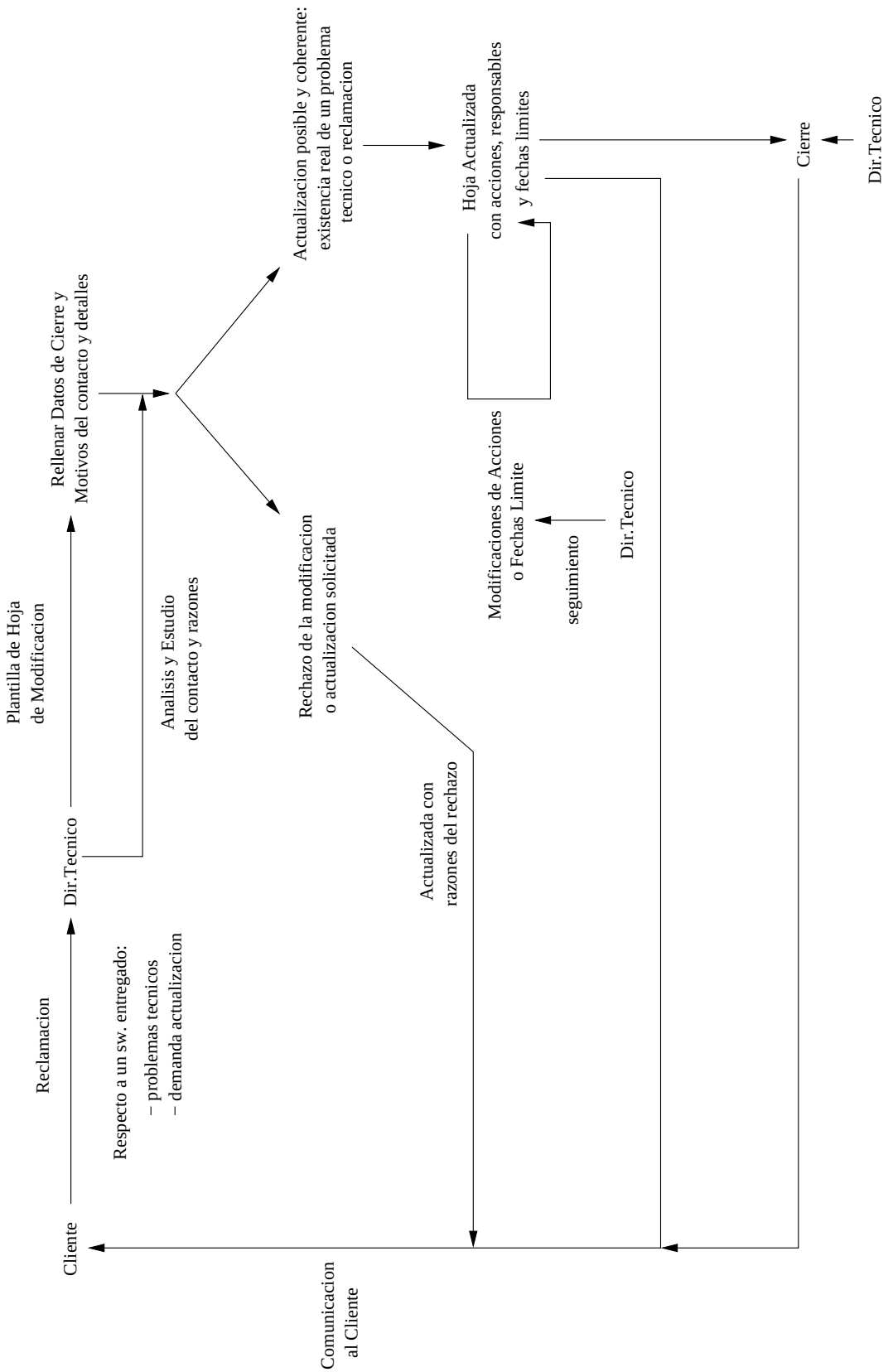


Figura A.13: Atención al Cliente.

A.2.10. Productos Cedidos por el Cliente

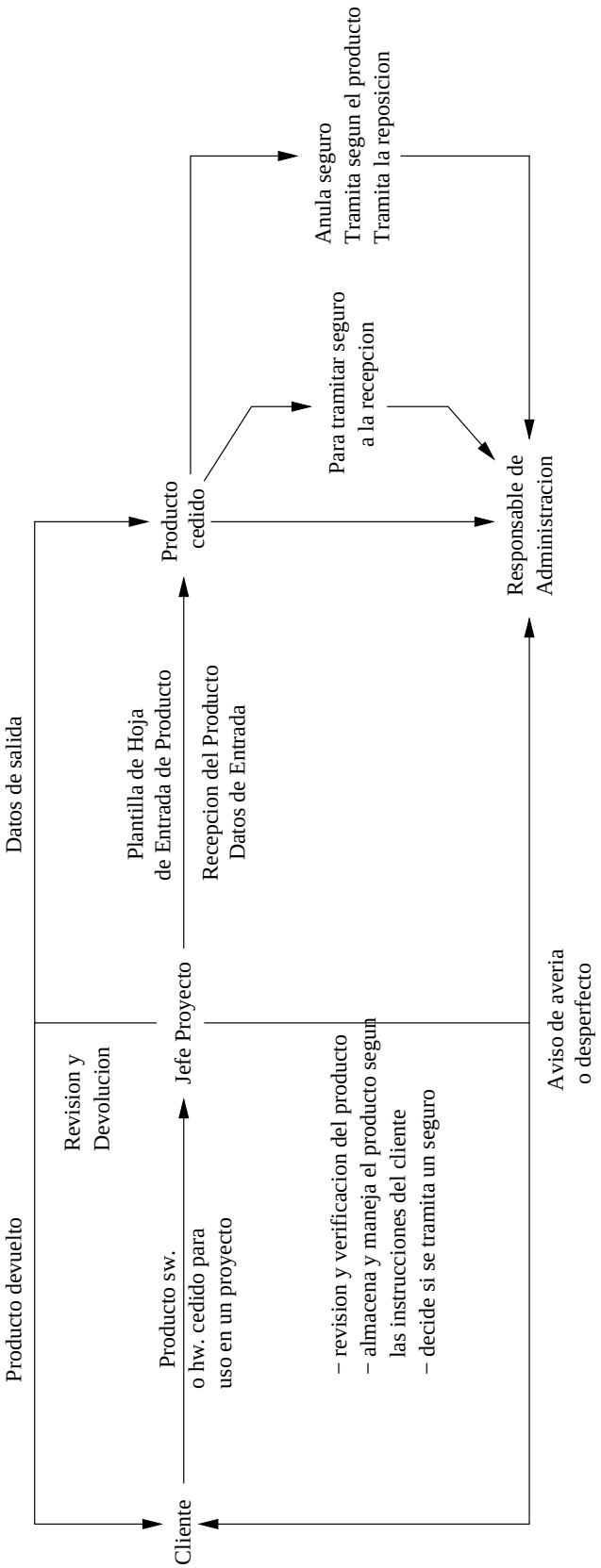


Figura A.14: Productos Cedidos por el Cliente.

A.2.11. Formación del Personal

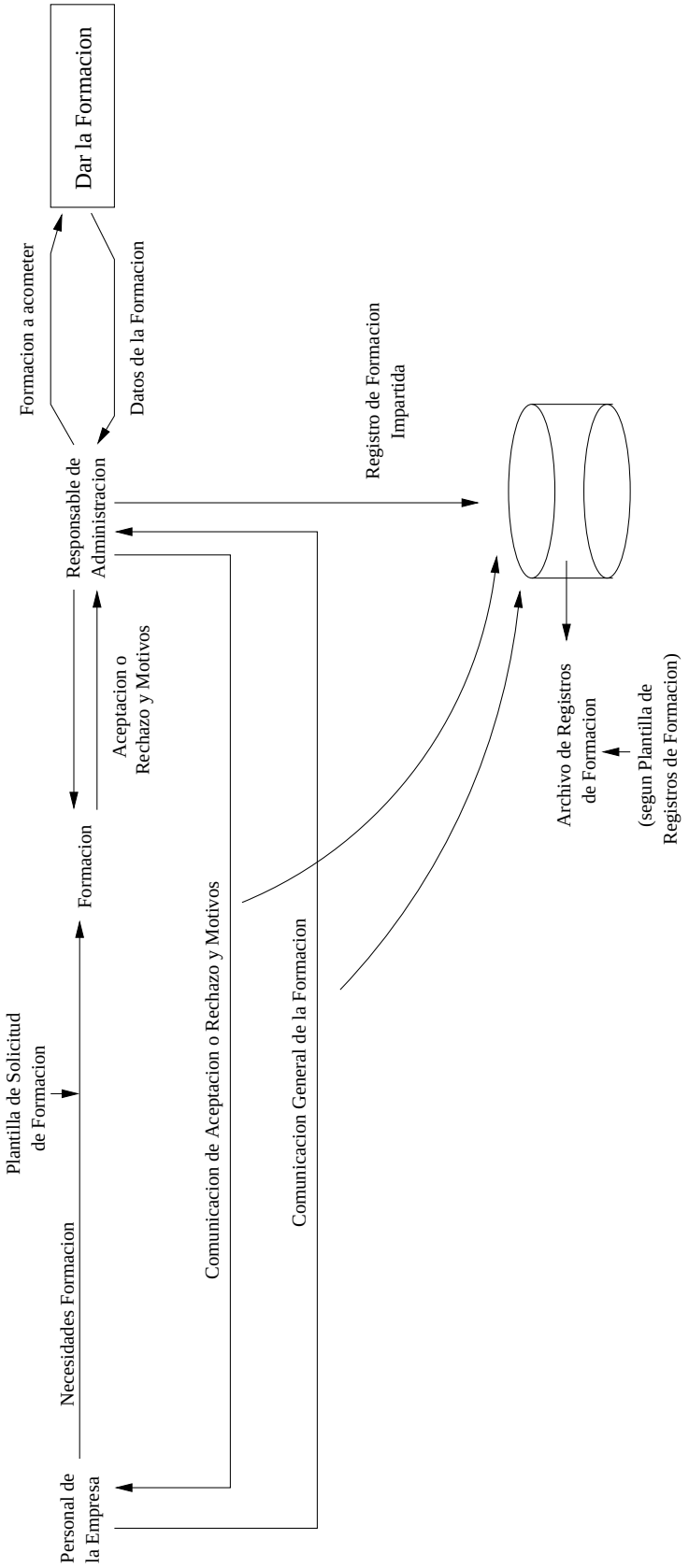


Figura A.15: Formación del Personal.

A.2.12. Compras

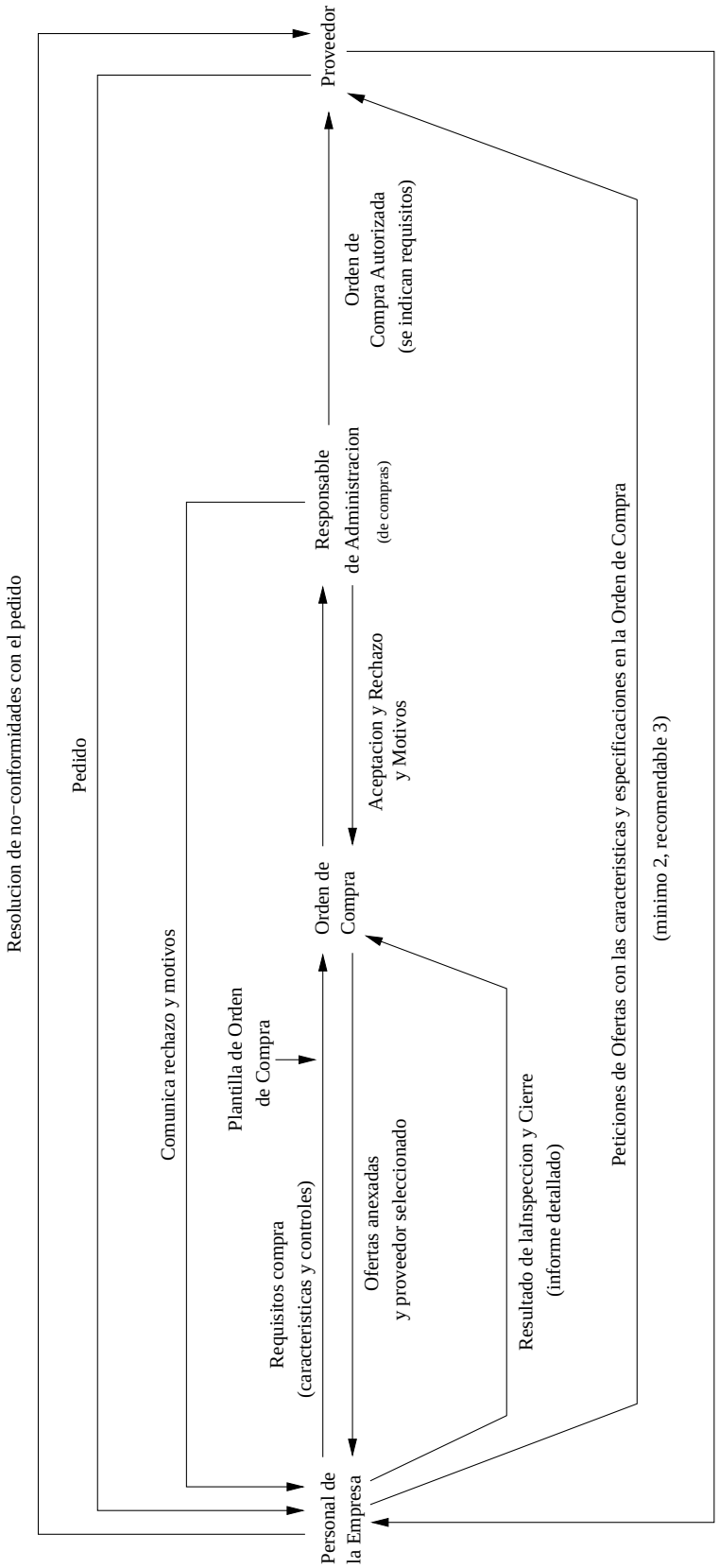


Figura A.16: Compras.

A.2.13. Control de la Documentación

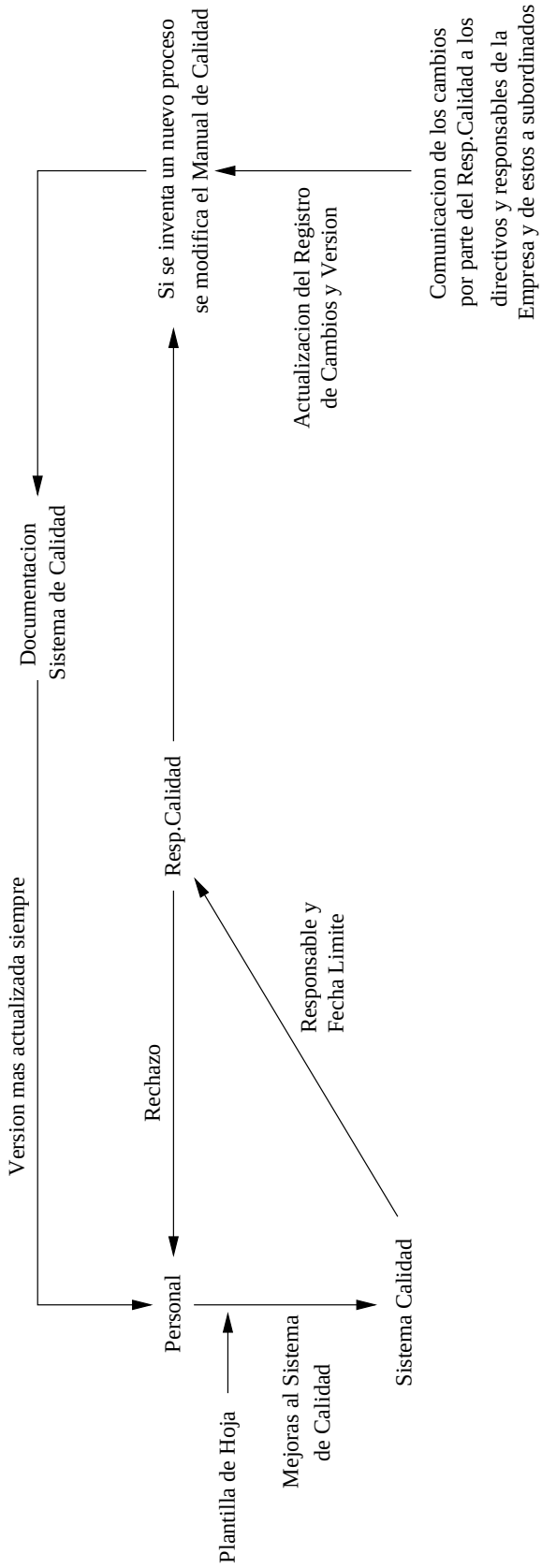


Figura A.17: Control de la Documentación.

A.2.14. Control de las Revisiones

Lo único a destacar al respecto del **Control de Revisiones** es que se debe especificar si tiene que haber reunión física o no de los responsables involucrados.

A.2.15. Auditorías

Distinguimos dos tipos de auditorías: **Auditorías internas** o de *verificación de fase* (que pueden incorporarse como parte de un Ciclo de Vida) y **Auditorías generales** o de *área*.

En las Auditorías internas, que pueden referirse a un proyecto o bien a un área de la empresa, se analiza la documentación, se llega a un consenso, se confecciona una hoja de acciones correctivas y un informe de verificación de fase. Una copia se dirige al Jefe de Proyecto, otra al Responsable de Calidad para que tomen las acciones pertinentes de acuerdo con los resultados recopilados. Será el propio Responsable de Calidad quien realice el cierre.

En cuanto a las Auditorías generales, al contrario que las anteriores no están planificadas y se comunican con suficiente antelación. Si se detectan no-conformidades se cubren las correspondientes hojas de acciones correctivas, indicando qué personas y en qué fechas deben llevar a cabo los cambios necesarios. Con todo ello, se redacta y se firma un informe.

Auditorías internas

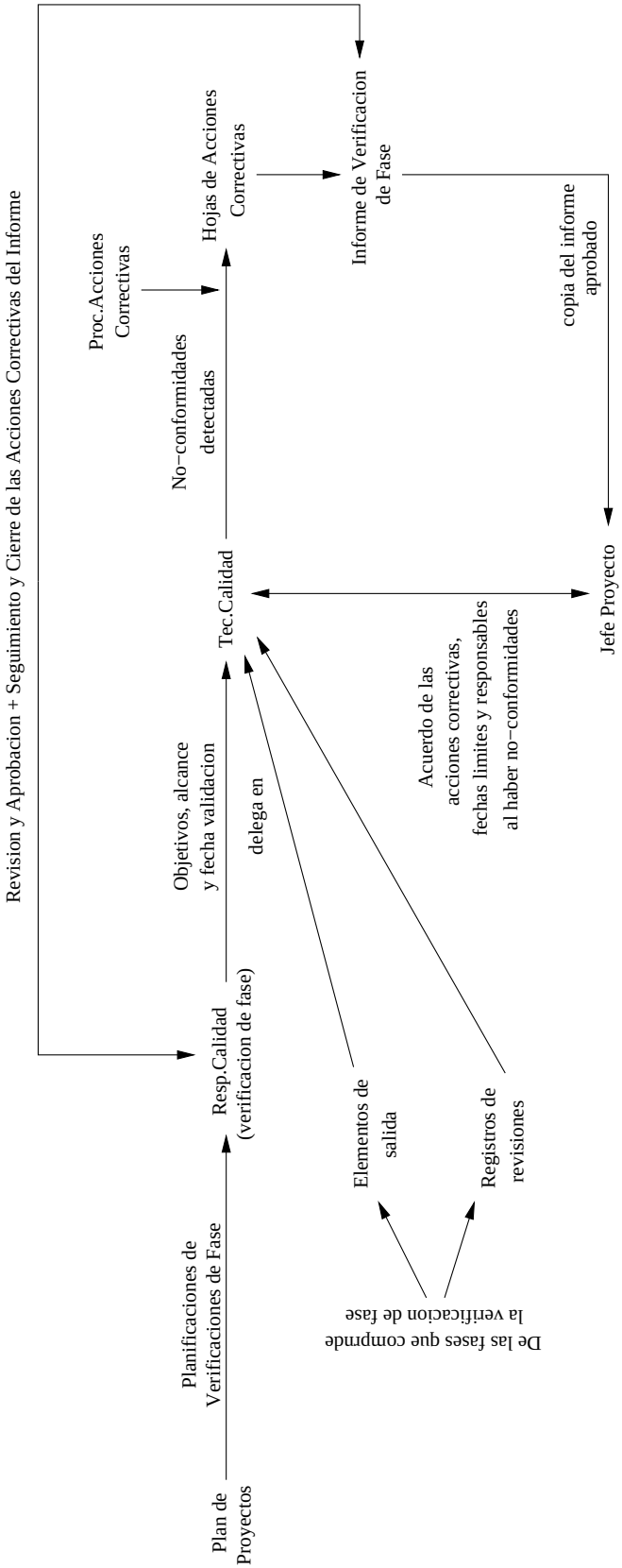


Figura A.18: Auditorías internas.

Auditorías generales

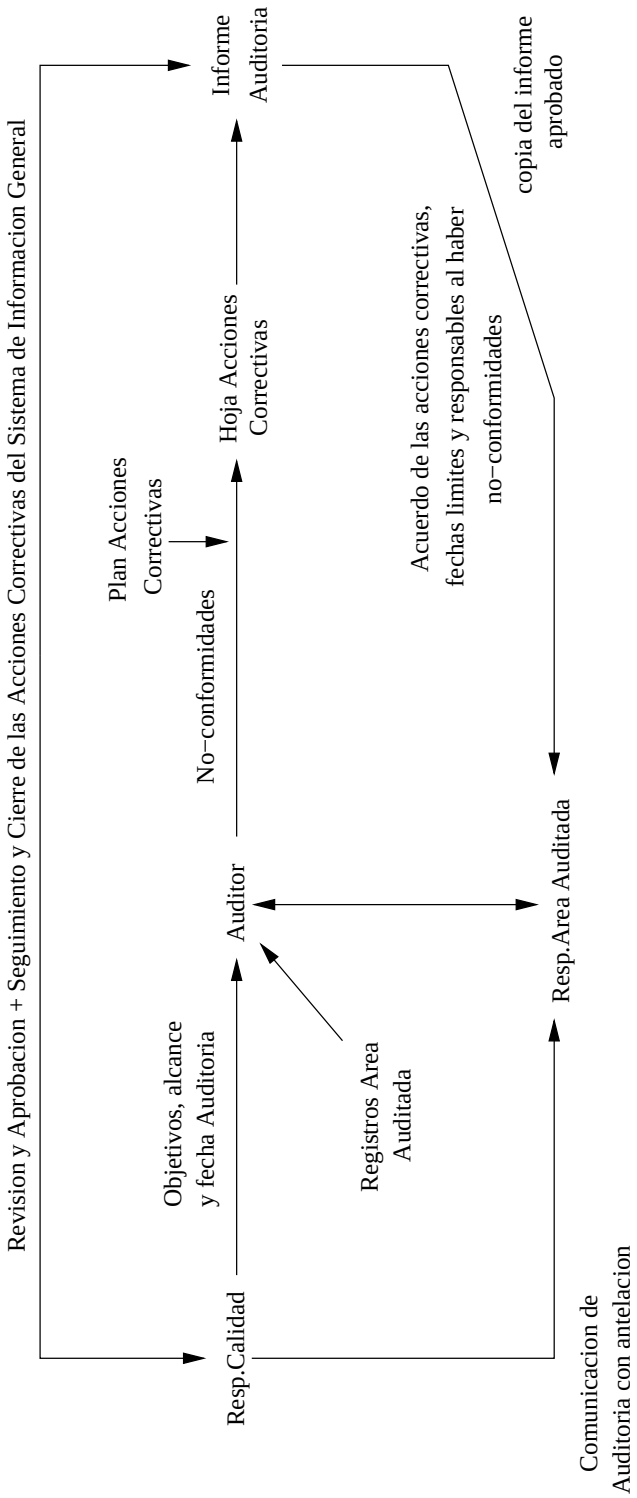


Figura A.19: Auditorías generales.

A.2.16. Métrica

Por **Métrica** entendemos las estadísticas en Ingeniería del Software. Existen varias métricas que se proponen:

- √ Tanto por ciento (%) de desvío en plazo de un proyecto
- √ Tanto por ciento (%) de desvío en coste de un proyecto
- √ Número de errores en las Pruebas de Sistema y Aceptación
- √ Número de errores notificados por el cliente (servicio Post-Venta)

Para cualquiera de ellas que sea elegida deberá definirse:

- ↪ *Alcance* (dónde se aplica)
- ↪ *Fórmula* (cómo se calcula)
- ↪ *Descripción* (qué es)
- ↪ *Frecuencia* (cada cuánto se obtiene)
- ↪ *Presentación gráfica* (cómo se presenta)
- ↪ *Responsable* (quién la obtiene)
- ↪ *Destinatario* (a quién se le presenta)

Lo visto en este apéndice es un conjunto mínimo de acciones imprescindibles para conseguir la certificación ISO 9000. A ellas hay que añadir las tratadas en distintos capítulos de este mismo documento, a saber:

- Ciclo de Vida (más detalladamente)
- Gestión de Proyectos (asociado con Gestión de Riesgos)
- Gestión de la Configuración
- Gestión de Requisitos

A.3. Gestión de Proyectos en una PYME

A.3.1. Inicio y Planificación

Para proyectos de alcance mayor a 20 días×hombre, la gestión de un proyecto tiene las etapas y desarrollo que veremos a continuación. En otro caso, por simplicidad, puede utilizarse simplemente una hoja de cálculo.

El inicio de un proyecto parte de la iniciativa y visto bueno de un *Director Técnico*. Esto es así en el caso de las PYMEs; en empresas de mayor embergadura esta responsabilidad recaerá en una *Comisión de Seguimiento* u organismo similar.

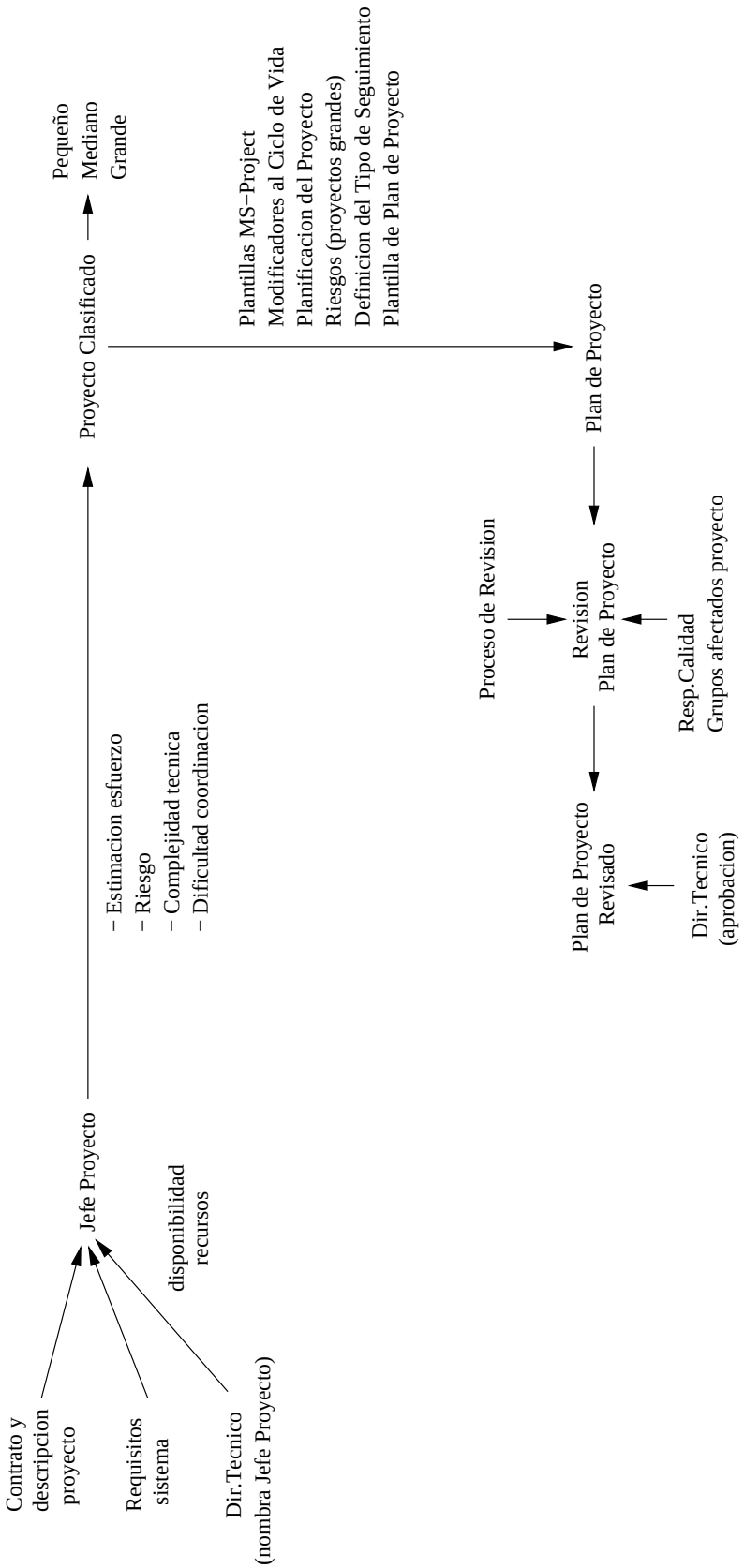


Figura A.20: Inicio y Planificación de un proyecto.

A.3.2. Seguimiento

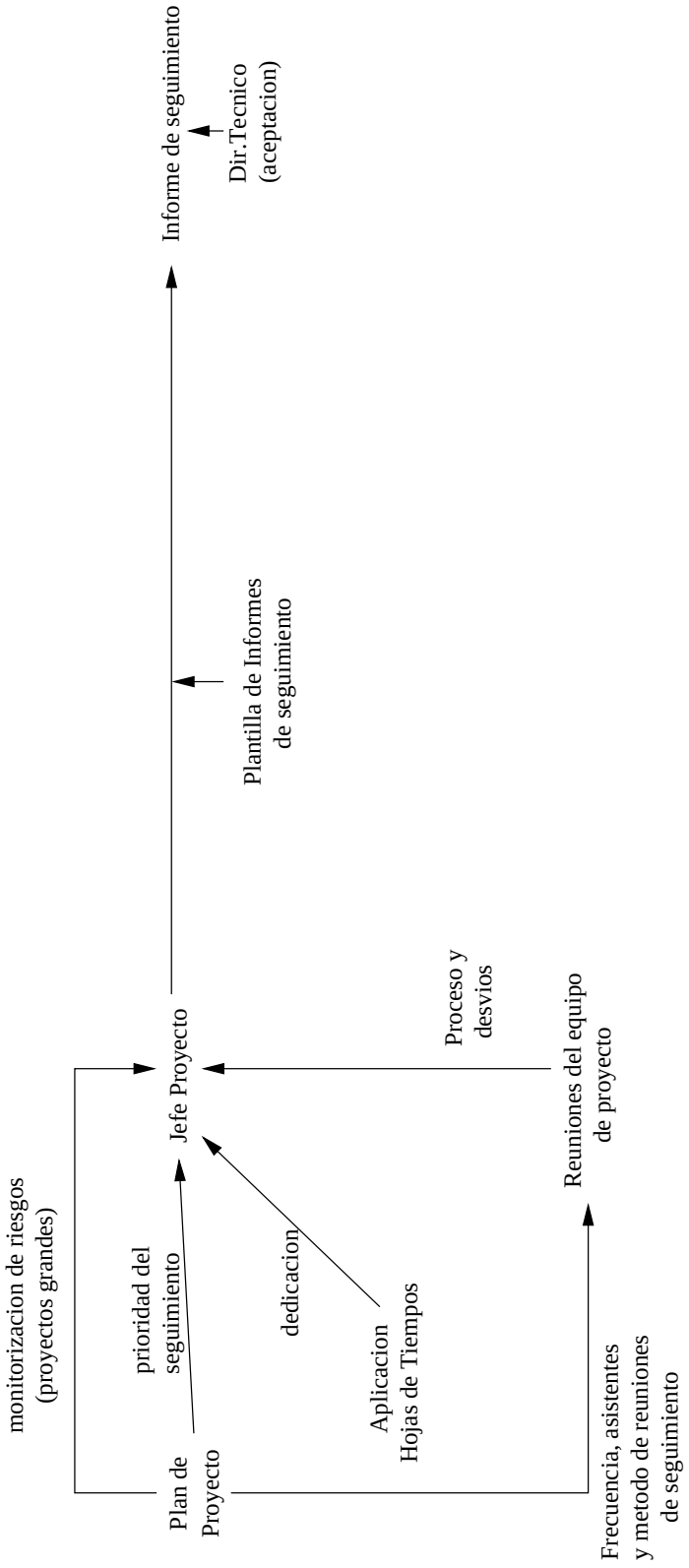


Figura A.21: Seguimiento de un proyecto.

A.3.3. Cierre

La presencia del *Responsable de Calidad* es clave en el cierre de todo proyecto, por la importancia de hacer constar el aspecto de *Lecciones Aprendidas*, tal y como indica la norma ISO.

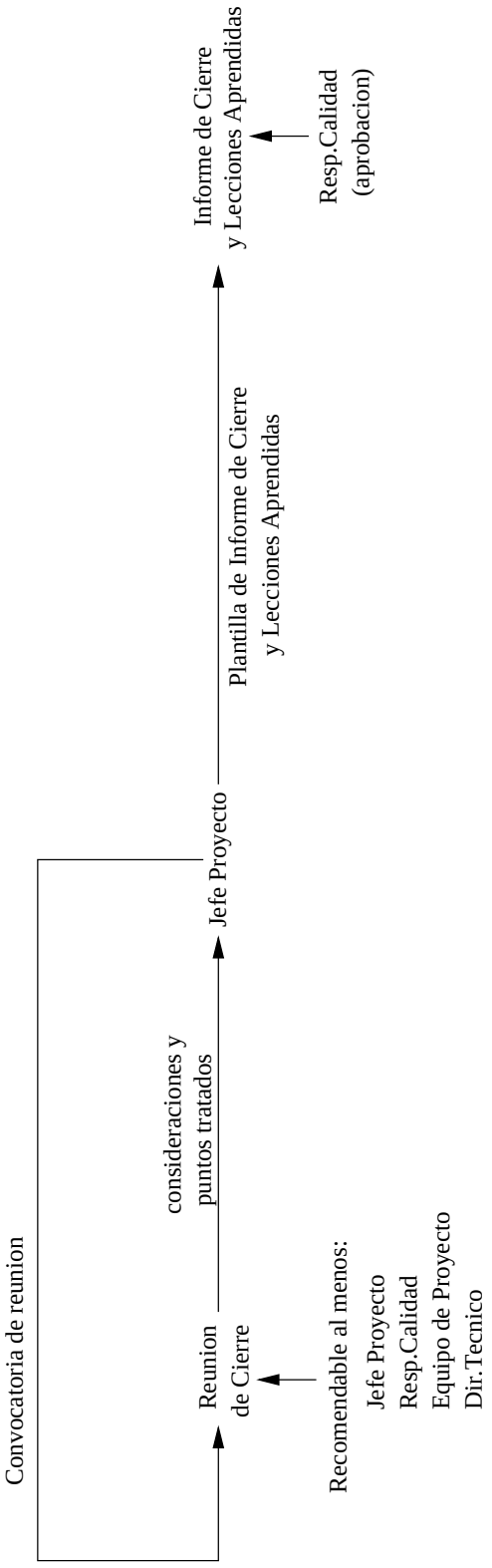


Figura A.22: Cierre de un proyecto.

A.4. Gestión de la Configuración del Software en una PYME

A.4.1. Gestión de la Configuración

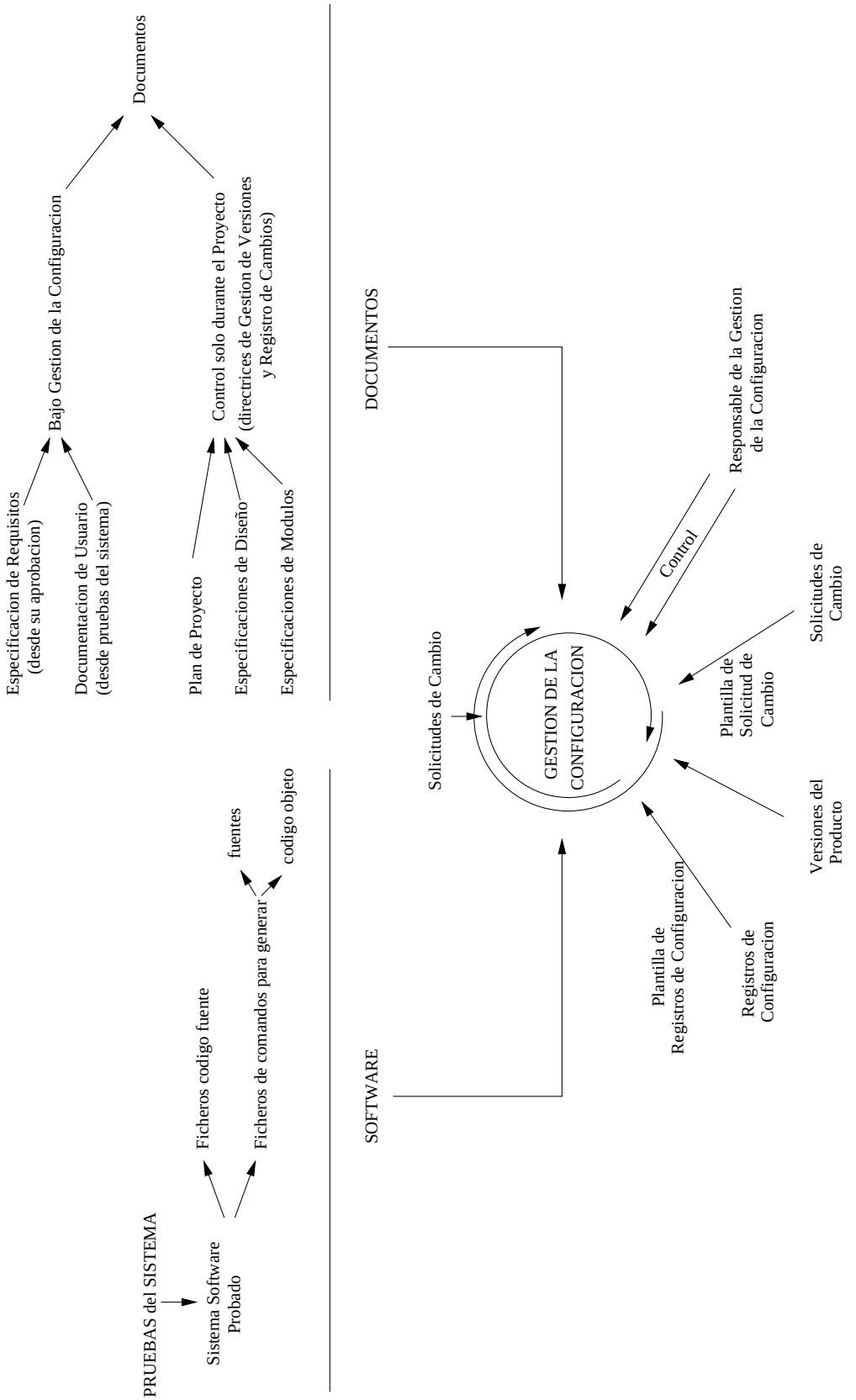


Figura A.23: Esquema general de Gestión de la Configuración del Software.

A.4.2. Gestión de Cambios

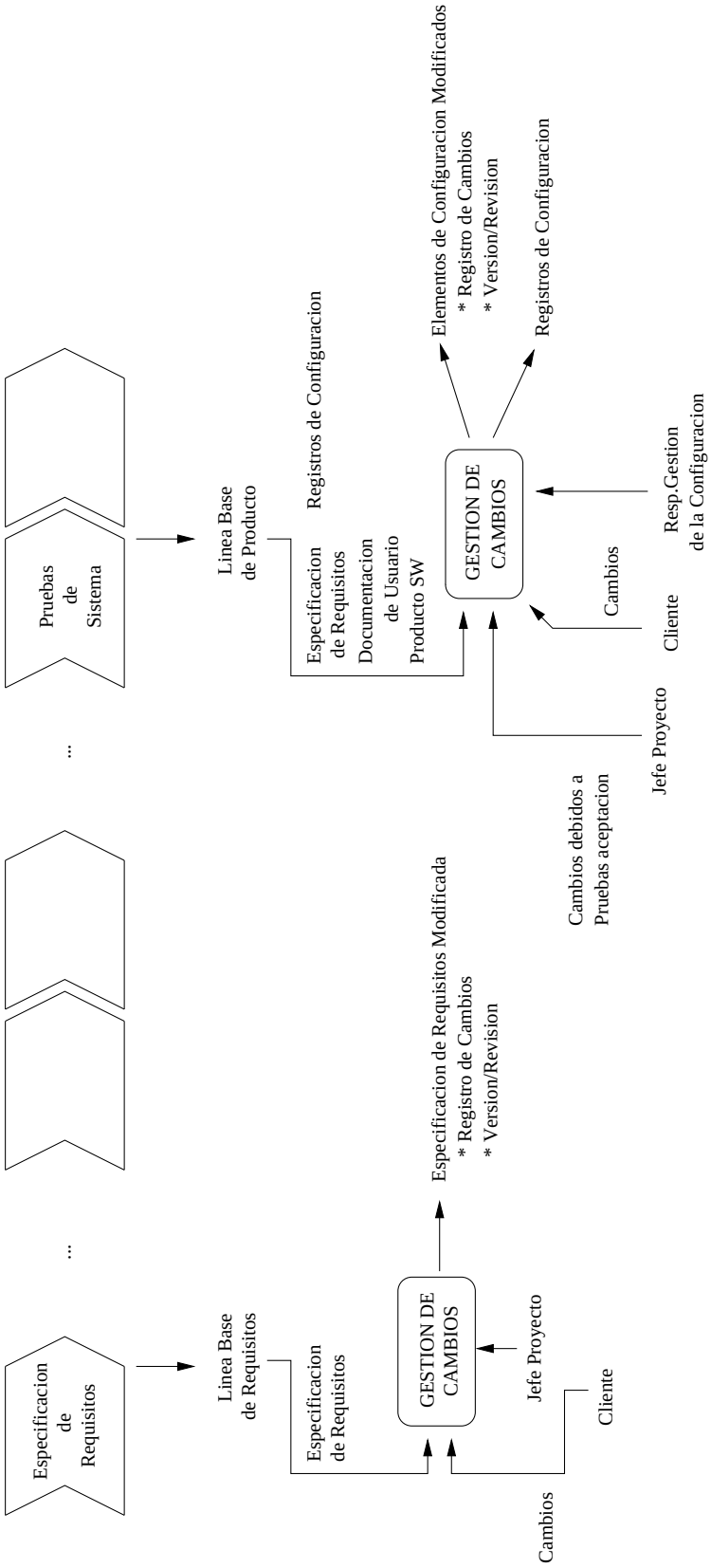


Figura A.24: Gestión de cambios en GCS.

A.4.3. Gestión y Realización de Cambios.

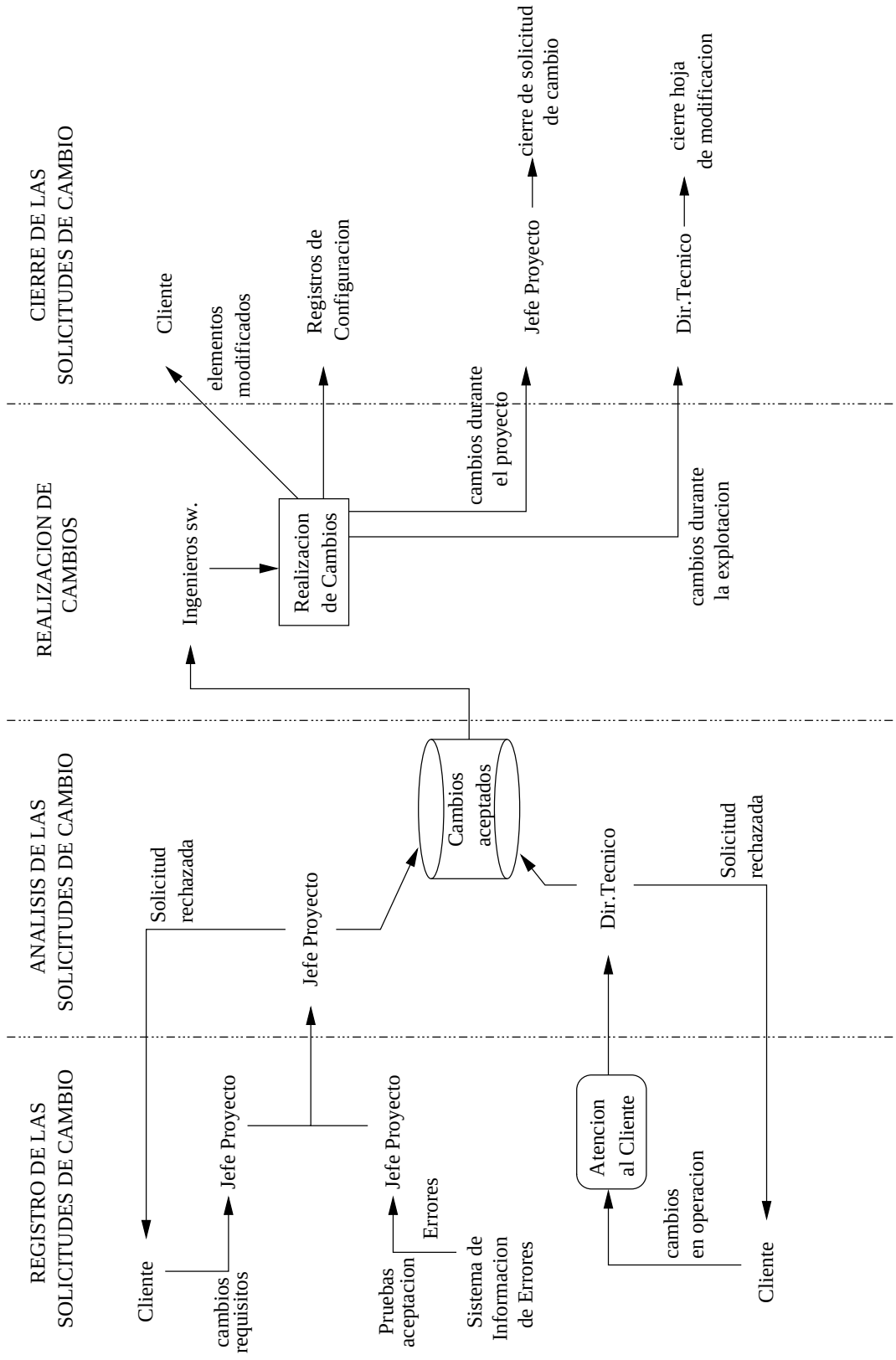


Figura A.25: Gestión y tratamiento de cambios en GCS.

Índice de cuadros

Índice de figuras

Bibliografía

[1] *Software Engineering.*

[2] AENOR. *Normas International Software Organization 9000.*

[3] Andrade, Javier. *Apuntes de clase.*

[4] H. Kan. *Metrics and Models in Software Quality Engineering.* Addisson-Wesley, 1995.

[5] Kehoc, Rajarond and Jarvic, Allov. *ISO-9000-3. A tool for software product and process improvement.* Springer.

[6] Ramón López Cortijo. *Aspectos de Gestión (I).*

[7] R.S. Pressman. *Ingen ería ded Software. Un enfoque práctico.* McGraw-Hill. 4^a edición.

[8] Lloyd's Register. *Cursos de Formación A.I.C.*

Índice alfabético