ADMINISTRACIÓN DE PROYECTOS

Este material tiene como principal objetivo apoyar el curso trimestral de “Administración de Proyectos”. Comúnmente, cuando los alumnos inician este curso, ya han aprendido y puesto en práctica diversos paradigmas de desarrollo de software. Además, han profundizado en el estudio de análisis de requerimientos y de las estrategias de calidad y pruebas más comunes para procurar el éxito de un proyecto. Sin embargo, les falta aún integrar todos estos conceptos y verlos desde la perspectiva de la administración del proyecto en su conjunto.

En la vida laboral es común, que la presión por entregar el producto a como de lugar y en una fecha límite, impida finalizar con éxito un proyecto. Los conocimientos sobre la Administración de Proyectos podrán ser de gran ayuda cuando el futuro ingeniero de software se enfrente con estos problemas.

MÉXICO, D.F. 2012

Dra. María del Carmen Gómez Fuentes

Dr. Jorge Cervantes Ojeda

Dr. Pedro Pablo González Pérez

UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD CUAJIMALPA

Casa abierta al tiempo

9 786074 778243

ADMINISTRACIÓN DE PROYECTOS — *Notas del curso*

UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD CUAJIMALPA
Casa abierta al tiempo

Dra. María del Carmen Gómez Fuentes

Dr. Jorge Cervantes Ojeda

Dr. Pedro Pablo González Pérez

Notas del curso

ADMINISTRACIÓN DE PROYECTOS

UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD CUAJIMALPA

Casa abierta al tiempo

Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD CUAJIMALPA

DIVISIÓN DE CIENCIAS NATURALES E INGENIERÍA

MATERIAL DIDÁCTICO

NOTAS DEL CURSO ADMINISTRACIÓN DE PROYECTOS

AUTORES:
Dra. María del Carmen Gómez Fuentes
Dr. Jorge Cervantes Ojeda
Dr. Pedro Pablo González Pérez

Departamento de Matemáticas Aplicadas y Sistemas

IBSN: 978-607-477-824-3

2012

U N I V E R S I D A D A U T Ó N O M A
M E T R O P O L I T A N A
Unidad Cuajimalpa

Notas del curso:

Administración de Proyectos

Dra. María del Carmen Gómez Fuentes.
Dr. Jorge Cervantes Ojeda.
Dr. Pedro Pablo González Pérez.

2012

Editores
María del Carmen Gómez Fuentes
Jorge Cervantes Ojeda
Pedro Pablo González Pérez

Departamento de Matemáticas Aplicadas y Sistemas.
División de Ciencias Naturales e Ingeniería
Universidad Autónoma Metropolitana, Unidad Cuajimalpa

Editada por:
UNIVERSIDAD AUTONOMA METROPOLITANA
Prolongación Canal de Miramontes 3855,
Quinto Piso, Col. Ex Hacienda de San Juan de Dios,
Del. Tlalpan, C.P. 14787, México D.F.

NOTAS DEL CURSO: ADMINISTRACIÓN DE PROYECTOS

Primera edición 2012
Segunda edición

ISBN: 978-607-477-824-3

Impreso en México
Impreso por Publidisa Mexicana S. A. de C.V.
Calz. Chabacano No. 69, Planta Alta
Col. Asturias C.P.

Contenido

Introducción

Estas notas tienen el objetivo de apoyar el curso trimestral de "Administración de Proyectos". Normalmente, cuando los alumnos inician este curso, ya han aprendido y puesto en práctica diversos paradigmas de desarrollo de software, como desarrollo en cascada, procesos evolutivos, reutilización de componentes o incluso el proceso unificado de desarrollo. Además han profundizado en el estudio de análisis de requerimientos y de las estrategias de calidad y pruebas más comunes para procurar el éxito de un proyecto. Sin embargo, les falta integrar todos estos conceptos y verlos desde la perspectiva de la administración del proyecto en su conjunto.

En la actualidad sigue siendo elevado el número de proyectos de software que se abandonan o fracasan, sobre todo cuando éstos son complejos e involucran miles o millones de dólares para su realización. En la mayoría de los casos, el fracaso se debe a que el tiempo utilizado para el desarrollo del proyecto hace que éste se convierta en *no viable*. Se han hecho estudios acerca de los fracasos en los proyectos de software, por ejemplo [Mangione, 2003] y [McManus & Wood, 2004]. El fracaso de proyectos de software algunas veces ha implicado la pérdida de muchísimo dinero o incluso la pérdida de vidas humanas por entregar productos defectuosos [Pfleger & Atlee, 2002; Weitzenfeld, 2004]. Es común que algunos desarrolladores de software piensen que tienen demasiado trabajo como para dedicar un tiempo a aprender métodos de trabajo eficaces que ayuden a resolver la mayoría de los problemas relacionados con el tiempo de desarrollo, sin embargo, mientras continúen sin aprender estos métodos, no dejarán de trabajar a marchas forzadas ni tendrán tiempo suficiente para su vida personal [McConnell, 1997].

Es importante que el futuro ingeniero de software esté capacitado en la "Administración de Proyectos" antes de que se integre por completo a la vida laboral, donde es común, que la presión por entregar el producto a como de lugar y en una fecha límite, impida finalizar con éxito un proyecto. El alumno debe saber que seguir un buen proceso de desarrollo de software, aplicando correctamente las normas de calidad, no garantiza que un proyecto se entregará trabajando correctamente y además en el tiempo estimado, ya que existen otros aspectos importantes que se deben tomar en cuenta.

El trabajo organizado de la siguiente manera. En el capítulo I se da una visión global de los aspectos más importantes de la administración de un proyecto. El capítulo II aborda los diferentes aspectos a tomar en cuenta al trabajar con el personal que participa en el proyecto. La forma de estudiar los problemas para evaluar si éstos deben o no, dar origen a un nuevo proyecto, se expone en el capítulo III. En el capítulo IV se estudia en detalle el

proceso de la administración de un proyecto. El capítulo V es una introducción a los temas relacionados con la planeación de un proyecto. El capítulo VI está dedicado a la administración basada en riesgos. En el capítulo VII se mencionan algunas de las herramientas existentes para aumentar la productividad y finalmente, el capítulo VIII es una introducción al tema de la recuperación de proyectos.

Objetivos

Que al final del curso el alumno sea capaz de:

1. Comprender y ubicar la importancia de los elementos personal, problema y proceso en la administración de proyectos de software.
2. Conocer los estándares, métodos, técnicas y herramientas para la administración de proyectos de software.
3. Organizar equipos eficaces, motivados y coordinados.
4. Obtener del cliente una especificación clara y detallada de los objetivos que éste persigue.
5. Adaptar el proceso de desarrollo a las necesidades del cliente y a las peculiaridades del personal.
6. Administrar un proyecto de software.

Capítulo I El espectro de la administración de proyectos.

I.1 ¿Qué significa la administración de proyectos?

Administrar un proyecto consiste en planificar y dar seguimiento a los proyectos de desarrollo de software utilizando los recursos necesarios para realizar el proyecto en el menor tiempo posible y con un mínimo número de fallas. Esto no es fácil, ya que en la práctica se tienen limitaciones como son un número reducido de mano de obra, falta de capacitación de los recursos humanos disponibles, equipo de cómputo insuficiente o inadecuado, etc. Para logar el éxito de un proyecto es necesario ayudarse con conocimientos, habilidades, herramientas y técnicas que estudiaremos a lo largo de este curso.

Para medir el éxito de un proyecto se toma en cuenta que los objetivos planteados se logren en el tiempo previsto y con el presupuesto asignado.

Según [Briseño, 2003] los objetivos de la administración de proyectos son principalmente:

- Terminar a tiempo.
- Dentro del presupuesto.
- Cumpliendo con los requerimientos.

Mientras que los elementos a administrar son: cliente, calidad, recursos, riesgos, comunicaciones, contrato y finanzas.

La administración de un proyecto consiste en:

- Comunicar a las personas lo que deben hacer y cuándo entregar resultados.
- Organizar el trabajo: dividirlo y programarlo en el tiempo.
- Supervisar todo el proceso para saber si se están obteniendo los resultados esperados.

La administración exitosa de un proyecto requiere tomar en cuenta los siguientes cuatro factores claves:

1. El personal que intervendrá.
2. El producto que se entregará.
3. El proceso que se aplicará.
4. La tecnología que se va a utilizar.

Comenzaremos con una exposición sobre *el personal* que interviene en un proyecto, este incluye a las personas que aprueban el proyecto, a los líderes o responsables del proyecto y al equipo de desarrolladores de software.

I.2 El personal

Cualquier organización que trate de mejorar su productividad de una manera seria y efectiva debe ocuparse en primer lugar, de los temas relacionados con el personal, como: la motivación, el equipo de trabajo, la selección y la formación del mismo. Antes de comenzar con estos temas, es interesante mencionar quienes son las personas que normalmente toman la decisión de aprobar un proyecto de software.

I.2.1 Los que proponen el proyecto

La revisión de propuestas para proyectos se lleva a cabo en la mayoría de las organizaciones por *los comités directivos*. En general este tipo de comités esta conformado por:

1. *Miembros de alto nivel administrativo*: como el vicepresidente ejecutivo, o el vicepresidente de producción.
2. *Gerentes departamentales*: como gerente de ventas y mercadotecnia o gerente del departamento de crédito.
3. *Gerentes técnicos*: como el gerente de investigación y desarrollo o el coordinador de control de calidad.
4. *Grupo de sistemas de información*: como el gerente de procesamiento de datos, o el jefe analista de sistemas.

Es importante notar que este grupo no está dominado por los especialistas en sistemas de software, que son los que realmente tienen una idea del esfuerzo que significa la elaboración del proyecto.

El comité recibe las propuestas y las evalúa. La mayor responsabilidad del comité es tomar una decisión, y con frecuencia ésta requiere de más información que la contenida en la propuesta. Por consiguiente, a menudo se solicita más información para reunir elementos suficientes. Las decisiones se toman con base a los costos del proyecto, su beneficio para la organización y la factibilidad de llevarlos a cabo dentro de los límites de la tecnología con la que cuenta la empresa.

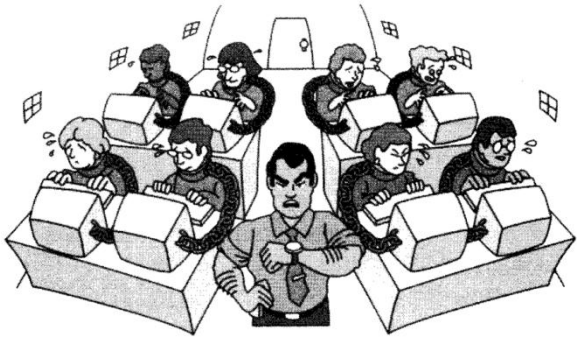
I.2.2 Selección del personal para los equipos del proyecto

La selección de las personas que van a conformar el equipo de desarrollo del proyecto implica otras habilidades por parte del administrador del proyecto que van más allá de su capacitación técnica. Podría creerse que una sólida formación científica y tecnológica garantiza la capacidad para coordinar y llevar a buen término un proyecto de este tipo. Sin embargo se ha comprobado que el administrador de un proyecto requiere también de habilidades que tienen que ver con su capacidad de establecer buenas relaciones interpersonales y con su conocimiento de técnicas de liderazgo [Robbins, 1996] junto con la facultad de detectar el talento y el tipo de personas con las que se cuenta para hacer un equipo de trabajo. Según McConell, [Bohem, 1981] presenta los siguientes cinco principios para la selección del personal que participará en un proyecto de software:

1. *Máximo talento*.- Significa usar poco y buen personal.
2. *Trabajo adecuado*.- Asignar tareas según la habilidad y motivación de la gente disponible.
3. *Progresión profesional*.- Ayudar a la gente a actualizarse por sí misma en vez de obligarles a trabajar donde más experiencia tienen o donde son más necesarios.
4. *Equilibrar el equipo*.- Seleccionar a personas que se complementen entre sí y armonicen con los demás.
5. *Eliminar la inadaptación*.- Aunque esto puede parecer muy duro, para que un proyecto tenga éxito será necesario eliminar y remplazar a lo miembros problemáticos del equipo lo antes posible.

I.2.3 La motivación del personal

La motivación es potencialmente el aliado más fuerte para el desarrollo del proyecto, una persona desmotivada es una persona apática que no entregará nunca los mismos resultados que una persona motivada.



(McConnell 1997)

Figura1.1: Es muy importante no confundir *motivación* con *represión*.

Cuando un coordinador no tiene el talento suficiente para manejar a las personas puede recurrir a argumentos represivos, como amenazas de cualquier tipo, las amenazas elevan el stress de las personas lo que a su vez las hace más propensas a cometer errores. Otro error común es hacer que las personas se sientan incapaces por no haber cumplido con algún objetivo en el plazo establecido. Desvalorizar a las personas baja su estado de ánimo, lo que trae como consecuencia su falta de interés por el trabajo.

Por otra parte, la motivación genera un alto rendimiento y la obtención de resultados, incluso mejores a las expectativas que el coordinador se había formado. La motivación no es una tarea fácil para el administrador del proyecto ya que no existe una "fórmula mágica" para motivar a los empleados, pues cada cabeza es un mundo y lo que funciona bien para un tipo de personas podría resultar contraproducente para otras. Es aquí donde se puede observar la complejidad del problema al que se enfrenta un jefe que debe lidiar con personas para que entre todos logren un objetivo común, que en el caso que nos concierne es la entrega de un producto de software funcionando correctamente y en el plazo establecido.

Si bien no hay fórmulas mágicas, el sentido común nos indica que hay ciertos puntos que motivan a cualquier ser humano, cabe señalar que el arte de dirigir un proyecto consiste en encontrar la forma de lograr:

- Que la persona se sienta útil y capaz de lograr el trabajo que se le ha encomendado.
- Que la persona sienta que su trabajo es reconocido y apreciado por los demás.
- Que la persona tenga interés por seguir aprendiendo cosas nuevas y seguirse superando.
- Que la persona esté dispuesta a cooperar con los demás miembros del equipo, a aceptar comentarios constructivos y a compartir sus logros y conocimientos sin sentirse amenazado.

En el capítulo II trataremos de una manera más amplia los diferentes aspectos que hay que tomar en cuenta acerca del personal cuando se requiere administrar un proyecto.

I.3 El problema.

Desde un primer punto de vista, el problema es aquel que resuelve el proyecto, desde este punto, podemos distinguir tres razones principales para proponer un proyecto [Senn, 1992]:

1. *Resolver un problema*: actividades, procesos o funciones que no satisfacen los estándares de desempeño o las expectativas, por lo que es necesario emprender una acción que resuelva las dificultades, por ejemplo, disminuir el número excesivo de errores en los datos de entrada.

2. *Aprovechar una oportunidad*: un cambio para mejorar el rendimiento económico de la empresa y su competitividad, por ejemplo: un nuevo programa con mayor numero de vuelos directos y descuentos en el precio del pasaje.
3. *Dar respuesta a directivos*: proporcionar información en respuesta a órdenes, solicitudes o mandatos originados por una autoridad legislativa o administrativa; llevar a cabo tareas de cierta manera, o también cambiar la información.

Desde un segundo punto de vista, el problema puede ser visto como *la administración del proyecto* en sí, ya que esto representa todo un reto para el responsable del proyecto.

Para la administración correcta es necesario poseer las siguientes habilidades:

- Liderazgo.
- Comunicación.
- Negociación.
- Solución de problemas.
- Capacidad de síntesis para el logro de objetivos.

Además, son necesarios conocimientos técnicos y administrativos que permitan al administrador manejar las herramientas y técnicas necesarias.

En la *figura 1.2* se muestra un diagrama de las entradas y salidas cuando se tiene una administración precisa en un proyecto de software, mientras que en la *figura 1.3* se representan los problemas que surgen cuando la planificación resulta ser demasiado optimista.

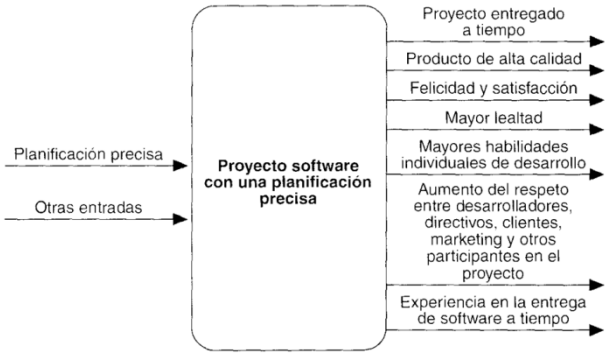


Figura 1.2: Diagrama de un proyecto con una planificación precisa (McConnell, 1997).

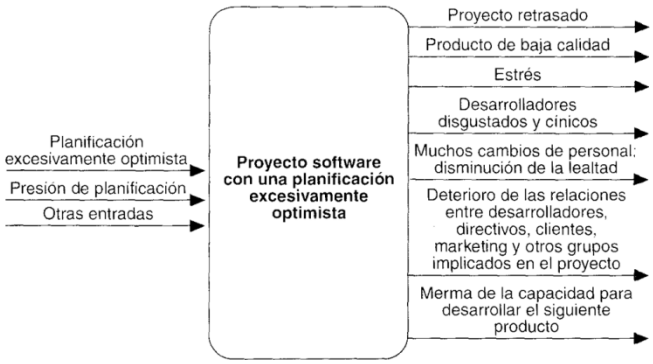


Figura 1.3: Diagrama de un proyecto con una planificación excesivamente optimista (McConnell, 1997).

En el capítulo III se aborda nuevamente el tema del problema con mayor profundidad.

I.4 El proceso de la administración de un proyecto.

La administración de proyectos tiene tres objetivos fundamentales:

- Terminar a tiempo.
- Dentro del presupuesto.
- Cumplir con los requerimientos.

Además, es necesario administrar correctamente todos los factores que intervienen en el desarrollo de un proyecto, los principales son: cliente, calidad, recursos, riesgos, comunicaciones, contrato y finanzas.

Para lograr los tres objetivos fundamentales, tomando en cuenta a todos los factores que intervienen, se ha establecido un *proceso de la administración proyectos* [PMI, 2008] el cual consta de las fases o etapas que se ilustran en la *figura 1.4*. Como puede observarse, los subprocesos de planeación y ejecución se retroalimentan mutuamente.

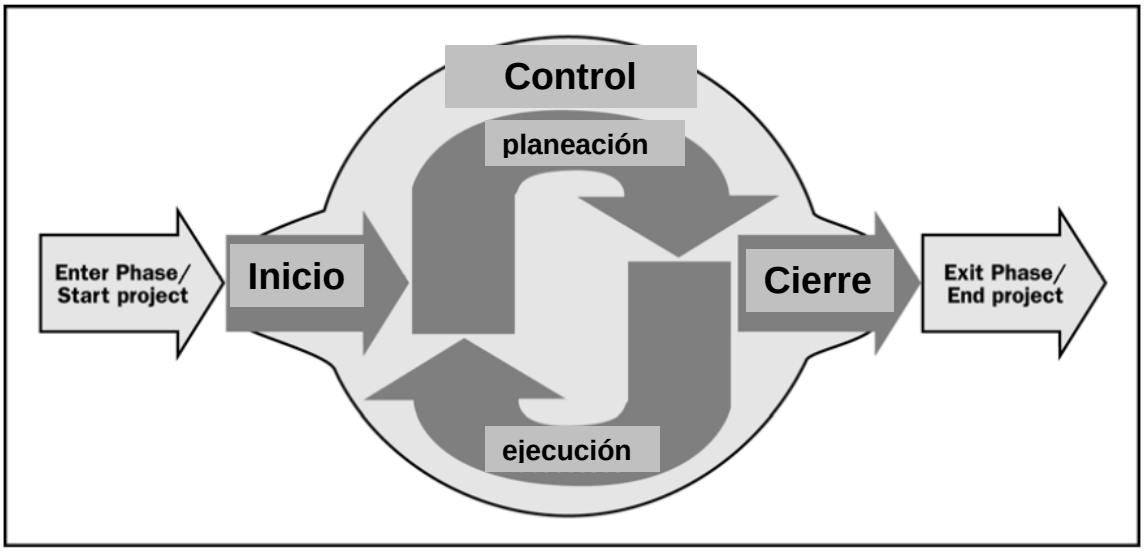


Figura 1.4: Etapas de la administración de proyectos [PMI, 2008].

Las fases o etapas permiten dividir el proyecto en subconjuntos lógicos que facilitan su dirección, planificación y control. El número de fases, la necesidad de establecerlas y el grado de control aplicado dependen del tamaño, la complejidad y el impacto potencial del proyecto. Independientemente de la cantidad de fases que compongan un proyecto, todas ellas poseen características similares. Cuando las fases son secuenciales, el cierre de una fase termina con la entrega del trabajo producido para que éste sea la entrada de la siguiente fase. La terminación de cada fase representa un punto natural para re-evaluar el esfuerzo en curso y, en caso de ser necesario, para cambiar o terminar el proyecto. Estos puntos se conocen como salidas de fase, hitos, puertas de fase, puntos de decisión, puertas de etapa o puntos de cancelación [PMI, 2008].

El Project Management Body of Knowledge, (PMBOK®) es un documento escrito y distribuido por el Project Management Institute (PMI®) de los Estados Unidos (www.pmi.org); el PMI es una organización profesional que se ha convertido en la referencia a nivel mundial para la administración de proyectos.

El inicio del proyecto incluye la definición de lo se debe lograr con el proyecto, plantear el *alcance* y la selección de los miembros iniciales del equipo. El *alcance* de un proyecto define el tamaño del proyecto, cuanto tiempo y cuantos recursos se requieren.

La planeación del proyecto consiste en: perfeccionar el alcance, hacer un listado de tareas y actividades para lograr las metas, definir una secuencia de actividades, desarrollar un calendario y elaborar un presupuesto. El plan del proyecto debe aprobarse según las normas de calidad establecidas antes de proceder con la siguiente etapa.

La ejecución del proyecto incluye: dirigir al equipo, comunicarse con el cliente, proveedores y demás externos, resolver conflictos y asegurar los recursos necesarios (dinero, personal, equipo y tiempo).

El cierre del proyecto contempla una serie de actividades, tales como:

- reconocimiento de logros y resultados,
- cierre de las actividades y dispersión del equipo,
- aprendizaje de la experiencia del proyecto,
- revisión del proceso y resultados, redacción del informe final,
- auditorías.

El control se lleva a cabo a lo largo de toda la administración del proyecto, las actividades que corresponden al *control* del proyecto son las siguientes:

- Vigilar las desviaciones del plan.
- Acciones correctivas.
- Recibir y evaluar cambios solicitados.
- Cambiar calendarios.
- Adaptar recursos.
- Regresar a la etapa de planeación para hacer ajustes.
- Control de costos.
- Control de calidad.
- Informes de resultados.
- Comunicación con los interesados.

En el capítulo IV se estudia en detalle el proceso de la administración de un proyecto.

I.5 Planeación, riesgos y herramientas.

La planeación o planificación de un proyecto comprende lo siguiente: desarrollar un plan para la dirección del proyecto, recopilar los requerimientos, definir el alcance del proyecto, crear la estructura de desglose del trabajo, definir las actividades y darles una secuencia, hacer una estimación de los recursos necesarios para llevar a cabo las actividades del proyecto y una estimación de la duración de cada actividad, desarrollar un cronograma, estimar los costos del proyecto, determinar el presupuesto, planificar la calidad, los recursos humanos, las comunicaciones, la gestión de riesgos y las adquisiciones. En el capítulo V se da un panorama general de lo que es la planeación de un proyecto y la introducción al modelo de estimación de costos llamado COCOMO.

El desarrollo de proyectos complejos de software lleva implícita la existencia de ciertos riesgos que si no se toman en cuenta y se analizan con cuidado, retrasarán considerablemente la entrega del producto o incluso podrían llegar a causar la cancelación

del proyecto. En el capítulo VI se abordan el proceso de gestión de riesgos, su estimación, y su control.

Existen herramientas para aumentar la productividad durante el proceso de la administración de proyectos, éstas se concibieron con el fin de automatizar o apoyar una o más fases del ciclo de vida del desarrollo de sistemas de software. En el capítulo VII se abordan estas herramientas.

Capítulo II El personal.

II.1 Los participantes.

Antes de empezar a planear un proyecto que involucra o que da soporte ya sea a una parte o a toda una empresa en general, es necesario tener un panorama general de lo que es una empresa. Los organigramas se emplean con frecuencia para describir la forma en la que están relacionados los diferentes componentes de una organización. En la *figura 2.1* se muestra un ejemplo de organigrama.

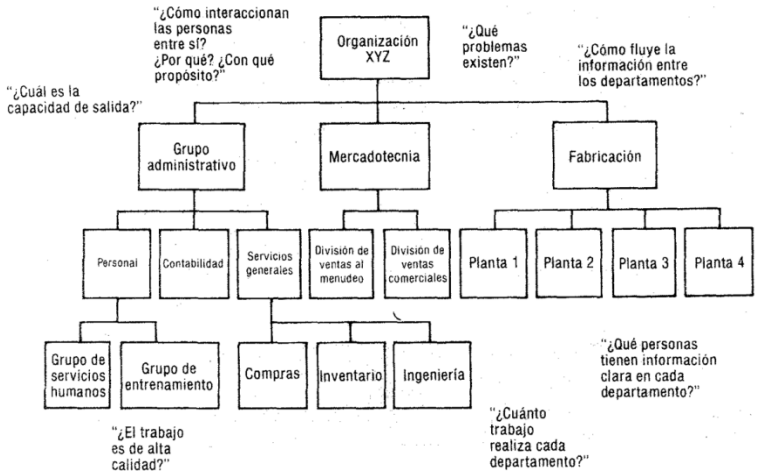


Figura 2.1: organigrama de una empresa "XYZ" de fabricación. (Senn 1992).

Como se puede observar, aunque un organigrama indica con precisión las relaciones formales entre los diferentes componentes de la organización tales como divisiones, departamentos, oficinas y empleados, no contienen información importante que es muy necesaria para el administrador del proyecto, quien debe investigar además:

- *Interacciones entre las personas y los departamentos.*
- *Interdependencias:* departamentos y otros componentes de la organización de los cuales depende un elemento en particular.
- *Personas y funciones clave*: Personas y elementos más importantes en el sistema para que éste tenga éxito.
- *Enlaces críticos de comunicación:* Determinar como es el flujo de información y de instrucciones entre los distintos componentes de la organización.

II.2 Los jefes de equipo

II.2.1 Directivos y responsables técnicos

Cuando un buen desarrollador de software lleva un tiempo considerable trabajando para una empresa llega un momento en el que sube de nivel y de pronto se enfrenta al reto de dirigir a otras personas, es así que de realizar labores cien por ciento técnicas, pasa a desempeñar labores de gestión y de dirección sobre sus compañeros de trabajo. En esta situación es muy común cometer uno o varios de los siguientes errores señalados por [McConnell, 1997]:

- Tratar de competir en conocimientos con el equipo técnico asignado.
- Atender más las labores propias que a las del grupo.
- Preocuparse más por los requerimientos técnicos que por los funcionales.
- Centrarse en continuar perfeccionando el perfil técnico descuidando otras disciplinas.
- Negarse a la necesidad de realizar labores de gestión y supervisión.
- Reportar a los responsables con excesivo nivel de detalle para demostrar nuestra capacidad de trabajo.

Es importante identificar estos problemas para así poder iniciar su solución. La capacitación sobre temas tales como gestión de costos, de riesgos, de adquisiciones, técnicas de valoración, negociación y motivación es básica cuando se pretende dirigir a un equipo. Probablemente, lo primero que aprenderá un nuevo dirigente es a incrementar su disciplina, a optimizar el tiempo y a ayudarse de sencillas técnicas y herramientas que le permitan acumular conocimiento y no repetir trabajo innecesariamente. La *figura 2.2* muestra las diferentes habilidades que deben poseer los miembros de un equipo de trabajo en función del papel que desempeñan: mandos superiores, mandos medios o niveles operativos.

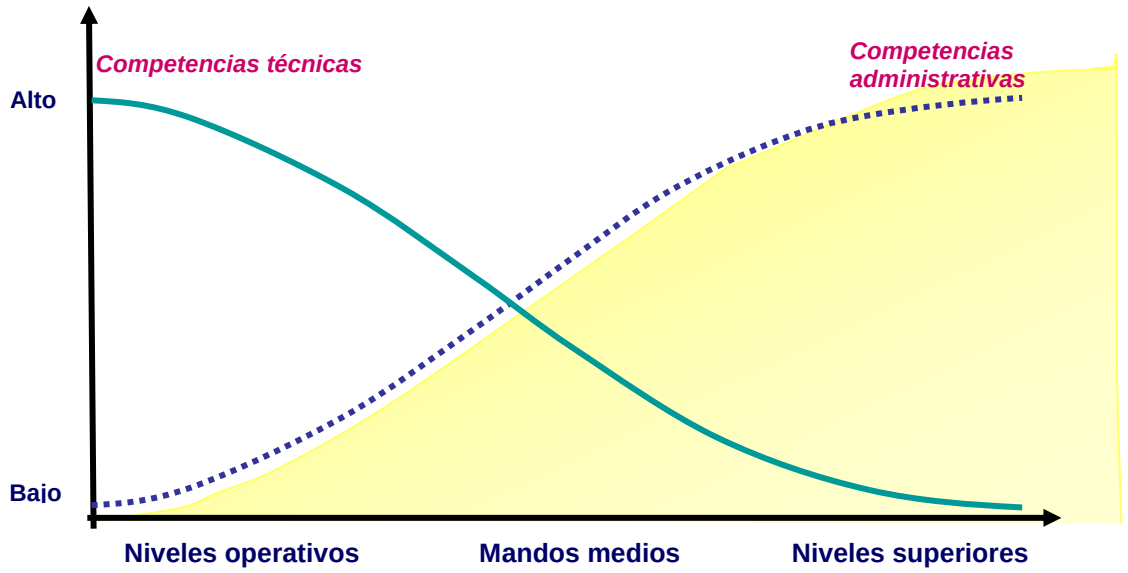


Figura 2.2: Habilidades de los equipos de trabajo (Subsecretaría de la función pública).

II.2.2 La motivación.

En el apartado I.2 se mencionó la importancia de la motivación del personal. En esta sección mencionaremos las diferentes personalidades y cual es la más común en los desarrolladores de software, cómo motivar y finalmente lo que hay que evitar para no bajar la moral de los participantes de un proyecto de software.

Existe una prueba llamada *Myers Briggs Type Indicator* (MBTI). E1 MBTI mide las preferencias de las personas en cuatro dimensiones y propone una clasificación usando letras. Las categorías son:

- Extraversión (E) o introversión (I).
- Sentido (S) o intuición (N).
- Pensamiento (T) o sentimiento (F).
- Opinión (J) o percepción (P).

En MBTI existen 16 tipos de personalidad. Sin demasiada sorpresa, dos estudios intensos han encontrado que los profesionales de las computadoras son mucho más “introvertidos” que las personas en general, según [McConnell, 1997]. *Introvertidos* en el contexto del MBTI simplemente significa que la persona está más interesada en el mundo interior de las ideas que en el mundo externo de las personas y cosas. Los desarrolladores en general están más interesados en la posibilidad de superación que los demás grupos, y menos interesados

en la posición social y el reconocimiento según dichos estudios. Los mismos estudios descubrieron que el 80% de los profesionales de las computadoras prefieren el pensamiento (T) sobre el sentimiento (F), comparados con el 50% de las personas en general. Los T prefieren tomar decisiones basadas en unas bases lógicas e impersonales más que en valores personales subjetivos. Esta inclinación lógica y planificada se refuerza por las preferencias de los profesionales de las computadoras en la opinión (J) sobre la percepción (P) (las dos terceras partes de los profesionales de las computadoras son J, comparados con la mitad de las personas en general). Los J tienden a vivir de una forma ordenada y planificada, mientras que los P tienden a ser más flexibles y adaptables.
La implicación de esta preferencia es clara: Si desea apelar a un desarrollador, será mejor utilizar argumentos lógicos. Por ejemplo, mucho de lo que se ha escrito sobre la motivación sugiere que una clave para la productividad excepcional es establecer objetivos aparentemente imposibles. Esto funciona para los Fs, quienes podrían encontrar tales objetivos inspiradores. Pero los T rechazarán tales objetivos por ser *ilógicos*. Ésta es una razón de que es raro el grupo de desarrolladores que responda positivamente a un objetivo de planificación imposible.

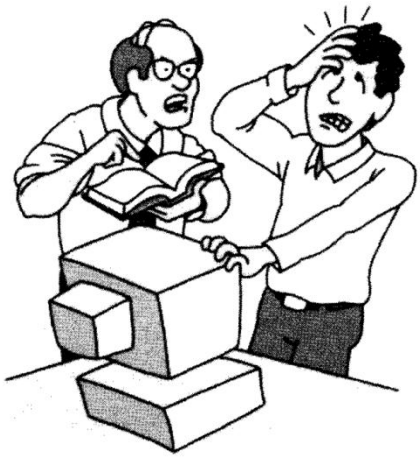
Para cada tipo de personalidad funcionan diferentes formas de motivación. La motivación puede proporcionar ideas generales, pero tendrá más éxito si se identifica cual es la motivación más efectiva para cada individuo. Preguntar a las personas lo que piensan es de gran ayuda.

Los cinco factores que más influyen en la motivación del desarrollador según [McConnell, 1997] son: realización, posibilidad de superación, el trabajo en sí, vida personal y oportunidad de supervisión técnica.

Figura 2.3 La motivación de un desarrollador de software (McConnell, 1997)

II.2.3 La desmotivación.

El líder debe tomar en cuenta la personalidad lógica de los desarrolladores de software, adelantar artificialmente la fecha de entrega para "cubrirse" o una presión irrazonable (*figura 2.4*) hace que los desarrolladores pierdan el respeto a sus directivos.



«¡Si el libro dice que la planificación más corta posible es de 6 meses, sólo tendría que trabajar duro y más tiempo para terminar en 4 meses!»

Figura 2.4: La presión irrazonable en la planificación puede hacer que los desarrolladores pierdan el respeto a sus directivos (McConnell, 1997)

El exceso de estrés en los participantes de un proyecto los hace generar más errores. Es preferible hacer un compromiso con las personas para que trabajen correctamente durante su jornada de ocho horas con la recompensa de irse temprano que presionar para que hagan muchas horas extras y/o trabajo durante vacaciones y fines de semana. La falta de esparcimiento y de un descanso adecuado genera personas malhumoradas, fatigadas e indispuestas con una fuerte tendencia a producir un trabajo de mala calidad, además tienden a perder mucho tiempo hablando por teléfono, platicando con sus compañeros o jugando en Internet.

Figura 2. 5: Circulo vicioso de presión en la planificación (McConnell, 1997).

Otros destructores de la moral:

Manipulación de los directivos: dar información falsa acerca de fechas límite.

Presión excesiva de la planificación: presentar a los desarrolladores una fecha límite imposible.

Falta de apreciación de los esfuerzos del desarrollador: menospreciar el trabajo de las personas es una manera segura de desmotivarla.

Participación de directivos sin preparación técnica: Si un directivo reconoce su falta de preparación técnica y reconoce las capacidades del equipo que dirige y sin embargo tiene habilidades para la coordinación y la motivación, entonces hay probabilidades de éxito, pero, si intenta meterse en decisiones técnicas que no comprende perderá el respeto de sus subordinados y de esta manera será muy difícil dirigirlos.

Lemas, posters, discursos y otros métodos de motivación excesivos: pueden resultar insultantes para la inteligencia del desarrollador de software, con los que funciona mejor un ligero toque.

II.3 El equipo de desarrollo de software.

Para organizar al equipo de trabajo es necesario hacer que su tamaño y la estructura concuerde con el tamaño del proyecto, las características del producto que se va a entregar y los objetivos de planificación. Según [McConnell, 1997], el responsable del proyecto de software ocupa el papel de productor de una obra de teatro.

Él es el responsable de obtener financiamiento, de coordinar, y asegurarse de que todo el mundo esta en el sitio correcto en el momento adecuado. Sin una cuidadosa selección de personal y una dirección estricta, algunos proyectos caen en la mediocridad o incluso fracasan. E1 modelo de teatro resulta particularmente apropiado para equipos software dominados por fuertes personalidades.

Todo el mundo sabe que los actores y las actrices son temperamentales, y algunos desarrolladores de software también tienen reputación de “prima donna”. Si dentro de un proyecto hay un papel suficientemente importante, y sólo puede desempeñarlo un desarrollador determinado, el director puede decidir si va a entenderse con la “prima

donna" para la buena marcha del proyecto. Pero si el resto del reparto también es potente, puede prescindir de la "prima donna" para tener un desarrollo tranquilo del proyecto. El modelo de teatro es un modelo adecuado para proyectos multimedia modernos. Mientras anteriormente los proyectos software necesitaban integrar las contribuciones de varios desarrolladores de software, ahora tienen que integrar las contribuciones de diseñadores gráficos, escritores, productores de vídeo, productores de audio, editores, ilustradores, coordinadores de contenido y varios desarrolladores de software, es decir hay que coordinar a grandes equipos.

Así como en las obras de teatro, cuando se contrata al personal se le asigna un rol específico, a algunos solo les interesan los papeles principales (diseño), otros aceptan cualquier papel mientras no sea el del villano (por ejemplo, para algunos las bases de datos), a otros no le importa hacer el papel del malo y puede que algún actor esté contratado en otra obra a otro horario y no pueda invertir mucho tiempo, así que le acomoda un papel secundario (pruebas, por ejemplo). Es muy importante que el responsable del proyecto tenga en cuenta lo anterior para no indisponer a los participantes cambiando los papeles, ya que después surgen los reproches como: "Yo dije que haría cualquier cosa menos bases de datos ¡y eso es precisamente lo que me pusieron a hacer!" ó "¿que hago yo haciendo interfaces de usuario?, ¡esto nunca funcionará correctamente si no me consideran en el diseño!".

Cuando los roles no están bien definidos y las habilidades de los participantes no son complementarias se corre el riesgo de que todos quieran imponer su voluntad.

El responsable técnico es aquella persona que tiene alguna responsabilidad de gestión dentro de un equipo de desarrolladores de software. Habitualmente, a esta persona se le asigna este papel en base a su experiencia técnica más que a su experiencia de gestión. Por lo general, al responsable técnico le resulta difícil diferenciar entre desarrollo técnico y gestión, y puede hundir el proyecto si no cumple bien su papel (por estar poco integrado con el equipo o mal relacionado con sus superiores).

Los directivos y los responsables técnicos no trabajan siempre en una estrecha relación. Hay muchos problemas (superposición de responsabilidades, motivación, relaciones con los clientes, baja calidad, poca coincidencia en los objetivos del proyecto, etc.) que pueden mejorarse cuando tienen una comunicación efectiva sobre los temas con los que están trabajando. Uno de los mayores obstáculos para un rendimiento efectivo de la figura del responsable técnico es la falta de una división clara de responsabilidades entre el responsable técnico y el director. En teoría, el responsable técnico es el responsable del trabajo técnico, y es responsable de un solo equipo. El director es responsable de la dirección de los aspectos no técnicos del equipo, y es responsable de dos o más proyectos. Desde el punto de vista del equipo, el papel del director consiste en liberar al responsable técnico gestionando ciertas tareas no técnicas. Desde el punto de vista de la organización, el papel del director es controlar el equipo para que se adapte a los objetivos de la organización. Como los detalles específicos de las relaciones entre responsable técnico y director son tan variables, resulta útil que ambos discutan sus funciones al principio del

proyecto. Esto ayuda a evitar choques de responsabilidad en los que ambas personas piensan que son responsables de lo mismo, y vacíos de responsabilidad, en los que ambas personas creen que el responsable es el otro. Como las funciones específicas pueden variar dependiendo del proyecto en cuestión, puede usarse lo sugerido por [McConnell, 1997] en la *figura 2.3* como punto de partida para definir las responsabilidades del director y del responsable técnico y aclarar la división de responsabilidades en un proyecto determinado.

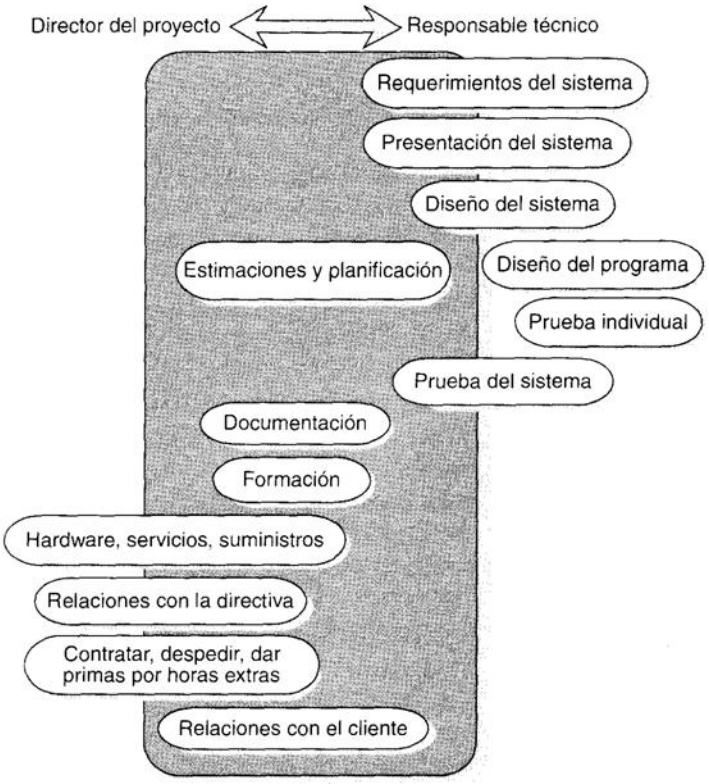


Figura 2.6: Responsabilidades del director y del responsable técnico de un proyecto (McConnell, 1997).

II.4 Aspectos sobre la coordinación y la comunicación.

II.4.1 El equipo de trabajo.

Los equipos grandes plantean problemas especiales de comunicación y coordinación, a medida que se incrementa el número de participantes en un proyecto, aumenta también el número de canales de comunicación y la necesidad de coordinación.

Un proyecto con dos personas sólo tiene una vía de comunicaciones. Un proyecto con cinco personas tiene 10 vías. Un proyecto con nueve personas tiene 45, suponiendo que todo el mundo hable con todo el mundo (ver *figura* 2.7). Cuantas más vías de comunicaciones tenga un proyecto, más tiempo se empleará en las comunicaciones, y habrá más oportunidades de que se produzcan errores en la comunicación. Un proyecto con 1.200 vías de comunicación tiene demasiados caminos para funcionar correctamente, y en la práctica, los proyectos con 50 personas no están organizados de modo que todo el mundo se comunique con todo el mundo. Los proyectos grandes requieren métodos de organización que formalicen y simplifiquen las comunicaciones. La formalización de las comunicaciones es un elemento importante para tener éxito en grandes proyectos. Todos los métodos para simplificar las comunicaciones se basan en la creación de algún tipo de jerarquía, es decir, crear grupos pequeños que funcionen como equipos, y luego asignar responsables dentro de dichos grupos para interactuar entre sí y con la directiva.

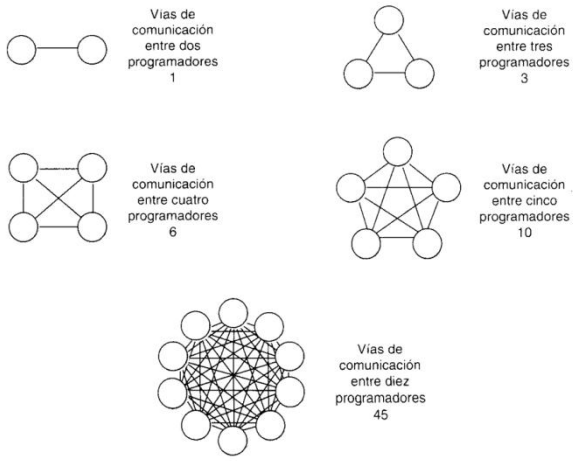


Figura 2. 7: Vías de comunicación en función del número de programadores (McConnell, 1997).

[McConnell, 1997] recomienda lo siguiente para optimizar la comunicación:

- Formar un conjunto de equipos de negocios, y luego designar un representante en cada equipo para comunicarse con los otros grupos.

• Formar un conjunto de equipos con jefe programador, y hacer responsable de la comunicación con otros grupos al programador de reserva.
• Formar equipos para desarrollar una o un grupo de funcionalidades, y hacer responsable de la comunicación con otros grupos al representante de gestión de programas.
• Debe haber una sola persona responsable de que todos los equipos conformen una buena solución global.

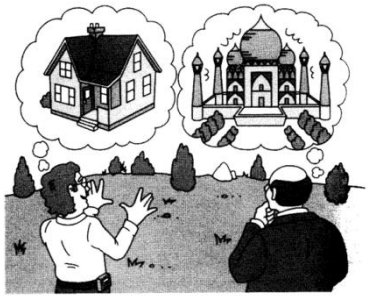


Figura 2. 8: Fallas en la comunicación (McConnell, 1997).

Figura 2.9: Responsabilidades de los líderes de proyecto (Briseño, 2003)

Las personas dentro de un proyecto necesitan saber:

- Qué tareas deben hacer.
- Cuándo las deben llevar a cabo.
- Qué productos deben entregar (documentos, código, etc.)

II.5 Errores clásicos relacionados con las personas.

A continuación se describen algunos de los errores clásicos relacionados con las personas según [McConnell, 1997]:

- **Motivación débil.**- Estudio tras estudio se ha mostrado que la motivación probablemente tiene mayor efecto sobre la productividad y la calidad que ningún otro factor [Boehm, 1981]. Es importante no tomar medidas que bajen la moral: como dar muchos ánimos al principio para pedir horas extras en la mitad del proyecto, o como irse de vacaciones mientras el equipo ha estado trabajando incluso los días festivos, o dar recompensas al final del proyecto que resulten ser de menos de lo que las personas deberían ganar por cada hora extra.
- **Personal mediocre**.- Después de la motivación, la capacidad individual de los miembros del equipo, así como sus relaciones como equipo, probablemente tienen la mayor influencia en la productividad [Boehm, 1981]. No hay que escatimar esfuerzos en buscar a personal realmente capaz aunque sea a costa de invertir más tiempo en la búsqueda.
- **Empleados problemáticos incontrolados**.- El personal problemático también representa una amenaza a la velocidad de desarrollo. El manejo incorrecto de un empleado problemático casi siempre ocasiona quejas de los demás miembros de un equipo. Cuando se tiene un empleado de este tipo y sus compañeros lo manifiestan, hay que tener cuidado, pues es muy probable que posteriormente otros tengan que re-hacer el trabajo de ese empleado.
- **Hazañas**.- Algunos directivos dan más aplausos a los alardes de "gran capacidad" que a los progresos firmes y consistentes y a los informes significativos de progreso, cuando se le da más importancia a estas actitudes que a los informes del estado exactos, que a veces son pesimistas, los directivos de estos proyectos no pueden tomar medidas correctivas al respecto, es más, ni siquiera saben que tienen que emprender acciones correctivas hasta que el daño ya está hecho.
- **Añadir más personal a un proyecto retrasado**.- Éste es quizás el más clásico de los errores clásicos. Cuando un proyecto se alarga, añadir más gente puede quitar más productividad a los miembros del equipo existente de la que añaden los nuevos miembros, ya que se puede perder más tiempo del que se pretende ahorrar, en capacitar a los nuevos miembros para incorporarlos y hacerlos realmente productivos.
- **Oficinas repletas y ruidosas**.- La mayoría de los desarrolladores consideran sus condiciones de trabajo como insatisfactorias. Los trabajadores que están en

oficinas silenciosas y privadas tienden a funcionar significativamente mejor que aquellos que ocupan cubículos en salas ruidosas y repletas. Los entornos repletos y ruidosos alargan los planes de desarrollo.

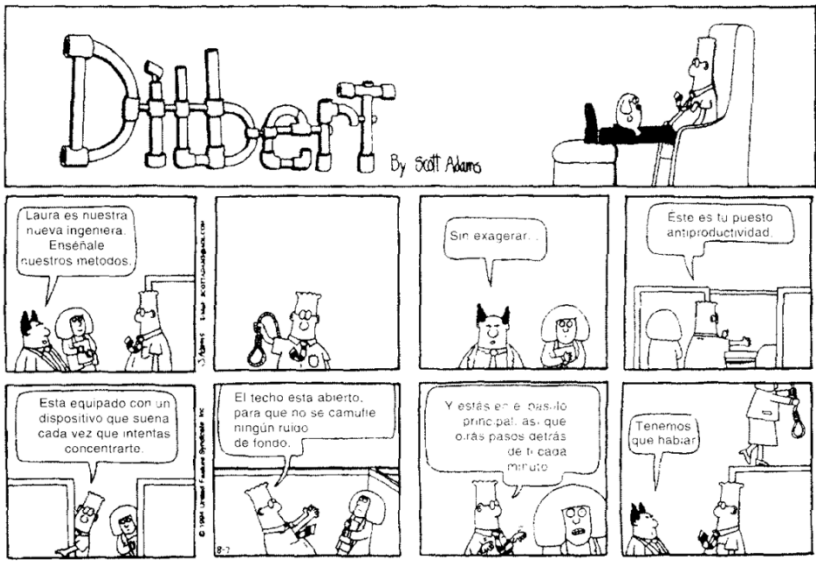


Figura 2.10: Ambiente improductivo
(McConnell, 1997)

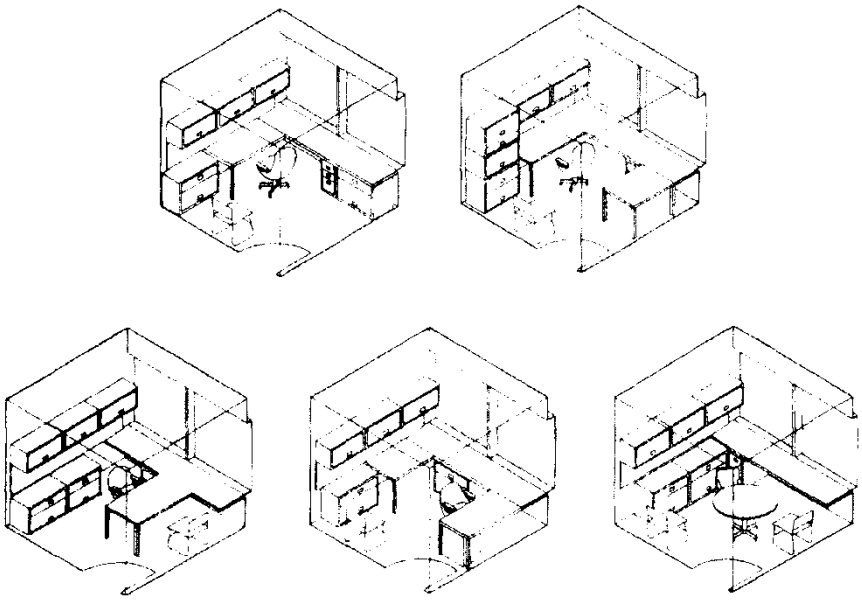


Figura 2.11: Diseños de oficinas productivas según IBM
(IBM Systems Journal, vol 17, núm. 1)
(McConnell 1997)

- **Fricciones entre los clientes y los desarrolladores**.- Las fricciones entre los clientes y los desarrolladores pueden presentarse de distintas formas. A los clientes puede parecerles que los desarrolladores no cooperan cuando no quieren comprometerse con el plan de desarrollo que desean los clientes o cuando fallan al entregar lo prometido. A los desarrolladores puede parecerles que los clientes no son razonables porque insisten en planes irreales o cambios en los requerimientos después de que éstos ya han sido fijados. Pueden ser simplemente conflictos de personalidad entre dos grupos. E1 principal efecto de esta fricción es la mala comunicación, y los efectos secundarios de la mala comunicación incluyen el pobre entendimiento de los requerimientos, pobre diseño de la interfaz de usuario y, en el peor caso, el rechazo del cliente a aceptar el producto acabado. En algunos casos, las fricciones entre clientes y desarrolladores de software llegan a ser tan severas que ambas partes consideran la cancelación del proyecto. Para remediar estas fricciones se consume tiempo y distraen tanto a desarrolladores como a clientes del trabajo real en el proyecto.
- **Expectativas poco realistas**.- Una de las causas más comunes de fricciones entre los desarrolladores y sus clientes o los directivos son las expectativas poco realistas. La incapacidad para corregir expectativas irreales es una de las principales fuentes de problemas. En algunos casos, los directivos o los desarrolladores de un proyecto se buscan problemas al pedir fondos basándose en estimaciones de planificación demasiado optimistas, en las que prometen entregar demasiadas funcionalidades en muy poco tiempo. El no entregar un proyecto en el tiempo estimado hace que parezca un proyecto malo.
- **Falta de un promotor efectivo del proyecto**.- Para lograr buenos resultados se necesita un promotor del proyecto de alto nivel capaz de llevar a cabo una planificación realista, el control de cambios y la introducción de nuevos métodos de desarrollo. Sin un promotor ejecutivo efectivo, el resto del personal de alto nivel de la empresa puede forzar a que se acepten fechas de entrega irreales o hacer cambios que debiliten el proyecto.
- **Falta de participación de los implicados.**-. La cooperación estrecha entre los participantes de un proyecto sólo se produce cuando todos se involucran, esto incluye a los promotores, ejecutivos, responsables del equipo, miembros del equipo, personal de ventas, usuarios finales y clientes.
- **Falta de participación del usuario**.- Cuando no se involucra al usuario desde el principio se corre el riesgo de no comprender bien los requerimientos del proyecto, así que será muy probable que se consuma tiempo en modificar requerimientos que finalmente lo retrasarán.
- **Política antes que desarrollo.**- Es importante evitar grupos que se dediquen a cultivar solamente las relaciones con los directivos, o grupos que mantienen cerradas las fronteras a los que no son miembros del equipo. Tampoco es bueno tener personas que hagan un poco de todo, ya que el que mucho abarca poco aprieta. Lo mejor es que en un ambiente de cooperación mutua, existan personas

dedicadas a explorar y reunir información y otros que coordinen a los equipos de trabajo y establezcan las relaciones con los directivos.

- **Ilusiones.-** Un ejemplo típico de ilusión es el caso en el que ninguno de los miembros del proyecto cree realmente que éste pueda completarse de acuerdo con el plan que tienen, pero piensan que quizás si trabajan duro y nada va mal, y tienen un poco de suerte, serán capaces de concluir con éxito. Otro caso común es el del equipo que no hace mucho trabajo para la coordinación de las interfaces entre las distintas partes del producto, pero tienen una buena comunicación para otras cosas, y como consideran que las interfaces son relativamente simples, probablemente solo necesitarán un día o dos para eliminar los errores. Otro ejemplo de ilusión es contar con un desarrollador de poco talento y/o poca experiencia y creer que puede compensar con mucho trabajo lo que le falta y que probablemente acabará a tiempo.

  Las ilusiones no son sólo optimismo. Realmente consisten en cerrar los ojos y esperar que todo funcione cuando no se tienen las bases razonables para pensar que será así. Las ilusiones al comienzo del proyecto llevan a grandes explosiones al final. Impiden llevar a cabo una planificación coherente y pueden ser la raíz de más problemas en el software que todas las otras causas combinadas.

Capítulo III El problema

Definición de Proyecto: Según la guía del *Project Management Institute* [PMI, 2008] un proyecto es un esfuerzo temporal para crear un producto, servicio o resultado único. La naturaleza temporal de los proyectos indica un principio y un final definidos. El final se alcanza cuando se logran los objetivos del proyecto o cuando se termina el proyecto porque sus objetivos no se cumplirán o no pueden ser cumplidos, o cuando ya no existe la necesidad que dio origen al proyecto. Temporal no necesariamente significa de corta duración. Todo proyecto crea un producto, servicio o resultado único.

III.1 Origen de un proyecto

La competitividad creciente, el constante cambio tecnológico y la globalización han obligado a las empresas a reaccionar con rapidez ante los cambios del entorno y es así como alguno o varios de los factores de la *figura 3.1* desencadenan la necesidad de llevar a cabo la gran mayoría de los proyectos:

Figura 3.1: El origen del proyecto (Subsecretaría de la Función Pública, Senn, 1992).

Una necesidad del mercado.- Modificar u optimizar los procesos existentes.
Una necesidad de negocios.- Fusiones entre empresas, adquisiciones.
El requerimiento de un cliente o proveedor.- Cambiar algo que ya existe o agregar nuevas funcionalidades.
Un avance tecnológico.- Nuevas plataformas tecnológicas.

Un requerimiento legal.- Disposiciones gubernamentales o ambientales más estrictas.

Una vez que surge la necesidad del proyecto, es necesario analizar el ámbito del software que se utilizará y posteriormente definir el problema a resolver descomponiéndolo adecuadamente.

III.2 Ámbito del software.

"La primera actividad de gestión de un proyecto es la determinación del ámbito del software" [Pressman, 2006]. Es necesario determinar si el nuevo producto se desarrollará dentro de un contexto más grande y que restricciones tendrá, también hay que identificar los datos de entrada y los que el usuario requiere visualizar y por último hay que identificar las funciones que realiza el software para transformar datos de entrada en salida.

Existen varios criterios para clasificar los proyectos de software, por ejemplo, McConnel los clasifica en 3 diferentes: 1) Software de *sistemas*: sistema operativo, controladores de dispositivos, compiladores y bibliotecas de código, también software empotrado (embebido), firmware, sistemas en tiempo real y software científico. 2) Software de *gestión* sistemas internos que se utilizan en una única empresa (nómina, contabilidad, control de almacén). 3 Software "Prêt-à-porter" paquetes de software que se venden comercialmente. Mientras que [Wiegers, 2006] clasifica los proyectos de software en: aplicaciones interactivas para usuarios, incluyendo aplicaciones web y kioscos de ventas, aplicaciones computacionalmente intensivas, almacenamiento de datos, procesamiento por lotes y productos hardware con software embebido.

El software que da soporte a una empresa puede agruparse según [Senn, 1992] en tres categorías principales que se describen a continuación:

1.- *Sistema para el procesamiento de transacciones*: Sustituye los procedimientos manuales por otros basados en computadora. Trata con procesos de rutina bien estructurados. Incluye aplicaciones para el mantenimiento de registros.

Una transacción es cualquier suceso o actividad que afecta a toda la organización. Las transacciones más comunes son: facturación, entrega de mercancía, pago a empleados y depósito de cheques. Los tipos de transacciones cambian para cada empresa en particular, sin embargo el procesamiento de transacciones incluye siempre las siguientes actividades:

- Cálculos.
- Clasificación.
- Ordenamiento.
- Almacenamiento y recuperación.
- Generación de resúmenes.

2.- *Sistema de información administrativa*: Proporciona 1a información que será empleada en los procesos de decisión administrativos. Ayuda cuando se presentan situaciones de decisión bien estructuradas y es posible anticipar los requerimientos de información más comunes.

Si bien los sistemas de transacciones están orientados hacia operaciones, los sistemas de información administrativa ayudan a los directivos a tomar decisiones y resolver problemas. Se debe identificar la información necesaria para formular decisiones y desarrollar sistemas de información que produzcan reportes de forma periódica en un formato diseñado específicamente para ello.

3.- *Sistema para el soporte de decisiones*: proporciona información a los directivos que deben tomar decisiones sobre situaciones particulares. Apoyan la toma de decisiones en circunstancias que no están bien estructuradas.

Una decisión se considera no estructurada si no existen procedimientos claros para tomarla y tampoco es posible identificar con anticipación todos los factores que deben considerarse en la decisión. Un factor clave en el uso de este tipo de sistemas es determinar la información necesaria, que en estos caso es difícil encontrar, pues conforme se adquiere la información puede ocurrir que el gerente se dé cuenta de que necesita más información, por lo tanto este tipo de sistemas debe tener una flexibilidad mayor que la de los otros. Los sistemas para el soporte de decisiones ayudan pero no reemplazan al criterio del directivo.

III.2.1 Ventajas del desarrollo de proyectos de software

El desarrollo de proyectos de software debe traer consigo por lo menos una de las siguientes ventajas:

- *Mejora de la comunicación*: el objetivo principal es acelerar el flujo de información y mensajes entre localidades remotas así como dentro de oficinas. La falta de comunicación es una fuente común de dificultades que afectan tanto a clientes como a empleados. Sin embargo, los sistemas de información bien desarrollados amplían la comunicación y facilitan la integración de funciones individuales.
- *Reducción o monitoreo de los costos*: uso de la capacidad de cómputo para procesar datos con un costo menor del que es posible con otros métodos al mismo tiempo que se mantiene la exactitud de los cálculos. También puede determinarse la evolución de costos de mano de obra, bienes e instalaciones en relación con lo esperado.
- *Aumento en la capacidad*: mayor velocidad de procesamiento, incremento en el volumen de los datos manejados por el sistema y recuperación más rápida de la información.

- *Control*: mayor exactitud y mejora en la consistencia de los datos. Salvaguardar datos importantes de tal forma que solo el personal autorizado tenga acceso a ellos.
- *Ventaja competitiva*: atraer clientes modificando los servicios proporcionados de tal forma que ellos no opten por cambiar de proveedor. Mejorar los acuerdos con los proveedores. Introducción de nuevos productos que utilizan sistemas de software.

III.3 Descomposición del problema para solicitar la aprobación del proyecto que lo resolverá.

Para abordar correctamente el problema que un nuevo sistema va a resolver, es importante descomponerlo y estudiarlo desde sus diferentes perspectivas. A continuación se mencionan los principales puntos que se deben incluir al hacer la solicitud de aprobación de un proyecto de software.

- Definición del problema.
- Detalles del problema.
- Importancia del problema.
- Solución propuesta.

Una vez que se ha definido el problema con todo y sus detalles, se ha determinado que su importancia amerita solución y se ha propuesto el desarrollo de un proyecto de software como solución a dicho problema, es necesario someter a evaluación la solicitud del proyecto. Cuando se evalúa la solicitud de un proyecto se procura satisfacer los siguientes objetivos:

- Determinación de lo que se requiere: ¿por qué se requiere? ¿existe alguna razón diferente a la identificada por el solicitante?
- Determinar del tamaño del proyecto: ¿se trata de un nuevo desarrollo o de la modificación de un sistema que ya existe?
- Evaluar los costos y beneficios de diversas opciones.
- Determinar la factibilidad técnica y operacional de las diferentes alternativas.
- Reportar hallazgos a la administración y formular recomendaciones que esbocen la aceptación o rechazo de la propuesta.

En la *figura 3.2* se muestra el formato de una *carta del proyecto* usado por la Subsecretaría de la Función Pública, en este formato se presenta la propuesta del proyecto ante un comité que se encargará de determinar si es viable o no llevarlo a cabo.

CARTA DEL PROYECTO
NOMBRE DEL PROYECTO:
RESPONSABLE DEL PROYECTO::
OBJETIVO:
BENEFICIOS:
DESCRIPCIÓN:
PRIORIDAD:
JUSTIFICACIÓN:
PRODUCTOS:
APROBACIÓN:

Figura 3.2: Carta del proyecto (Subsecretaría de la Función Pública).

III.4 Errores clásicos relacionados con la tecnología

Según [McConnell, 1997], los errores típicos relacionados con el uso correcto o incorrecto de la tecnología moderna son los que se mencionan a continuación:

- **Síndrome de la panacea**.- Sucede cuando se confía demasiado en las supuestas ventajas de tecnologías con las que el equipo de desarrollo no ha trabajado antes (generadores de informes, diseño orientado a objetos, nuevos ambientes de desarrollo). Es probable que la nueva tecnología sí sea muy buena, sin embargo no sea la más apropiada para el ambiente de desarrollo del proyecto en particular.
- **Sobreestimación de las ventajas del empleo de nuevas herramientas o métodos.-** Las organizaciones mejoran raramente su productividad a grandes saltos, sin importar cuántas herramientas nuevas o métodos empleen ni tampoco lo buenos que éstos sean. Los beneficios de las nuevas técnicas tienen su contraparte, pues es necesario aprender a utilizarlas para aprovecharlas al máximo y hacer esto lleva su tiempo. Las nuevas técnicas implican nuevos riesgos, que sólo se descubren usándolas. Normalmente las mejoras son lentas y continuas en lugar de obtenerse grandes saltos. Un caso especial de sobreestimaciones de las mejoras se produce cuando se reutiliza código de proyectos anteriores. Este tipo de reutilización puede ser una técnica muy efectiva, pero el tiempo que se gana no es tan grande como se espera.
- **Cambiar de herramientas a mitad del proyecto**.- Es un viejo recurso que rara vez funciona. Cuando estamos a la mitad de un proyecto, la curva de aprendizaje, rehacer el trabajo y los inevitables errores cometidos con una herramienta totalmente nueva, normalmente anulan cualquier beneficio posible.
- **Falta de control automático del código fuente**.- Una falla en la utilización del control automático del código fuente expone a los proyectos a riesgos innecesarios. Cuando los desarrolladores trabajan sobre una misma parte de un programa sin control automático, deben coordinar su trabajo manualmente. Teóricamente deberían coordinarse para poner la última versión de cada archivo

del código fuente en el directorio maestro y verificarlo con los demás antes de copiarla. Sin embargo, muy a menudo sucede que alguno sobre-escribe el trabajo de otro, esto ocasiona que se desarrolle código nuevo con interfaces desfasadas, y después se tiene que rediseñar el código al descubrir que se ha utilizado una versión equivocada. También sucede que los usuarios avisan de errores que no podemos reproducir porque no hay forma de volver a crear los elementos que han utilizado.

Capítulo IV El proceso de la administración de un proyecto

IV.1 Ciclo de vida del proceso de administración del proyecto

Así como el desarrollo de software se divide en varias fases, también la administración de un proyecto se divide en fases las cuales en su conjunto son conocidas como el ciclo de vida de la administración del proyecto. Como se expuso en el capítulo I.4, las fases del proceso de la administración de un proyecto son: el *inicio*, la *planeación*, la *ejecución* y el *cierre.* Además se lleva un *control* durante el desarrollo de todo el proyecto. En la *figura 4.1* se puede apreciar una gráfica del nivel de actividad del administrador del proyecto durante el ciclo de vida de la administración del proyecto

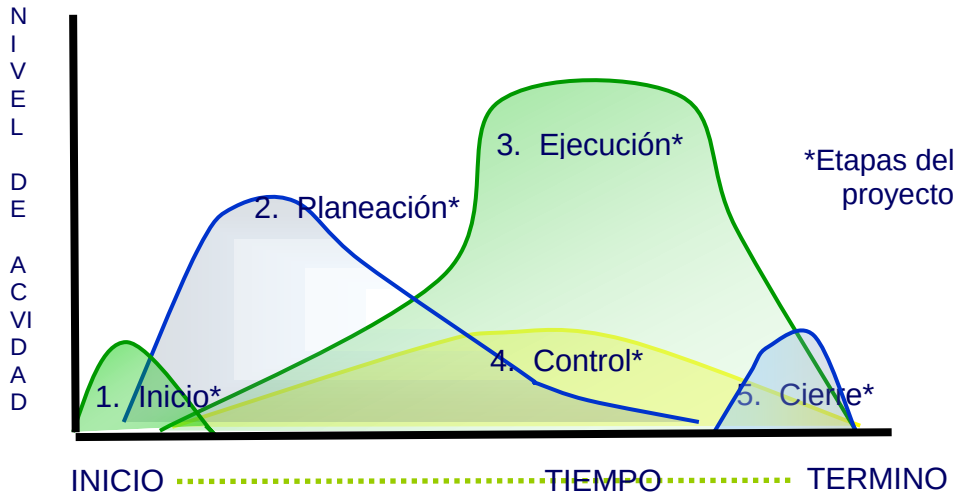


Figura 4.1: ciclo de vida del proceso de administración de un proyecto (Subsecretaría de la función pública).

IV.1.1 El inicio

El inicio del proyecto incluye poner en claro qué se debe lograr con éste, plantear el *alcance* y la selección de los miembros iniciales del equipo, la conformación de un equipo de personas adecuado influirá mucho en el desarrollo posterior. El *alcance* de un proyecto define su tamaño del proyecto, el tiempo y los recursos que se requieren.

Los recursos necesarios se deben aprobar durante esta fase y se debe determinar la fecha en la que estarán disponibles para poder empezar. Otras actividades importantes en esta

etapa son: la definición de los factores de riesgo y la definición del o los modelos de desarrollo de software que se utilizarán.

Al terminar la fase de inicio, además del contrato, se debe contar con el Acta de Constitución del proyecto y el documento del Enunciado del Alcance. Al final del inicio del proyecto se debe formalizar la aceptación del alcance del proyecto, si después surgen cambios éstos se aceptan o rechazan mediante un proceso de control. Cuando hay una mala definición del alcance, lo más probable es que el costo del proyecto sea muy alto.

En la *tabla 4.1* se especifica lo que debe incluir cada uno de los documentos según la Guía de los fundamentos para la Dirección de Proyectos del *Project Management Institute*.

Acta de Constitución	Enunciado del Alcance
Propósito o justificación del proyecto	Descripción del alcance del producto (elaborado gradualmente)
Objetivos medibles del proyecto y criterios de éxito relacionados	Entregables del proyecto
Requisitos de alto nivel	Criterios del usuario para la aceptación del producto
Descripción del proyecto a alto nivel, características del producto	Límites del proyecto
Resumen del cronograma de hitos	Restricciones del proyecto
Resumen del proyecto	Supuestos del proyecto
Requisitos para la aprobación del proyecto (qué constituye el éxito, quién lo decide, quién firma la aprobación del proyecto)	
Director del proyecto asignado, nivel de responsabilidad y de autoridad	
Nombre(s) y responsabilidad(es) de la(s) persona(s) que autoriza(n) el acta de constitución del proyecto	

Tabla 4.1: Elementos del Acta de Constitución y del Enunciado del Alcance [PMI, 2008].

IV.1.2 La planeación

La planeación del proyecto consiste en: perfeccionar el alcance, hacer un listado de tareas y actividades para lograr las metas, definir una secuencia de actividades, desarrollar un calendario, elaborar un presupuesto, establecer fechas de entregas parciales (hitos) y hacer un plan de emergencia, para que cada participante sepa que hacer en el caso de un cambio en las prioridades o en las tareas.

El plan del proyecto debe aprobarse según las normas de calidad establecidas antes de proceder con la siguiente etapa. Dependiendo del tamaño del proyecto, el proceso de planeación puede resultar bastante complejo, esto se expone con más detalle en el capítulo V. En la *figura 4.2* podemos observar gráficamente las etapas de la administración del proyecto y todas las actividades de la etapa de planeación.

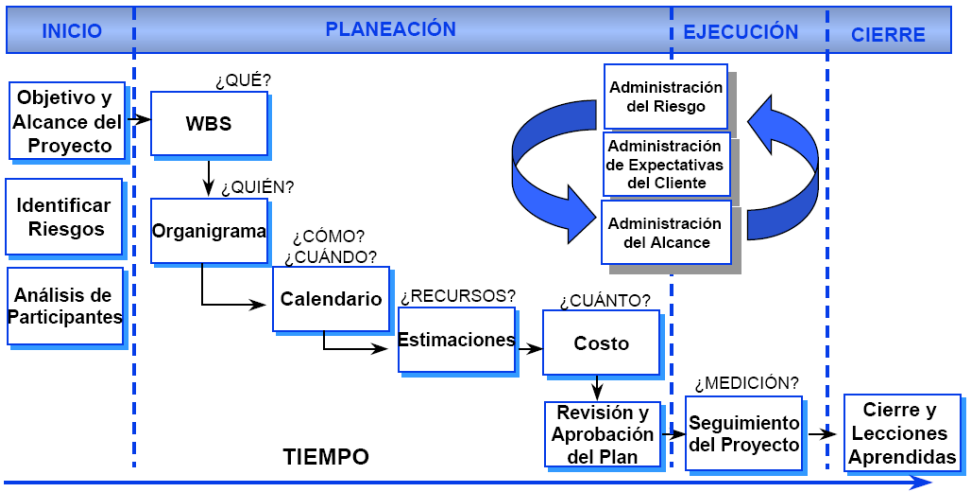


Figura 4.2: Etapas de la administración de proyectos (Briseño, 2003).

IV.1.3 La ejecución

La ejecución del proyecto incluye: dirigir al equipo, comunicarse con el cliente, proveedores y demás externos, resolver conflictos y asegurar los recursos necesarios (dinero, personal, equipo y tiempo). Como se puede apreciar en la *figura 4.1* el proceso de ejecución en la administración de un proyecto es el que requiere de mayor actividad por parte del administrador.

El monitoreo del proyecto es una de las actividades más importantes de la *ejecución* de la administración. Un buen sistema de monitoreo debe contener los siguientes cuatro puntos fundamentales:

1.- Un *estándar* para lograr un desempeño aceptable.
2.- Un método para *medir* el desempeño actual.
3.- Un método para *comparar* el desempeño actual contra el estándar.
4.- Un método de *retroalimentación*.

En la práctica, como se puede apreciar en la *figura 4.3*, las etapas de planeación y ejecución se retroalimentan una a la otra constantemente.

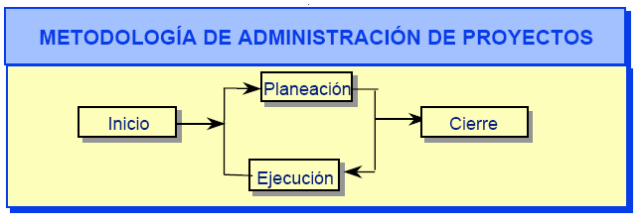


Figura 4.3: Metodología de la administración de proyectos (Briseño, 2003)

IV.1.4 El cierre

El cierre del proyecto contempla una serie de actividades como:

- reconocimiento de logros y resultados,
- cierre de las actividades y dispersión del equipo,
- aprendizaje de la experiencia del proyecto,
- revisión del proceso y resultados, redacción del informe final,
- auditorías.

El *cierre* de un proyecto se puede clasificar según se ilustra en la *figura4.4*:

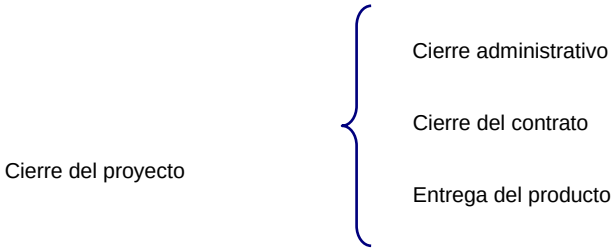


Figura 4.4: El cierre de un proyecto. (Subsecretaría de la función pública)

El *cierre administrativo* comprende las siguientes actividades:

- La elaboración y entrega de un **Reporte final** el cual debe contener:
  - El presupuesto.
  - El programa.
  - Las evidencias.
  - Las lecciones aprendidas
  - El Reporte de control de cambios.
- La entrega de los **Archivos** del proyecto.
- Un **Plan de transición** entre el proyecto que finaliza y el siguiente.

El *cierre del contrato* abarca la recopilación de lo siguiente:

- Archivos de contrato.
- Manuales, planos.
- Bitácoras.
- Comunicados.
- Lecciones aprendidas.

Y finalmente la *entrega del producto* consiste en entregar el sistema, servicio o resultado final al cliente.

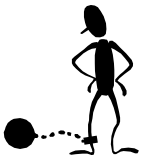
IV.1.5 El control

El control se lleva a cabo a lo largo de toda la administración del proyecto, las actividades que corresponden al *control* del proyecto son las siguientes:

- Vigilar las desviaciones del plan.
- Acciones correctivas.
- Recibir y evaluar cambios solicitados.
- Cambiar calendarios.
- Adaptar recursos.
- Regresar a la etapa de planeación para hacer ajustes.
- Control de costos.
- Control de calidad.
- Informes de resultados.
- Comunicación con los interesados.

Es muy importante que el administrador sepa que:

Durante la ejecución del proyecto se debe *controlar el trabajo* y no a los trabajadores.



IV.1.6 La norma IEEE-1058 para la Planificación de Proyectos de Software

La dirección de proyectos es una tarea integradora que requiere que cada proceso del producto y del proyecto esté alineado y conectado de manera adecuada con los demás, a fin de facilitar la coordinación. Normalmente, las acciones tomadas durante un proceso afectan a ese proceso y a otros relacionados. Una dirección de proyectos exitosa incluye dirigir activamente estas interacciones a fin de cumplir con los requisitos del patrocinador, el cliente y los demás interesados. En determinadas circunstancias, será necesario repetir varias veces un proceso o conjunto de procesos para alcanzar el resultado requerido [PMI, 2008].

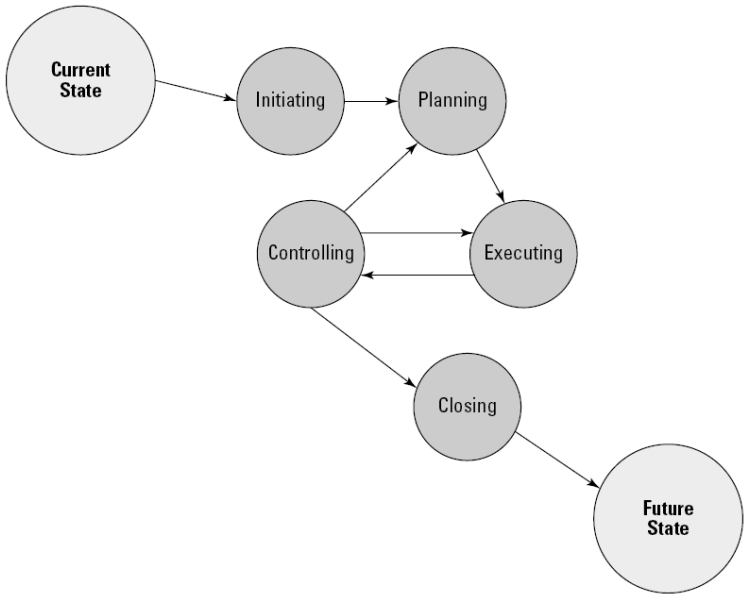


Figura 4.5: Diagrama de estados de la Administración de un Proyecto [Luckey & Phillips, 2006].

Como se observa en la *figura 4.5*, la administración de un proyecto puede modelarse con un diagrama de estados, el *control* se hace en paralelo a la *ejecución* para verificar que el proyecto se esta llevando a cabo conforme al plan.

Si el proyecto es un éxito es mérito de todo el equipo, pero si el proyecto falla solo es culpa del administrador del proyecto [Luckey & Phillips, 2006].

La norma IEEE-1058 [IEEE-1058,1993] para la Planificación de Proyectos de Software es un estándar internacional que especificar cual debe ser el contenido y el formato para presentar el Plan de Administración de un Proyecto de software. A continuación se muestra el formato general de la IEEE-1058.

- Página del título
- Hoja de revisión
- Prefacio
- Tabla de contenidos
- Lista de figuras
- Lista de tablas
- 1. Introducción.
  1.1. Visión general del proyecto.
  1.2. Entregables del proyecto.
  1.3. Evolución del PGPS.
  1.4. Materiales de referencia.

1.5. Definiciones y acrónimos.
- 2. Organización del proyecto.
  2.1. Modelo de procesos.
  2.2. Estructura organizativa.
  2.3. Fronteras e interfaces organizativas.
  2.4. Responsabilidades
3. Procesos de gestión.
  3.1. Objetivos y Prioridades de la Gestión.
  3.2. Supuestos, dependencias y restricciones.
  3.3. Gestión de Riesgos.
  3.4. Mecanismos de supervisión y control.
  3.5. Plan de personal.
- 4. Procesos técnicos.
  4.1. Métodos, herramientas y técnicas.
  4.2. Documentación del software.
  4.3. Funciones de soporte al proyecto.
- 5. Paquetes de trabajo, calendario y presupuesto.
  5.1. Paquetes de trabajo.
  5.2. Dependencias.
  5.3. Requerimientos de recursos.
  5.4. Presupuesto y distribución de recursos.
  5.5. Calendario.
- Componentes adicionales
- Índice
- Apéndices

Para mayor información consultar [IEEE-1058,1993].
Taxonomía de los modelos y metodologías de desarrollo de software más utilizados

IV.1.7 Relación entre *proceso* y *modelo* del software

Un **proceso de desarrollo de software** es el conjunto estructurado de las actividades requeridas para elaborar un sistema de software, estas actividades son: especificación de requerimientos, diseño, codificación, validación (pruebas) y mantenimiento. Al proceso de desarrollo de software también se le conoce como ciclo de vida del software porque describe la vida de un producto de software; primero nace con la especificación de los requerimientos, luego se ejecuta la implantación, que consiste en su diseño, codificación y pruebas, posteriormente el producto se entrega, y sigue viviendo durante su utilización y mantenimiento. Cuando el ciclo de vida es evolutivo, estas actividades se llevan a cabo cíclicamente. La vida del sistema de software termina cuando éste se deja de utilizar. Se dice que el producto evoluciona cuando se le hacen modificaciones que generan nuevas versiones.

Por otra parte, un **modelo de desarrollo de software** es una representación abstracta de este proceso [Sommerville, 2006]. Un modelo de desarrollo de SW determina el orden en el que se llevan ejecutan las actividades del proceso de desarrollo de SW, es decir, es el procedimiento que se sigue durante el proceso.

Hay una gran variedad de paradigmas o modelos de desarrollo de software. Los libros más conocidos de ingeniería de software [Braude, 2003; McConnell, 1997; Pfleger, 2002; Pressman, 2002; Sommerville, 2006; Weitzenfeld, 2004] explican solo los que consideran más importantes y el problema en ellos es que las opiniones acerca de cual es la lista de modelos que debe considerarse son diversas. Sommerville [2006], clasifica todos los procesos de desarrollo de software en tres modelos o paradigmas generales que no son descripciones definitivas de los procesos del software sino más bien, son abstracciones de los modelos que se pueden utilizar para desarrollar software, y son los siguientes:

a) **Modelos en cascada**. Las actividades fundamentales del proceso de desarrollo de software se llevan a cabo como fases separadas y consecutivas. Estas actividades son: especificación (análisis y definición de requerimientos), implantación (diseño, codificación, validación) y mantenimiento. Los modelos en cascada constan básicamente de 5 fases que son:

Análisis y definición de requerimientos. Se trabaja con los clientes y los usuarios finales del sistema para determinar el dominio de aplicación y los servicios que debe proporcionar el sistema así como sus restricciones. Con esta información se produce el documento de "Especificación de Requerimientos del Sistema".

Diseño del sistema y del software. Durante el proceso de diseño del sistema se distinguen cuales son los requerimientos de software y cuales los de hardware. Después se establece una arquitectura completa del sistema. Durante el diseño del software se identifican los subsistemas en los cuales estará compuesto el sistema y se describe cómo funciona cada uno y las relaciones entre éstos.

Implementación y validación de unidades. Consiste en codificar y probar los diferentes subsistemas por separado. La prueba de unidades implica verificar que cada una cumpla su especificación (proveniente del diseño).

Integración y validación del sistema. Una vez que se probó que funciona individualmente cada una de las unidades, éstas se integran para formar un sistema completo que debe cumplir con todos los requerimientos del software. Cuando las pruebas del sistema completo son exitosas, éste se entrega al cliente.

Funcionamiento y mantenimiento. El sistema se instala y se pone en funcionamiento práctico. El mantenimiento implica corregir errores no descubiertos en las etapas anteriores del ciclo de vida y mejorar la implantación de las unidades del sistema para darle mayor robustez (y no nuevas funcionalidades).

b) **Modelos de desarrollo evolutivo**. Los modelos evolutivos son iterativos. Se caracterizan por la forma en que permiten a los ingenieros de software desarrollar versiones cada vez más completas del sistema. Los modelos evolutivos iteran sobre las actividades de

especificación, desarrollo y validación. Un sistema inicial se desarrolla a partir de los requerimientos prioritarios o los que están mejor definidos. Esta primera versión se refina en una nueva iteración en base a las peticiones del cliente para producir un sistema que satisfaga sus necesidades. Sommerville [2006] define dos tipos de desarrollo evolutivo:

b.1) *Desarrollo exploratorio*. Se le presenta al cliente la implementación de la parte de los requerimientos que se entendió bien para recibir sus comentarios y, en base a éstos, refinar el sistema hasta que se logra desarrollar el sistema adecuado.

b.2) *Prototipos desechables.* Para descubrir o terminar de comprender los requerimientos del cliente se construye un prototipo con funcionalidad simulada y, si éste no es lo que el cliente espera, se construye otro prototipo (posiblemente desde cero) en base a una definición mejorada de los requerimientos para el sistema. El diseño del prototipo del sistema va evolucionando según se vayan entendiendo los requerimientos aunque la funcionalidad siga siendo simulada. Cuando se aclaran los requerimientos se completa la funcionalidad según el último prototipo.

c) **Modelos de componentes reutilizables**. Se basa en la existencia de un número significativo de componentes reutilizables. El reuso de los componentes tiene como finalidad usar de nuevo ideas, arquitecturas, diseños o código de una aplicación para construir otras. El proceso de desarrollo del sistema se enfoca en integrar estos componentes en el sistema en lugar de desarrollarlos desde cero.

Según Sommerville [2006], en la mayoría de los proyectos existe algo de reutilización de software. Por lo general, esto sucede informalmente cuando las personas que trabajan en el proyecto conocen diseños de código similares al requerido. Los buscan, los modifican según lo creen necesario y los incorporan en el sistema. Las etapas de especificación de requerimientos y de validación son comparables con los otros procesos, sin embargo, las etapas intermedias en el proceso orientado a la reutilización son diferentes. Estas etapas son:

Análisis de componentes. Consiste en encontrar componentes que sirvan para implementar la Especificación de Requerimientos. En general los componentes que se utilizan solo proporcionan parte de la funcionalidad requerida por lo que se necesita modificarlos.

Modificación de requerimientos. Con la información que se tiene de los componentes ya identificados, se analizan los requerimientos. Si es posible, se modifican los requerimientos para que concuerden con los componentes disponibles. Si las modificaciones no son posibles entonces se lleva a cabo nuevamente el análisis de componentes para buscar soluciones alternativas.

Diseño del sistema con reutilización. Se diseña o se reutiliza un marco de trabajo para el nuevo sistema teniendo en cuenta los componentes que se reutilizan y los componentes que serán completamente nuevos.

Desarrollo e integración. El software que no se tiene disponible y que no se puede adquirir externamente se desarrolla integrando los componentes reutilizables disponibles. En este

modelo, la integración de los sistemas es parte del desarrollo más que una actividad separada.

Los tres paradigmas o modelos de procesos genéricos: *cascada, evolutivo* y *componentes reutilizables*, se utilizan ampliamente en la práctica actual de la ingeniería del software. No se excluyen mutuamente y a menudo se utilizan juntos, especialmente para el desarrollo de sistemas grandes [Sommerville, 2006]. El problema es que en la literatura no se hace una clasificación explícita que ubique cada modelo o metodología dentro de alguna de estas clases abstractas. En la sección IV.1.10 hacemos esta clasificación para los modelos más representativos.

IV.1.8 El Proceso Unificado Racional

IBM Rational propone el desarrollo de software basado en las mejores prácticas recopiladas en innumerables proyectos exitosos. Es una metodología que llama al proceso de desarrollo de software: Rational Unified Process (RUP) llamado en español el Proceso Unificado Racional. El RUP es un modelo de proceso híbrido ya que reúne elementos de los modelos de procesos genéricos: cascada, evolutivo y reutilización [Sommerville, 2006], además propone buenas prácticas para la especificación y el diseño. El proceso unificado se describe desde tres perspectivas:

1.- *Una perspectiva dinámica*.- Muestra las fases del modelo sobre el tiempo.
2.- *Una perspectiva estática*.- Muestra las actividades que tienen lugar durante el proceso de desarrollo, se denominan *flujos de trabajo.*
3.- *Una perspectiva práctica*.- Sugiere buenas prácticas a utilizar durante el proceso.

En cuanto a la *perspectiva práctica*, se recomiendan seis buenas prácticas aconsejables en el desarrollo de sistemas: *Desarrollar el software de forma iterativa* (entregando primero los requerimientos más importantes), *Gestionar los requerimientos* (analizando el impacto de los cambios en el sistema antes de aceptarlos y documentando los cambios aceptados), *Utilizar arquitecturas basadas en componentes* (en la mayor medida posible), *modelar el software visiblemente* (con modelos gráficos como UML), *verificar la calidad del software, controlar los cambios del software* y *gestionar los cambios del software* usando herramientas de gestión de configuraciones.

El Proceso Unificado no es un proceso apropiado para todos los tipos de desarrollo, sin embargo representa una nueva generación de procesos genéricos [Sommerville, 2006]. Las innovaciones más importantes son la separación de fases y los flujos de trabajo, y el reconocimiento de que la utilización del software en un entorno de usuario es parte del proceso. Las fases son dinámicas y tienen objetivos. Los flujos de trabajo son estáticos y son actividades técnicas que no están asociadas con fases únicas sino que pueden utilizarse durante el desarrollo para alcanzar los objetivos de cada fase.

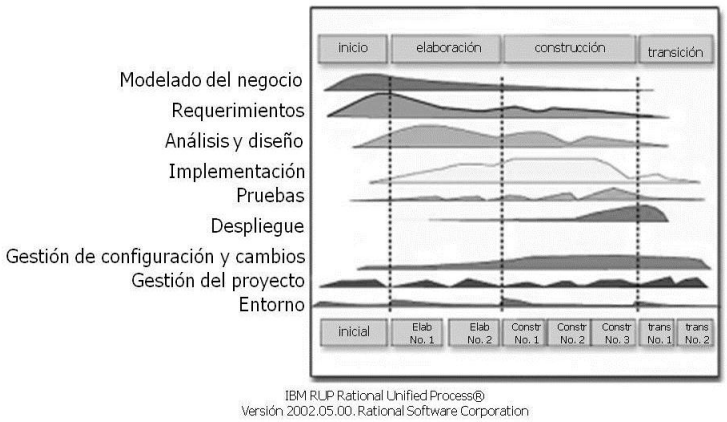


Figura 4.6 Combinación de las fases con los flujos de trabajo en el Proceso Unificado.

En la *figura 4.6* se muestra una visión global de cómo se combinan las fases dinámicas del proceso unificado, con los flujos de trabajo estáticos. Además se pueden observar los incrementos en cada fase.

Los requerimientos se trabajan desde la fase de inicio, y muy especialmente durante la fase de elaboración, pues el objetivo es tener claro por lo menos un 80% del sistema que se requiere construir [Wiegers, 2006].

Las características del Proceso Unificado según [Ambler, 2005] son:

- Visto a lo largo de todo el proyecto, es *serial* en el tiempo: comienza con la etapa de inicio, luego la etapa de elaboración, después la etapa de construcción y al final la etapa de transición.
- Visto en cada etapa es *iterativo*: la etapa puede estar compuesta de varias entregas.
- Hay entregas parciales del producto, las funcionalidades se van incluyendo de manera incremental.
- Se apoya en buenas prácticas probadas en innumerables proyectos exitosos para una gran variedad de dominios.

Al final de cada una de las etapas del Proceso Unificado se debe entregar un producto importante (hito):

- Al final del *inicio*: se entregan los objetivos y definición del alcance del proyecto.
- Al final de la *elaboración*: se entrega la arquitectura del sistema.
- En cada iteración de la *construcción*: se entrega un producto con la función anterior más el incremento correspondiente a la nueva iteración, de tal forma que al final de la *construcción* se obtiene la versión inicial del sistema con capacidad operacional, es decir, con toda la funcionalidad requerida.
- Al final de la *transición*: se entrega el producto completamente funcional.

Aunque el RUP fue concebido inicialmente para sistemas Orientados a Objetos, en algunos casos vale la pena aplicarlo a situaciones no Orientadas a Objetos y obtener de todas formas algunas de las ventajas del RUP como son:

- Hacer frente a los riesgos de cambios en los requerimientos,
- Disminuir el riesgo financiero al hacer entregas parciales de software funcional que puede probarse y ser evaluado por el cliente.
- Se puede adaptar para administrar el proceso con los niveles de flexibilidad y rigor necesarios para cada situación en particular.

"El Proceso Unificado (PU) define la actividad de la administración del proyecto como el arte de balancear los objetivos significativos, administrar el riesgo y superar las limitaciones para entregar con éxito un producto que satisfaga tanto a los clientes (aquellos que solicitan que se desarrolle el software) como a los usuarios del mismo" [Charbonneau, 2004]. La administración de Proyectos del PU define:

- Un marco de referencia para la administración de proyectos de software.
- Guías prácticas a seguir para la planeación, el manejo del personal, la ejecución de la administración y el monitoreo del proyecto.
- Un marco de referencia para la administración basada en riesgos.

La Administración del Proyectos del PU hace énfasis en los aspectos más importantes de los procesos iterativos, que son:

- El manejo de los riesgos.
- La planeación de un proyecto iterativo: definición de las iteraciones para cada una de las etapas.
- Monitoreo del progreso y métricas para un sistema iterativo.

IV.1.9 Otros tipos de desarrollo de software: Las Metodologías *Ágiles*

Hasta ahora hemos hablado de las llamadas *metodologías tradicionales* las cuales se basan en la disciplina y el orden, sin embargo, "por más que en los últimos años han evolucionado diversas metodologías para asegurar un mejor control del proceso, los clientes quedan frecuentemente insatisfechos con el resultado" [Aguilar, 2002]. Si bien, la mala administración es la principal causa detectada en los fracasos en los proyectos de software, algunos también atribuyen el fracaso a la metodología empleada para su desarrollo. Las *metodologías ágiles* surgen como otra alternativa de desarrollo para contrarrestar estos fracasos. Las prácticas que recomiendan son algunas veces opuestas a lo recomendado por las *metodologías tradicionales*. La metodologías ágiles se basan en un "desarrollo iterativo e incremental en muy breves ciclos y un diseño inicial simple" [Araújo, 2007]. Según estudios recientes, las metodologías ágiles tienen una gran aceptación en la industria del

software [West & Grant, 2010], sin embargo, según sus fundadores, éstas solo son aplicables cuando se dan las siguientes condiciones [Fowler, 2000]:

- Proyectos pequeños y equipos con menos de 100 personas.
- Requerimientos cambiantes.
- Equipo de desarrollo competente.
- Cliente dispuesto a participar con el equipo.

En febrero de 2001 se emitió el "Manifiesto para el desarrollo ágil del software" [Manifesto, 2001] en el cual se estipulan las características que debe tener un desarrollo ágil. Las metodologías ágiles valoran más a los individuos y las interacciones entre éstos que a los procesos y a las herramientas. Se fomenta más la comunicación cara a cara que la documentación, de tal manera que el tiempo se emplea en producir software que funciona en lugar de usarlo para producir documentación. Se le da más énfasis a la colaboración con el cliente en los aspectos claves del desarrollo que a la negociación del contrato y se concentran en la respuesta a los cambios en lugar de elaborar un plan y seguirlo ya que, según esta filosofía, es imposible anticipar todos los requerimientos desde el inicio del desarrollo [Pfleeger & Atlee, 2006].
[West & Grant, 2010] realizaron un estudio sobre los métodos y metodologías más utilizados en la industria del software durante el 2008 y el 2009. Según los resultados, en el 2008 algunas metodologías ágiles ocuparon el primer lugar, éstas fueron: SCRUM [Pazderski, 2010], programación extrema, llamada en inglés "XP: eXtreme Programming" [Aguilar, 2002], Desarrollo Dirigido por Pruebas, llamado en inglés "TDD: Test-Driven development" [Araújo, 2007], Delgado o Menudo, conocido como "Lean" [Poppendieck & Poppendieck, 2003], Desarrollo Dirigido por Características, llamado en inglés "FDD: Feature Driven Development" [Anderson, 2004] y Modelado Ágil [Ambler, 2008]. En segundo lugar se utilizaron modelos iterativos y en tercer lugar el modelo en cascada. Otras metodologías ágiles y el Proceso Unificado tuvieron una participación muy pequeña. En el 2009, los resultados fueron muy similares, el 35% de 1,298 encuestados utilizaron metodologías ágiles (destacaron las mismas que en el 2008), el 21% utilizó métodos iterativos y el Proceso Unificado y el 13.4% utilizó el cascada. Es interesante mencionar que el 30.6% de los encuestados no utilizaron ninguna metodología formal.

IV.1.10 Taxonomía de los modelos de desarrollo de software

A continuación, en la *figura 4.7* proponemos la clasificación de los modelos y metodologías concretos más citados en la literatura en cinco clases abstractas: Cascada, Evolutivos, Minimización de Desarrollos, Híbridos y Ágiles. Los dividimos en *Modelos Tradicionales* (también llamados *pesados*), que son los que promueven la disciplina por medio de la planificación y la comunicación escrita, y los *Metodologías Ágiles*, que dan prioridad a la interacción entre los individuos y a la comunicación con el cliente.

Modelo Abstracto		Modelos Concretos
Tradicionales o Pesados	En Cascada	Pura Con fases solapadas Con subproyectos Con reducción de riesgos
	Evolutivos	Espiral Entrega por etapas o incremental Entrega evolutiva o iterativo Diseño por planificación Cascada en V
	Minimización de Desarrollos	Componentes Reutilizables Diseño por herramientas
	Híbridos	Proceso Unificado Racional Otros
Metodologías Ágiles		Programación extrema SCRUM Desarrollo dirigido por pruebas Desarrollo dirigido por Características Agile, Lean, Crystal, …, etc.

Figura 4.7: Clasificación de modelos concretos en clases abstractas.

La descripción detallada de los ciclos de vida que involucran cascada pueden consultarse en [Braude, 2003; McConnell, 1997; Pfleger, 2002; Pressman, 2002; Sommerville, 2006; Weitzenfeld, 2004], el modelo en cascada es adecuado cuando los requerimientos del sistema son claros y estables, por lo general, el software que se produce contiene pocos defectos. Sin embargo tienen el inconveniente de ser muy poco flexibles, ya que un cambio en los requerimientos sale muy caro y retrasa todo el proceso. Este tipo de modelos permite una documentación completa y los documentos que se generan al término de cada etapa hacen visible la estructura del sistema, lo cual es muy útil en sistemas grandes donde intervienen muchos desarrolladores.

La descripción detallada de los ciclos de vida evolutivos se pueden consultar en [Braude, 2003; McConnell, 1997; Pfleger, 2002; Pressman, 2002; Sommerville, 2006; Weitzenfeld, 2004]. Los modelos evolutivos tienen como objetivo principal reducir el riesgo y la incertidumbre en el desarrollo. Los requerimientos y el diseño requieren de la investigación repetida para asegurar que el desarrollador, el usuario y el cliente tengan una comprensión unificada tanto de lo que se necesita como de lo que se propone como solución. Con el desarrollo evolutivo se pretende que: con incrementos pequeños se facilite la administración del proyecto, también que se facilite la comprensión y las pruebas de estos incrementos, y que sea factible satisfacer el cambio de los requerimientos a través del tiempo. Sin embargo, algunas veces no es sencillo determinar cuales serán los requerimientos que se implantarán en cada iteración.

Los modelos de componentes reutilizables se pueden consultar en [Braude, 2003; McConnell, 1997; Pressman, 2002; Sommerville, 2006; Weitzenfeld, 2004]. El modelo de

componentes reutilizables tiene la ventaja obvia de reducir los costos y los riesgos, sin embargo, la modificación de los requerimientos puede provocar que el sistema no cumpla con las necesidades reales de los usuarios.

Existe también el llamado "modelo de métodos formales" [Braude, 2006; Pressman, 2002], el cual pretende que se especifique, desarrolle y verifique un sistema aplicando una notación rigurosa y matemática. El objetivo es descubrir y corregir ambigüedades, inconsistencias o requerimientos incompletos. La definición de este modelo es escasa e incipiente, su descripción se limita básicamente a los métodos formales para la Especificación de Requerimientos [Braude, 2006, Pfleeger, 2006]. Como su desarrollo requiere de personal altamente especializado, es un modelo bastante caro y consume mucho tiempo, por lo que no goza de una amplia aceptación [Jackson, 1998, Norris & Rigby, 1994]. Esta categoría no se incluye dentro de la taxonomía que aquí presentamos debido a que lo que está documentado en libros son los modelos formales de Especificación de Requerimientos y no toda la secuencia *análisis de requerimientos/diseño/ implementación/pruebas.*

En la sección IV.3.2 se exponen las ventajas y desventajas de cada uno de estos modelos.

IV.2 Elección del ciclo de vida del proyecto

El proceso concreto de desarrollo de software (ciclo de vida) depende de cada proyecto, ya que cada uno tiene necesidades diferentes. Independientemente del proceso que se elija, hay que tomar en cuenta que descuidar la *calidad del proyecto en sus fases iniciales* produce un desperdicio de tiempo al corregir los errores al final, justo cuando es más caro. Como se observa en la *figura 4.8*, corregir los errores cuando están en un estado inicial es más sencillo.

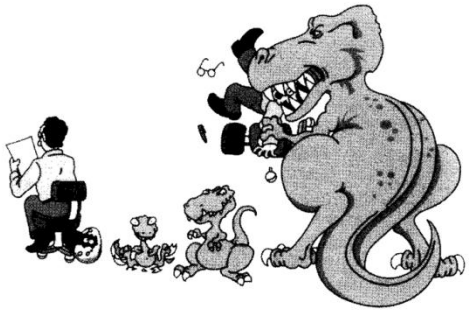


Figura 4.8: Mientras más tiempo permanezca oculto un defecto se necesitará más tiempo para corregirlo. (McConnell, 1997)

Crear un buen diseño antes de comenzar a codificar mejora la robustez del producto, si hay cambios o modificaciones en la concepción del producto durante el proceso de desarrollo será mucho más conveniente atacar el problema desde el diseño que desde el código.

Otro punto que no se debe descuidar es el *control de riesgos* ya que éste permite identificar los riesgos que podrían retrasarán el proyecto y tomar medidas preventivas. En el capítulo VI se abordará el tema del control de riesgos.

IV.2.1 Características de algunos ciclos de vida

La elección del ciclo de vida que se va a utilizar para desarrollar un proyecto de software es una de las decisiones importantes que debe tomar el administrador. A continuación se mencionan las ventajas y desventajas que tienen los ciclos de vida más populares según [McConnell, 1997].

IV.2.1.1 Modelos en cascada

Modelo en cascada puro.

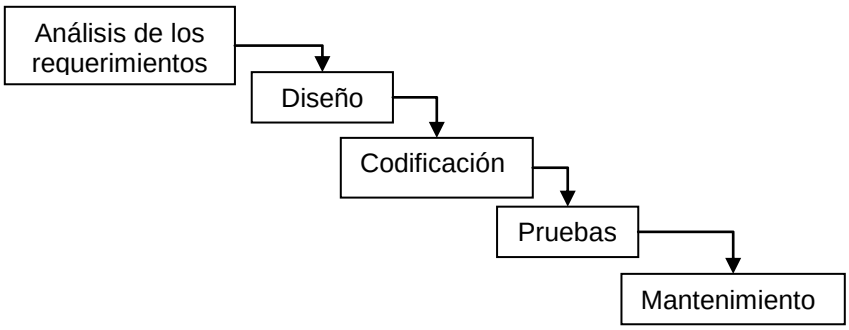


Figura 4.9: Modelo en cascada.

Ventajas:

- ☺ Genera un sistema fiable (pocos defectos cuando comienza a funcionar).
- ☺ Se produce una documentación completa del sistema.
- ☺ Funciona muy bien cuando se tiene una definición estable del producto y se trabaja con técnicas conocidas (nueva versión de un producto existente, migración de un producto a otra plataforma).
- ☺ Funciona con proyectos grandes, ya que se enfrenta a la complejidad de forma ordenada.

Desventajas:

- ☹ No ofrece a los clientes signos visibles de progreso
- ☹ No tiene buena aceptación de cambios en los requerimientos.
- ☹ No contempla gestión de riesgos.
- ☹ No funciona cuando los requerimientos no son claros desde el principio.

Modelo en cascada con fases solapadas.

Figura 4. 10: Modelo en cascada con fases solapadas (McConnell 1997).

Ventajas:

- ☺ Genera un sistema fiable (pocos defectos cuando comienza a funcionar).
- ☺ Se pueden descubrir ideas importantes al avanzar en los ciclos de desarrollo.

Desventajas:

- ☹ No contempla gestión de riesgos.
- ☹ No ofrece a los clientes signos visibles de progreso.
- ☹ Los hitos son ambiguos por lo que es difícil trazar el proceso correctamente.
- ☹ Las actividades en paralelo pueden provocar mala comunicación, suposiciones incorrectas e ineficacia en proyectos grandes.

Cascada con subproyectos

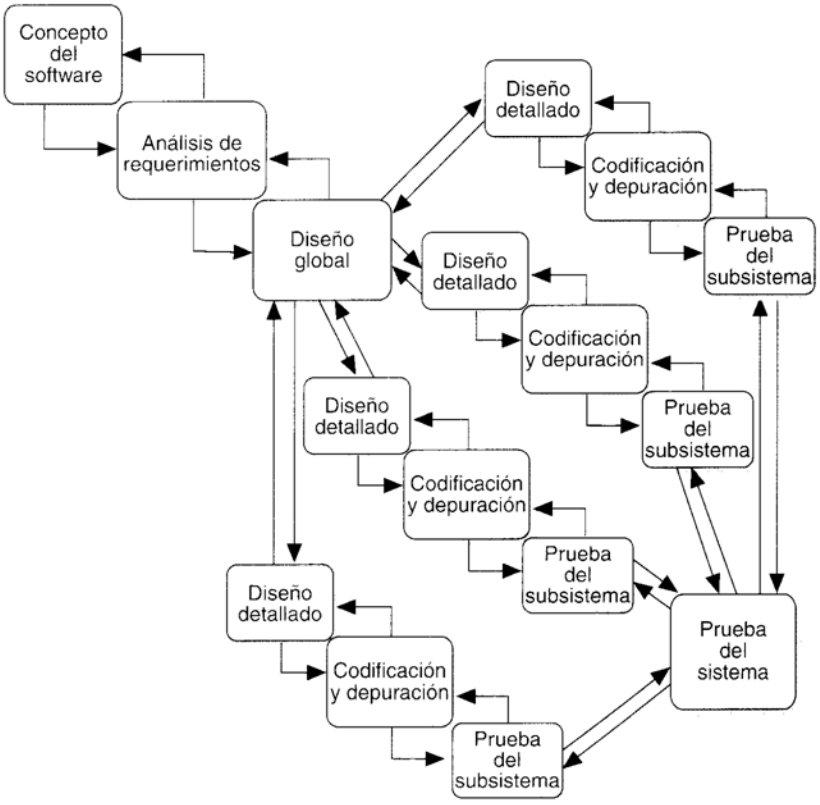


Figura 4.11 : Modelo en cascada con suproyectos (McConnell 1997).

Ventajas:

- ☺ Genera un sistema fiable (pocos defectos cuando comienza a funcionar).
- ☺ Cada subsistema lógicamente independiente es un subproyecto que avanza a su propio ritmo.
- ☺ Se produce una buena documentación completa del sistema.

Desventajas:

- ☹ Pueden existir interdependencias no previstas entre los subproyectos.
- ☹ No tiene buena aceptación de cambios en los requerimientos.
- ☹ No contempla gestión de riesgos.
- ☹ Hay problemas cuando los requerimientos no son claros desde el principio.

Cascada con reducción de riesgos

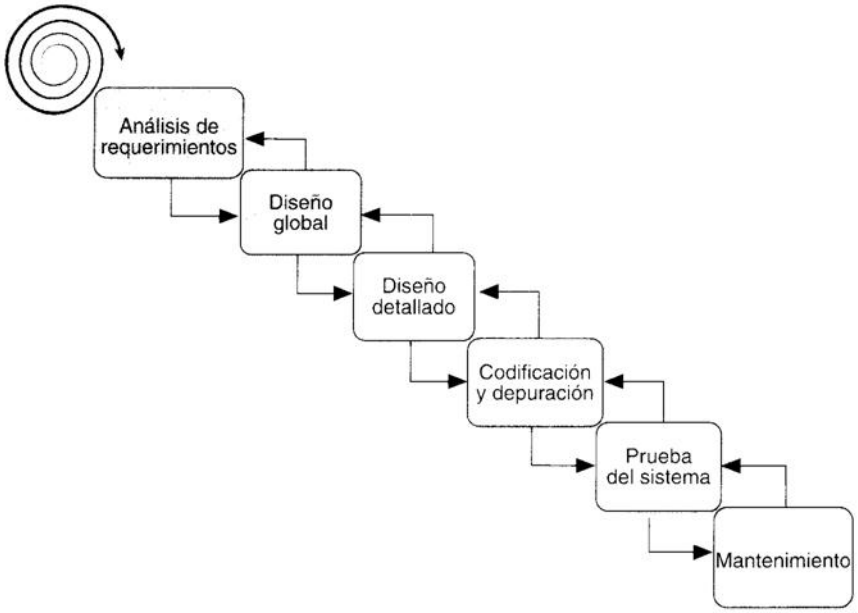


Figura 4.12: Modelo en cascada con reducción de riesgos (McConnell 1997).

Ventajas:

- ☺ Genera un sistema fiable (pocos defectos cuando comienza a funcionar).
- ☺ Se controla el riesgo de los requerimientos con una espiral que identifica los requerimientos.
- ☺ Se produce una buena documentación completa del sistema.

Desventajas:

- ☹ Las modificaciones en los requerimientos afectan bastante una vez que ya se esta trabajando en etapas avanzadas.
- ☹ No ofrece a los clientes signos visibles de progreso.

IV.2.1.2 Modelos evolutivos

Modelo en espiral.

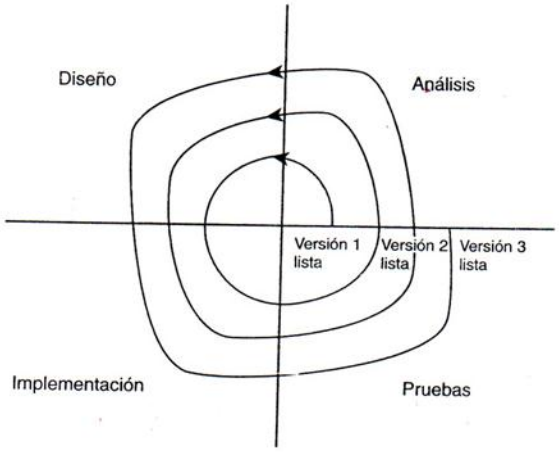


Figura 4.13: Modelo en espiral (Weitzenfeld, 2004).

Ventajas:

- ☺ Genera un sistema fiable (pocos defectos cuando comienza a funcionar).
- ☺ Tiene buena aceptación de cambios en los requerimientos.
- ☺ Se puede empezar a trabajar aún con poca identificación de los requerimientos.
- ☺ Contempla la gestión de riesgos.
- ☺ Ofrece a los clientes y directivos signos visibles de progreso.

Desventajas:

- ☹ Requiere de una alta capacitación por parte de los directivos y desarrolladores del proyecto para utilizar el modelo con éxito.
- ☹ El proceso no es visible (no es practico hacer documentación para cada etapa).
- ☹ Los cambios continuos tienden a corromper la estructura del software. Incorporar cambios en él se convierte cada vez más en una tarea difícil y costosa sobre todo cuando se trata de sistemas grandes.

Prototipos evolutivos.

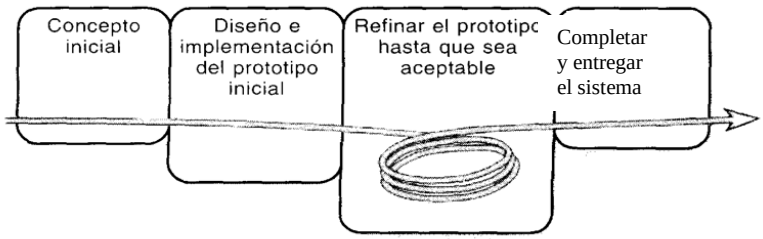


Figura 4.14: Modelo de prototipos evolutivos (McConnell 1997).

Ventajas:

- ☺ Se puede empezar a trabajar aún con poca identificación de los requerimientos.
- ☺ Tiene buena aceptación de cambios en los requerimientos.
- ☺ Ofrece a los clientes signos visibles de progreso.

Desventajas:

- ☹ Requiere de una alta capacitación por parte de los directivos y desarrolladores del proyecto para utilizar el modelo con éxito.
- ☹ Se contempla la gestión de riesgos, sin embargo ésta no es óptima.
- ☹ El sistema no es altamente fiable.
- ☹ No se puede conocer al inicio del proyecto cuanto tiempo se empleará en crear un producto aceptable.
- ☹ Contempla medianamente la gestión de riesgos.

Entrega por etapas (implementación incremental).

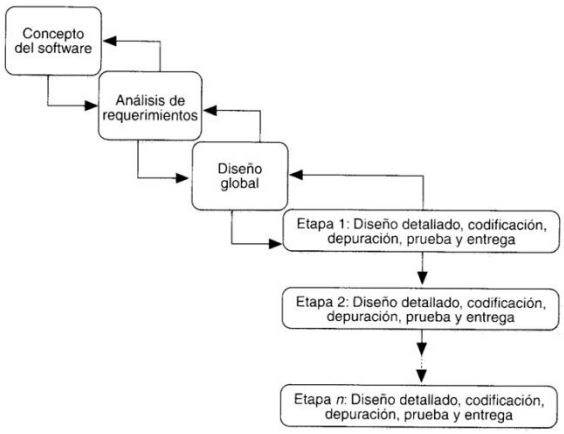


Figura 4.15: Modelo de entrega por etapas (McConell, 1997).

Ventajas:

- ☺ Se ofrece al cliente funcionalidad útil conforme esté lista.
- ☺ Genera un sistema fiable (pocos defectos cuando comienza a funcionar).
- ☺ Se ofrece a los directivos signos visibles de progreso.

Desventajas:

- ☹ No funciona sin una planificación adecuada de las etapas significativas para el cliente y sin una distribución adecuada del trabajo entre los desarrolladores.
- ☹ Se corre el riesgo de retrasar la implementación de un componente para una etapa posterior y luego darse cuenta que alguna de las primeras etapas no funciona sin este componente.
- ☹ No funciona cuando los requerimientos no son claros desde el principio.
- ☹ No tiene buena aceptación de cambios en los requerimientos.
- ☹ Contempla medianamente la gestión de riesgos.

Entrega evolutiva

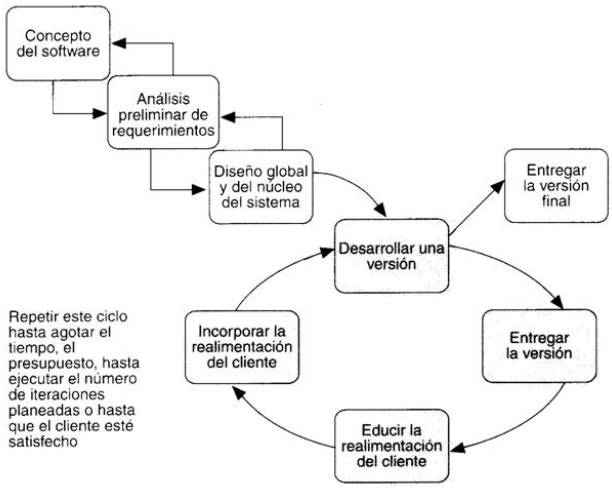


Figura 4.16: Modelo de entrega evolutiva (McConell, 1997).

Ventajas:

- ☺ Ofrece el control de la entrega por etapas y la flexibilidad de los prototipos evolutivos.
- ☺ Trabaja con poca identificación de los requerimientos.
- ☺ Se ofrece al cliente y directivos signos visibles de progreso.
- ☺ Reduce el riesgo de entregar al cliente un producto que no desea.
- ☺ Reduce los problemas de integración integrando pronto y con frecuencia.

Desventajas:

- ☹ Los cambios en los requerimientos no son manejables fácilmente.
- ☹ Contempla medianamente la gestión de riesgos.
- ☹ Si no se conoce completamente el sistema son mejores los "prototipos evolutivos".

IV.2.1.3 Modelo basado en componentes y diseño por herramientas

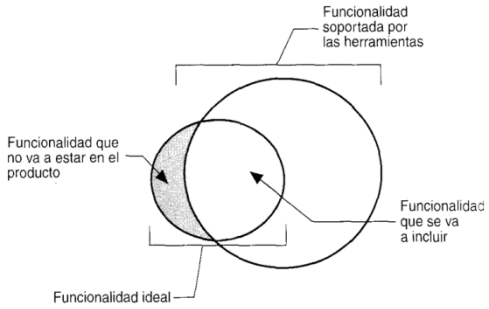


Figura 4.17: Modelo de diseño por herramientas (McConell, 1997).

Ventajas:

- ☺ Se ofrece al cliente y directivos signos visibles de progreso.
- ☺ Los cambios en los requerimientos son manejables.

Desventajas:

- ☹ Podrían no cumplirse las necesidades reales de los usuarios.
- ☹ Podría generar un sistema poco fiable.
- ☹ No contempla gestión de riesgos.

IV.2.2 Ventajas y desventajas de los diferentes modelos

"La elección de un modelo de ciclo de vida erróneo puede dar lugar a la omisión de tareas y a una secuenciación inapropiada de las mismas, lo cual va en contra de la planificación y eficiencia del proyecto. La elección de un modelo apropiado tiene el efecto contrario, asegurando que todo el esfuerzo se utiliza eficientemente" [McConnell, 1997].

Puesto que la necesidad de un proyecto se genera a partir del surgimiento de requerimientos, la naturaleza de éstos es uno de los aspectos importantes para la elección del modelo de desarrollo. Los requerimientos están estrechamente relacionados con la elección del modelo para desarrollar el proyecto.

En los *modelos tipo cascada*, los requerimientos tienen que estar bien definidos desde el inicio del proyecto y la probabilidad de que cambien debe ser mínima. Cabe mencionar que esto aplica tanto al desarrollo de sistemas nuevos como al desarrollo de modificaciones

sobre un sistema existente. Pfleeger & Atlee [Pfleeger & Atlee: 2006] recomiendan además el uso de un modelo en cascada cuando los requerimientos están fuertemente acoplados o cuando son complejos, es decir, cuando no es sencillo separar los requerimientos para desarrollarlos uno por uno ya que se corre el riesgo de que la implementación de unos no sea compatible con la de otros. Otro caso en el que el modelo en cascada es viable, es cuando muchas personas participan en el proyecto, ya sea porque el proyecto es grande (en cuyo caso ya se ha justificado el uso de un modelo tipo cascada) o porque se requiere de la colaboración de especialistas. En estos casos es mucho más sencillo el uso de modelos tipo cascada ya que resulta muy importante tener procedimientos de control estrictos y una comunicación formal y disciplinada entre los participantes. Una recomendación importante [McConnell, 1997] es que no se aumente el número de participantes cuando los tiempos de entrega son cortos ya que solamente se logra que la organización se complique debido a lo complejo de la comunicación entre las personas. Si el tiempo de entrega es muy importante entonces no se recomienda el uso de modelos tipo cascada. Solamente cuando la calidad del producto sea prioritaria sobre el tiempo de entrega es bueno adoptar uno de estos modelos [Pfleeger & Atlee, 2006].

En suma, los modelos tipo cascada son recomendables para: requerimientos bien definidos y que no cambian; requerimientos fuertemente acoplados o complejos; proyectos donde interviene una gran cantidad de personas y, cuando es más importante entregar un sistema funcionando correctamente que cumplir con una fecha de entrega preestablecida. Un estudio realizado por [Khalifa, 2000] confirma que para sistemas grandes, los desarrolladores prefieren el modelo en cascada, mientras que para sistemas pequeños los procesos evolutivos son más aceptados que el cascada.

En los *modelos evolutivos*, los requerimientos se trabajan al inicio de cada iteración para aumentarlos, corregirlos o redefinirlos. En estos modelos se complica mantener actualizada y correcta la documentación. Además, en sistemas muy grandes, cada nueva adición puede implicar que el código se corrompa debido a una mala administración de los cambios. En este tipo de procesos, la Especificación se desarrolla junto con el software, esto puede crear conflictos en las organizaciones en las que la Especificación de Requerimientos es parte del contrato [Sommerville, 2005]. Los procesos evolutivos permiten mostrar al cliente una versión parcial preliminar que permita obtener retroalimentación y evite problemas con la integración de un código muy grande. Para que este modelo sea útil se debe poder comenzar con algunos requerimientos prioritarios y dejar para los ciclos posteriores los demás, además hay que contar con la participación del usuario, quien debe dedicar tiempo a evaluar y retroalimentar las entregas parciales [Braude, 2007]. Los modelos evolutivos como el espiral o el incremental tienen grandes ventajas: los clientes pueden comenzar a utilizar un sistema que tiene los requerimientos prioritarios para ponerlo a prueba y reportar sus fallas. De esta manera aumenta la probabilidad de entregar un software que opere satisfactoriamente. Además se facilita la recolección de métricas acerca del proceso en cada iteración. Sin embargo [Sommerville, 2005] recomienda que el código no rebase las 20,000 líneas en cada incremento y a veces puede ser difícil adaptar los requerimientos del cliente al tamaño apropiado de un incremento. Braude [Braude, 2006] señala un punto importante: “con el propósito de optimizar la productividad en equipo, con frecuencia es necesario

comenzar una nueva iteración antes de que la anterior haya terminado", esto no solo dificulta la coordinación de la documentación, sino que dificulta la coordinación de los cambios en los diferentes módulos del sistema cuando un requerimiento tiene impacto en varios de estos módulos.

En los *modelos de minimización de desarrollos*, las funcionalidades de los componentes o módulos deben ser similares a las nuevas funcionalidades que se requieren, de tal forma que el esfuerzo en modificar los componentes base no sea tan alto que el proceso se entorpezca en lugar de agilizarse. Se considera una buena práctica en la ingeniería de software hacer diseños modulares ya que éstos tienen el potencial de la reutilización de algunas de sus partes [Braude, 2006], sin embargo, esto no es fácil. Cuando se adopta este modelo, normalmente se negocia con el cliente para hacer modificaciones a sus requerimientos con la finalidad de que éstos se adapten a los componentes base. Es muy importante cuidar que estas modificaciones no produzcan un sistema que no cumpla con las necesidades reales de los usuarios. La calidad de un sistema basado en reutilización de componentes dependerá mucho de la robustez de éstos y el mantenimiento del sistema estará limitado por la facilidad de acceso que se pueda tener para modificarlos.

El *Proceso Unificado Racional* (RUP) es un modelo híbrido que pretende sacar las ventajas de los modelos cascada, evolutivos y las de los de componentes reutilizables. Como se mencionó anteriormente, visto a lo largo de todo el proyecto, es decir, desde una perspectiva dinámica, el RUP es *serial* en el tiempo, tal y como el modelo en cascada. La parte evolutivo/iterativa en la que se descompone cada una de las etapas, reduce riesgos y es hasta cierto punto flexible en los cambios de requerimientos. La reutilización de componentes que se fomenta con este modelo permite reducir costos y tiempo de desarrollo. El uso del Lenguaje de Modelado Unificado: UML [Booch et al., 1999] asociado al RUP, facilita el análisis y el diseño de los componentes del sistema. Sus procedimientos de control de calidad y control de cambios contribuyen a la producción de un software satisfactorio. Sin embargo, el RUP es un modelo complicado, se requiere de una alta capacitación del administrador del proyecto para llevarlo a buen término. Además, los miembros del equipo de desarrollo también deben tener una alta capacitación en el uso de este complejo modelo, probablemente este sea el motivo por el cual, según estudios recientes [West & Grant, 2010], el RUP no es uno de los modelos más utilizados.

Las *metodologías ágiles* están pensadas para afrontar el problema de los requerimientos inciertos o cambiantes, las entregas iniciales tienen el objetivo de implementar los requerimientos esenciales del cliente. Con el uso de estas primeras versiones se comprende mejor el problema a solucionar y emergen nuevos requerimientos que son cubiertos en entregas posteriores. Además, la entrega continua de nuevas versiones permite hacer frente a los cambios de última hora. Cada entrega contiene requerimientos determinados al momento. El riesgo de éste tipo de metodologías es la carencia del documento de Especificación de Requerimientos. Por ejemplo, en la programación extrema, y en el Desarrollo Dirigido por Pruebas la documentación de los requerimientos se substituye por "casos de prueba" que el sistema debe pasar cuando se implantan ciertos requerimientos. Esta escasa documentación dificulta hacer cambios al sistema cuando ya se borraron, agregaron o modificaron requerimientos anteriores sobre los que influyen estos nuevos

cambios. Además, si la documentación de los requerimientos en los casos de prueba, es pobre, posiblemente surgirán malos entendidos en su implementación o modificación [Pfleeger & Atlee, 2006].
El modelo de desarrollo que se adopte depende de cada proyecto ya que cada uno tiene necesidades diferentes. Hay que tomar en cuenta la capacidad y experiencia del personal en el tipo de proyecto a desarrollar para elegir algún método específico. La cantidad de personal con el que se cuenta también puede llegar a ser decisivo. No es necesario limitar la elección a un solo modelo de desarrollo, pues en algunos casos es mejor combinar varios modelos [Braude, 2006; Sommerville, 2006]. Los estudios de [West & Grant, 2010] informan que un alto porcentaje de las empresas de software decide mezclar modelos tradicionales con metodologías ágiles. Por otra parte, hay que tomar en cuenta que, independientemente del modelo que se elija, descuidar la calidad del proyecto en sus fases iniciales implica un desperdicio de esfuerzo al corregir los errores al final, justo cuando es más caro [McConnell, 1997]. Esto no significa que el inicio sea *lo* importante pero sí que es muy importante y decisivo para que un proyecto tenga éxito.

IV.3 Administración eficiente de un proyecto

IV.3.1 Estrategias para disminuir la probabilidad de fracaso de un proyecto

Una vez que surge la necesidad de llevar a cabo un proyecto, el reto es administrar tareas que nunca antes se habían llevado a cabo y que tal vez no se repetirán en el futuro. [Pressman, 2002; McConnell, 1997; Toledo; Sommerville, 2006,] plantean algunas estrategias a seguir para disminuir la posibilidad del fracaso de un proyecto, éstas se resumen a continuación:

1. **No iniciar sin tener un objetivo bien definido.-** Muchos proyectos se originan con ideas sueltas o buenos deseos. Hay que evitar lo anterior a toda costa definiendo por escrito desde un principio cual es el producto o servicio que pretende crear el proyecto, los objetivos específicos, las restricciones que debe cumplir, si existen, y los beneficios que la empresa u organización obtendrá con el proyecto. Lo anterior debe estar asentado en un documento que tenga el visto bueno de todos los involucrados.

2. **Fragmentar el proyecto.-** La mayoría de los administradores de proyecto, han encontrado que la mejor estrategia para definir un proyecto es la de "divide y vencerás"**.** Una vez definidos claramente los objetivos, se fragmenta de forma gradual el proyecto a un nivel razonable de detalle. Es decir, se divide el proyecto en cuatro, cinco o seis partes grandes (comúnmente llamadas fases, etapas o partidas) y se continúa subdividiendo cada parte en grupos más pequeños hasta llegar al nivel de actividades o tareas (usualmente paquetes de trabajo de 40 a 80 horas de esfuerzo).

3. **Invertir tiempo en la planeación.-** Invertir tiempo en planeación, ahorra tiempo en la ejecución. Cuando se analiza previamente el camino que se va de seguir podemos evitar pérdidas de tiempo y aumentos en el costo. Según los principios básicos de Calidad Total, el trabajo de planeación, normalmente es mucho más barato que el de ejecución, por lo que siempre es bueno tener en mente que ningún proyecto es tan urgente como para no dedicarle tiempo suficiente a su planeación.

4. **Involucrar al equipo de trabajo, tanto en la planeación como en el control.-** Algunos líderes de proyecto cometen el error de realizar individualmente la mayor parte del trabajo administrativo. Cuando comparten con el equipo de proyecto, el trabajo de planeación y control, éste se enriquece y casi automáticamente se generan dos situaciones ventajosas: compromiso y entendimiento, de los involucrados. Por regla general, los equipos de trabajo están poco dispuesto a aceptar un programa de trabajo o presupuesto "impuestos" por un tercero.

5. **Unir al equipo de trabajo.-** En muchas empresas modernas la organización de los recursos humanos esta dada de tal forma que en un proyecto intervienen personas de diferentes divisiones o departamentos. El líder del proyecto debe fomentar la cohesión entre los miembros del equipo. Hay estrategias que ya se ha probado que son efectivas para este fin, tales como: premios o recompensas monetarias, reuniones cara a cara que permitan a los miembros de un equipo sentirse parte del mismo y espacios físicos específicamente destinados para el trabajo en equipo.

6. **Prevenir los problemas antes de que ocurran.-** Un buen análisis de los riesgos a los que se enfrenta un proyecto y su efectiva administración minimizan o en algunos casos eliminan por completo, muchos contratiempos y problemas a los que los proyectos se enfrentan. La Administración de los Riesgos en un proyecto se ha convertido en una de las herramientas más usadas hoy en día.

7. **Monitorear el avance real del proyecto.-** Comparar el desempeño real con la planeación inicial del proyecto y, en caso de detectar desviaciones, aplicar las medidas correctivas necesarias para regresar el proyecto a buen camino.

8. **Establecer un proceso bien definido para monitorear y controlar el proyecto.-** Una de las tareas más difíciles es la revisión constante del avance del mismo. Muchas veces la medición de este avance es subjetiva y no se tiene claro en que punto del camino nos encontramos y si efectivamente vamos a llegar al final en el tiempo y costo planeados. Por lo anterior, es importante establecer un proceso de monitoreo y control muy claro, y apegarse a él durante todo el desarrollo del proyecto. Hay que establecer cuales serán los factores a medir y en que puntos se aplicarán acciones correctivas.

9. **Cuidar las Estimaciones y los Cambios.-** Prestar especial atención a las estimaciones de tiempo y costo proporcionadas por los miembros del equipo y de los proveedores, hay que validarlas como regla general, ya que normalmente son

irreales. Los cambios en el alcance o las especificaciones del proyecto pueden provocar un desastre si estos no se controlan, analizan, autorizan y documentan (en este orden) adecuadamente.

10. **Atender especialmente los puntos críticos del proyecto.-** Estos puntos normalmente son los hitos (milestones) de un programa de trabajo y coinciden con la obtención de los productos entregables más representativos del proyecto, como: la especificación, el documento de diseño, el código, los ejecutables, las interfaces de usuario y las pruebas.

11. **Invertir el tiempo suficiente en el cierre del proyecto.-** Es como invertir tiempo en la planeación. La principal razón para detenerse al final de un proyecto y hacer un análisis de lo que ocurrió, es aprender del mismo para repetir en proyectos futuros aquello que funcionó y evitar lo que no. Aparte de la meta cumplida, el producto más importante que se obtiene al final de un proyecto son las "lecciones aprendidas".

12. **Utilizar una metodología estándar para todos los proyectos.-** La estandarización es el único camino a largo plazo que permite a una organización mejorar el desempeño global de los proyectos. Una empresa exitosa no es aquella que termina con éxito un proyecto, sino la que de manera constante ejecuta proyectos exitosos. La metodología a usar depende de la elección personal del administrador del proyecto, cual sea no es lo importante siempre y cuando se aplique adecuadamente.

IV.3.2 Diagrama de Gantt.

El diagrama de Gantt es una técnica simple y muy utilizada en la administración de proyectos que muestra visualmente la relación entre las distintas actividades, identifica las relaciones de precedencia y permite hacer un mejor uso de los recursos humanos, materiales y monetarios para el proyecto. Con el diagrama de Gantt se pueden identificar fácilmente los recursos, sus funciones y el estado de sus actividades. En la *figura 4.18* se muestra un ejemplo de diagrama de Gantt.

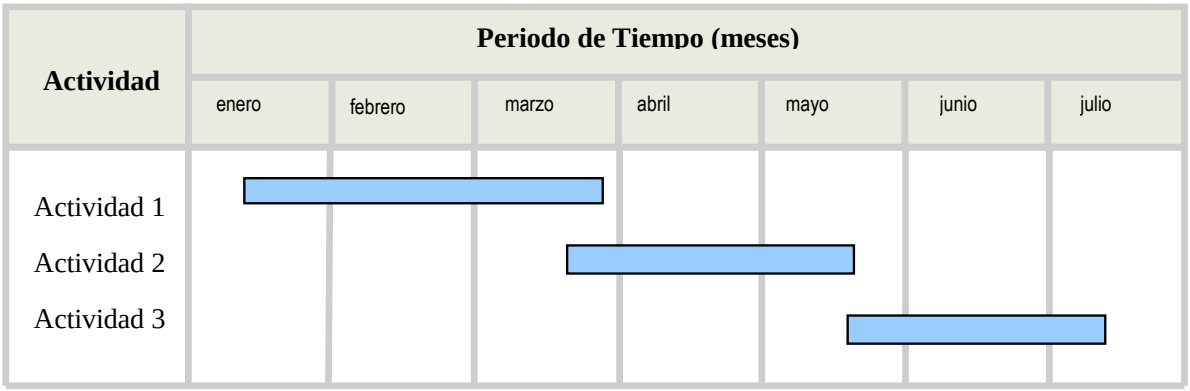


Figura 4.19: Diagrama de Gantt.

Las actividades son aquellos trabajos en que se puede dividir un proyecto. Éstas no deben ser generales, sino específicas, sin ir a demasiado detalle.

IV.4 Errores clásicos relacionados con el proceso

Los errores relacionados con el proceso vuelven lentos los proyectos porque malgastan el talento y el esfuerzo del personal. A continuación se muestran algunos de los peores errores relacionados con el proceso según [McConnell, 1997]:

- **Planificación excesivamente optimista.-** Los retos a los que se enfrenta alguien que desarrolla una aplicación en tres meses son muy diferentes de aquellos a los que se enfrenta alguien que desarrolla una aplicación que necesita un año. Fijar un plan excesivamente optimista predispone a que el proyecto falle por menospreciar alguna de las actividades críticas para el desarrollo, como el análisis de requerimientos o el diseño. También supone una excesiva presión para los desarrolladores, quienes a largo plazo se ven afectados en su moral y su productividad.
- **Gestión de riesgos insuficiente**.- Algunos errores no son lo suficientemente habituales como para considerarlos clásicos. Son los llamados *riesgos*. Como con los errores clásicos, si no ejercemos una gestión activa de los riesgos, con una sola cosa que vaya mal el proyecto se retrasará.
- **Fallas de los contratados**.- Las compañías a veces contratan la realización de partes de un proyecto cuando tienen demasiada prisa para hacer el trabajo en casa. Pero los contratados frecuentemente entregan su trabajo tarde, con una calidad inaceptable o que no coincide con la especificación [Boehm, 1989]. Si las relaciones con los contratados no se gestionan cuidadosamente, la utilización de desarrolladores externos puede entorpecer enormemente el proyecto.

- **Planificación insuficiente**.- Es indispensable la planeación correcta de un proyecto para poder llevarlo a su fin con éxito.
- **Abandono de la planificación cuando ésta falla**.- Los equipos de desarrollo hacen planes y con frecuencia los abandonan cuando se tropiezan con un problema en la planificación. El problema no está en el abandono del plan, sino más bien en no crear un plan alternativo, y caer entonces en el modo de trabajo de codificar y corregir.
- **Pérdida de tiempo en el inicio difuso.-** El *inicio difuso* es el tiempo que transcurre antes de que comience el proyecto; este tiempo normalmente se pierde en el proceso de aprobar y hacer el presupuesto. No es poco común que un proyecto desperdicie meses o años en un inicio difuso, lo que provoca un plan con prisas cuando finalmente se define bien. Es mucho más fácil, barato y menos arriesgado suprimir unas pocas semanas o meses del inicio difuso en vez de comprimir el plan de desarrollo en ese mismo tiempo.
- **Escatimar en las actividades iniciales**.- Hay proyectos que se aceleran intentando acortar las actividades "no esenciales". Es una tentación caer en el error de pensar que como el análisis de requerimientos, la arquitectura y el diseño no producen código directamente, se pueden acortar. Dice McConnell: "en un proyecto desastroso en el que participé, pedí que me enseñaran el diseño, el responsable del equipo me dijo: «No hemos tenido tiempo de hacer el diseño»". Los proyectos que normalmente escatiman en sus actividades iniciales tendrán que hacer ese trabajo en otro momento, con un costo de 10 a 100 veces superior a haberlo hecho bien inicialmente [Boehm y Papaccio, 1988]. Si no podemos encontrar cinco horas para hacer el trabajo correctamente la primera vez, ¿cómo vamos a encontrar 50 para hacerlo correctamente más tarde?
- **Diseño inadecuado**.- Un caso especial de escatimar en las actividades iniciales es el diseño inadecuado. Proyectos acelerados generan un diseño indeterminado, si no se asigna suficiente tiempo para éste, se necesitarán varios ciclos de diseño antes de poder finalizar completamente el sistema.
- **Escatimar en el control de calidad**. En los proyectos que se hacen con prisa se suele eliminar las revisiones del diseño y del código, la planificación de las pruebas y sólo se realizan pruebas superficiales. Acortar en un día las actividades de control de calidad al comienzo del proyecto probablemente supondrá mucho más esfuerzo al final del proyecto para corregir errores.
- **Control insuficiente de los directivos.-** Es importante que los directivos del proyecto detecten a tiempo los signos de posibles retrasos en el plan, estos se hace estableciendo controles al comienzo del proyecto. El directivo debe ser capaz de observar si el proyecto va por buen camino.
- **Convergencia prematura o excesivamente frecuente**.- Bastante antes de que se haya programado entregar un producto, hay un impulso para preparar el producto para la entrega, mejorar el rendimiento del producto, imprimir la documentación final, incorporar entradas en el sistema final de ayuda, pulir el

programa de instalación, eliminar las funciones que no van a estar listas a tiempo y demás. En proyectos hechos con prisa, hay una tendencia a forzar prematuramente la convergencia. Los intentos adicionales de convergencia no benefician al producto. Sólo son una pérdida de tiempo y prolongan el plan.

- **Omitir tareas necesarias en la estimación**.- Si no se guardan cuidadosamente los datos de proyectos anteriores, es muy factible olvidar las tareas menos visibles, que sin embargo son tareas que hay que añadir en la estimación.
- **Pretender ponerse al día más adelante.-** Si el producto que estamos construyendo cambia, el tiempo necesario para construirlo cambiará también. Es un error común que cuando los requerimientos cambian entre la propuesta original y el comienzo del proyecto no se haga la correspondiente reestimación del plan o de los recursos. El aumento de nuevas funcionalidades sin ajustar el plan garantiza que no se alcanzará la fecha de entrega.
- **Programación a destajo.**- Algunas organizaciones creen que la codificación rápida, libre, tal como salga, es el camino hacia el desarrollo rápido. Piensan que si los desarrolladores están lo suficientemente motivados, pueden superar cualquier obstáculo, lo cual obviamente no es verdad.

Capítulo V Planeación del proyecto

V.1 La planificación del proyecto

A continuación se enumeran las actividades de la planeación de un proyecto en general según la Guía de los fundamentos para la Dirección de Proyectos [PMI, 2008]:

1.- ***Desarrollar el Plan para la Dirección del Proyecto***: es el proceso que consiste en documentar las acciones necesarias para definir, preparar, integrar y coordinar todos los planes subsidiarios. El plan para la dirección del proyecto es la fuente primaria de información que determina la manera en que se planificará, ejecutará, supervisará y controlará, y se cerrará el proyecto.
2.- ***Recopilar Requerimientos*****:** es el proceso que consiste en definir y documentar las necesidades de los interesados a fin de cumplir con los objetivos del proyecto.
3.- ***Definir el Alcance***: se hace una descripción detallada del proyecto y del producto.
4.- ***Crear la Estructura de Desglose del Trabajo***: se subdividen los entregables y el trabajo del proyecto en componentes más pequeños y más fáciles de dirigir.
5.- ***Definir las Actividades***: se identifican las acciones específicas que deben realizarse para elaborar los entregables del proyecto.
6.- ***Secuenciar las Actividades***: se identifican y documentan las relaciones entre las actividades del proyecto.
7.- ***Estimar los Recursos de las Actividades***: se estima el tipo y las cantidades de materiales, personas, equipos o suministros requeridos para ejecuta cada actividad.
8.- ***Estimar la Duración de las Actividades***: se establece aproximadamente la cantidad de períodos de trabajo necesarios para finalizar cada actividad con los recursos estimados.
9.- ***Desarrollar el Cronograma***: se analiza el orden de las actividades, su duración, los requisitos de recursos y las restricciones para crear el cronograma del proyecto.
10.- ***Estimar Costos***: se desarrolla una aproximación de los recursos monetarios necesarios para completar las actividades del proyecto.
11.- ***Determinar el Presupuesto***: se suman los costos estimados de actividades individuales o paquetes de trabajo para establecer una línea base de costos autorizados.
12.- ***Planificar la Calidad***: se identifican los requisitos de calidad y/o normas para el proyecto y el producto, y se documenta la manera en que el proyecto demostrará el cumplimiento con los mismos.
13.- ***Desarrollar el Plan de Recursos Humanos***: se identifican y documentan los roles dentro de un proyecto, las responsabilidades, las habilidades requeridas y las relaciones de comunicación, y se crea el plan para la dirección de personal.

14.- ***Planificar las Comunicaciones***: se determinan las necesidades de información de los interesados en el proyecto y se define cómo abordar las comunicaciones.
15.- ***Planificar la Gestión de Riesgos***: se define cómo realizar las actividades de gestión de riesgos para un proyecto.
16.- ***Identificar Riesgos***: se determinan los riesgos que pueden afectar el proyecto y se documentan sus características.
17.- ***Realizar Análisis Cualitativo de Riesgos***: consiste en priorizar los riesgos para realizar otros análisis o acciones posteriores, evaluando y combinando la probabilidad de ocurrencia y el impacto de dichos riesgos.
18.- ***Realizar Análisis Cuantitativo de Riesgos***: consiste en analizar numéricamente el efecto de los riesgos identificados sobre los objetivos generales del proyecto.
19.- ***Planificar la Respuesta a los Riesgos***: se desarrollan opciones y acciones para mejorar las oportunidades y reducir las amenazas a los objetivos del proyecto.
20.- ***Planificar las Adquisiciones***: consiste en documentar las decisiones de compra para el proyecto, especificar el enfoque e identificar posibles vendedores.

La etapa de planificación de un proyecto de software puede llegar a ser bastante compleja dependiendo del tamaño del proyecto, como se observa en la *figura 5.1*. Primero debe hacerse una planeación estratégica, en la que se toma en cuenta la misión del sistema que se va a desarrollar, su ambiente, los objetivos que se persiguen con su construcción, y se elaboran estrategias. Posteriormente se lleva a cabo el análisis de requerimientos y se planea la asignación de recursos. Una vez establecido el plan de requerimientos de recursos se procede con el desarrollo del *plan del proyecto* para someterlo a su evaluación y aprobación.

Los objetivos específicos de la planificación de un proyecto se listan en las viñetas de la *figura 5.1,* dichos objetivos están encaminados a lograr el objetivo principal.

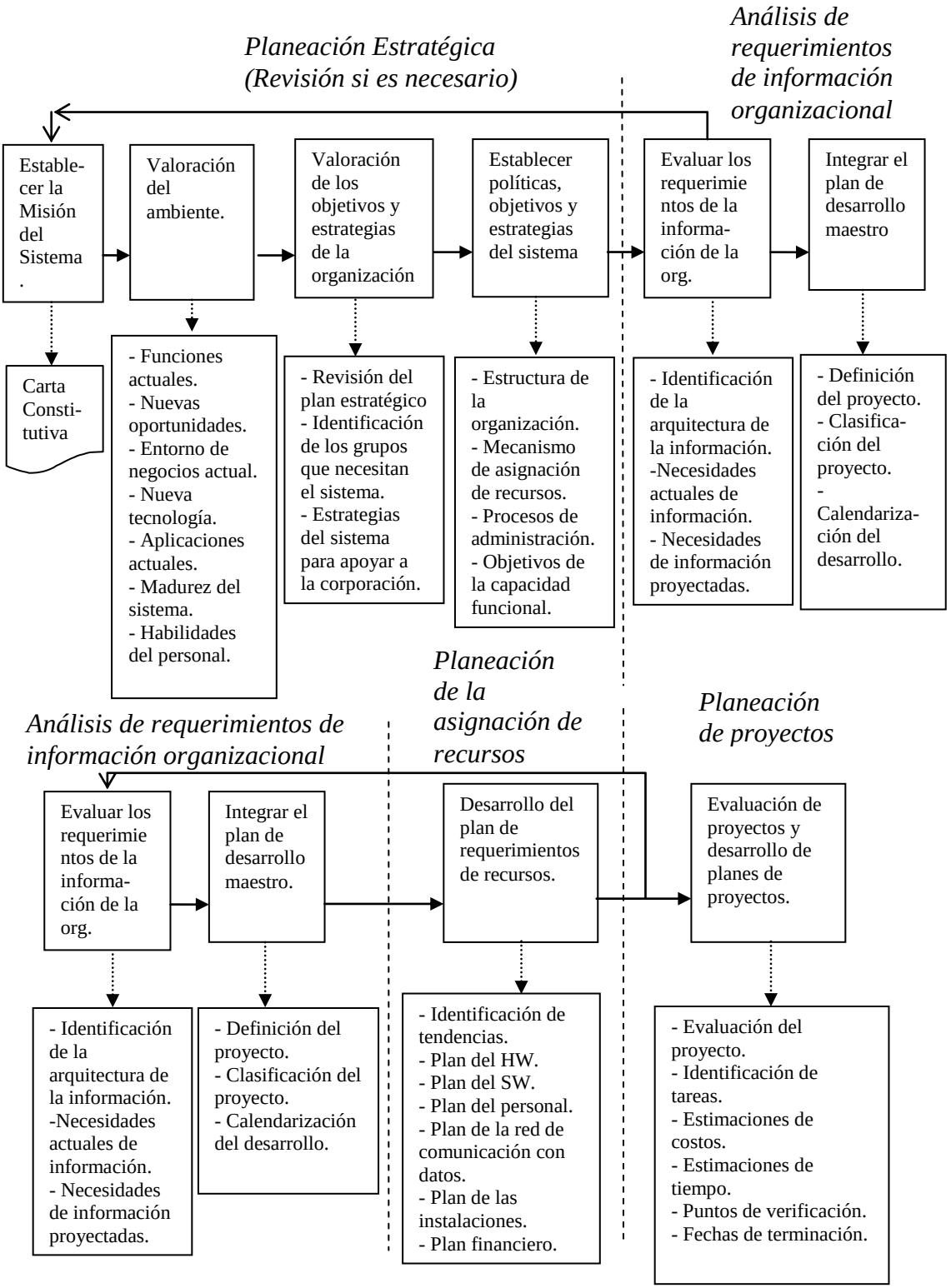


Figura 5.1: Planeación de un Sistema de Información (SI) (Senn, 1992).

Un *plan de proyecto* debe contener las tareas a realizar medidas en tiempo y número de personas necesarias, así como un calendario con fechas de inicio y terminación. También deben programarse fechas de entregas parciales a las cuales suele llamárseles *hitos* (milestones), que son los puntos de verificación que permitirán monitorear el avance del proyecto. Como se observa en la *figura 5.2*, el plan de un proyecto se compone de planes para cada uno de los siguientes aspectos:

- *Recursos humanos*.- Uso efectivo del personal involucrado, contempla entrenamiento y motivación de los empleados para que desempeñen con éxito sus labores asignadas. Incluye elaborar una matriz de roles y responsabilidades.
- *Riesgos*.- Identificar y evaluar sistemáticamente los riesgos para reducirlos y que el proyecto tenga una mayor probabilidad de éxito (la definición y análisis de *riesgos* se estudia en el capítulo VI).
- *Calidad*.- Se definen y programan inspecciones y revisiones para asegurar que se están aplicando efectivamente las prácticas de administración de proyectos. Se desarrollan planes de prueba para el control de la calidad.
- *Plan de comunicación*.- Se determina que tipo de información, a quien hay que enviarla, con que frecuencia y que formato. Se elaboran: lista de contactos, mapas y planes de comunicación.
- *Plan de métricas*.- Establecer los mecanismos para monitorear y evaluar el desarrollo del proyecto.
- *Plan de requerimientos*.- Se establecen los requerimientos, su prioridad, la(s) personas responsables de implantarlos, y formatos para reportar el grado de avance y revisión.
- *Plan de trabajo*.- Se utilizan plantillas o herramientas para poder observar el proyecto desde el más alto nivel hasta los niveles más detallados. Se establecen las actividades y la dependencia entre las mismas

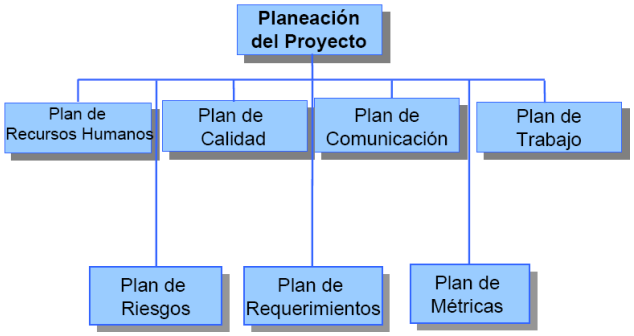


Figura 5.2: Planeación del proyecto (Briseño, 2003).

Además de los aspectos mencionados anteriormente, vale la pena señalar que dentro del plan deben contemplarse las *finanzas* y los aspectos del *contrato* tanto con clientes como con proveedores.

Finanzas.- Se establece un presupuesto y un sistema de monitoreo y control de los recursos para asegurar que el proyecto se realiza bajo el presupuesto aprobado.
Contrato.- Planificación de las adquisiciones, selección de proveedores, seguimiento al cumplimiento de los requerimientos legales, fiscales y normativos, verificación de las condiciones de contratos con clientes y proveedores.

El objetivo principal de la planificación de un proyecto es que la estimación se acerque lo más posible a la realidad. McConnell propone las siguientes opciones para optimizar la planificación de un proyecto:

- Establecer un objetivo de planificación, pero no una fecha de entrega máxima para el proyecto, hasta que se haya finalizado el diseño detallado, el diseño del producto, o al menos la especificación de requerimientos.
- Utilizar rangos de estimación o estimaciones a grandes rasgos que se van redefiniendo conforme avanza el proyecto.

También es recomendable añadir al plan de trabajo de 3 a 5 semanas de margen para cubrir los retrasos inesperados.

La actividad de planeación se compone de las tareas que aparecen en la siguiente tabla según [Métrica 3].

Tarea		Productos	Técnicas y Prácticas	Participantes
GPI 2.1	Selección de la Estrategia de Desarrollo	- Planificación General del Proyecto: o Estrategia de desarrollo		- Jefe de Proyecto
GPI 2.2	Selección de la Estructura de Actividades, Tareas y Productos	- Planificación General del Proyecto: o Estructura de actividades - Catálogo de productos a generar	- Estructura de Descomposición de Trabajo - Catalogación	- Jefe de Proyecto
GPI 2.3	Establecimiento del Calendario de Hitos y Entregas	- Planificación General del Proyecto: o Hitos del proyecto o Productos a entregar	- Planificación - Catalogación	- Jefe de Proyecto
GPI 2.4	Planificación Detallada de Actividades y Recursos Necesarios	- Planificación General del Proyecto: o Organización de los recursos o Planificación detallada del Proyecto	- Planificación	- Jefe de Proyecto
GPI 2.5	Presentación y Aceptación de la Planificación General del Proyecto	- Aceptación de la Planificación General del Proyecto	- Presentación	- Jefe de Proyecto - Comité de Seguimiento

Figura 5.3: Actividades de la planeación [métricas 3].

V.2 Estimación

La estimación de un proyecto consiste en definir aproximadamente en cuanto tiempo se va a llevar a cabo, cuantas personas se van necesitar y cual será su costo total. Como ya se mencionó anteriormente, el objetivo ideal es que la estimación se acerque lo más posible a la realidad. Si la estimación es demasiado baja, habrá retraso en la entrega y un costo mayor de lo esperado, por otra parte, si la estimación es demasiado alta, el trabajo se repartirá para cubrir el tiempo disponible así que se desperdicia tiempo y recursos.

Antes de abordar cualquier proyecto, es muy conveniente que las siguientes condiciones iniciales estén definidas:

- El objetivo del proyecto.
- El número de personas que se requerirá para llevarlo a cabo.
- El presupuesto.
- Las tareas que dependen de la realización de otras.

- Las tareas que se pueden hacer en paralelo.
- Las restricciones para la realización del proyecto.
- Identificación costos, fechas límites, y personas que aprobarán el proyecto.

V.2.1 Factibilidad

Cuando se inicia un proyecto, lo primero que se debe estimar es su factibilidad, es decir, la posibilidad de que el proyecto sea útil para la empresa u organización. [McConnell, 1997] propone descomponer en tres aspectos la factibilidad de un proyecto:

1. *Factibilidad operacional*.- ¿El sistema será aceptado y utilizado por los usuarios? ¿Existe alguna resistencia de los usuarios hacia un sistema nuevo?

2.- *Factibilidad técnica*.- ¿Se dispone de la tecnología necesaria para realizar lo que se pide?, ¿el equipo de cómputo que se propone utilizar tiene la capacidad suficiente para trabajar con los datos, la velocidad y/o las interconexiones que se requieren? ¿Hay posibilidades de expansión del sistema a futuro? ¿Hay garantías técnicas de exactitud, confiabilidad, facilidad de acceso y seguridad de los datos?

3.- *Factibilidad financiera*.- Los beneficios financieros de desarrollar e implantar un sistema nuevo deben igualar o superar los costos. En este punto cabe hacer notar que no solo hay que considerar el costo del Hardware y el Software, sino el costo de lo que sucedería en caso de que el proyecto no se llevara a cabo.

V.2.2 Estimación y control del tiempo de desarrollo

Los tres métodos más comunes para simular el tiempo de desarrollo del proyecto son los siguientes:

1.- *El método histórico*.- Se basa en registros cuidadosos del desarrollo de proyectos anteriores. En estos registros se guardan las características principales del proyecto, como asignación de tareas, requerimientos del tiempo del personal y los problemas que se presentaron. Este método, además de laborioso, solo es útil cuando el proyecto a estimar es similar a un desarrollo anterior.

2.- *El método intuitivo*.- Se basa en la experiencia del personal más antiguo, el cual estima, por medio de sus experiencias personales, el tiempo de desarrollo esperado. La popularidad de este método se debe a que es rápido y fácil de aplicar, sin embargo es muy inexacto ya que poca gente es capaz de estimar correctamente el tiempo real de desarrollo.

3.- *El método de la forma estándar*.- Se basa en métodos más concretos para la estimación. Se identifican y cuantifican los factores que afectan al tiempo de desarrollo, se evalúan las características del personal, los detalles del sistema y la complejidad del proyecto. Se usan fórmulas aritméticas que relacionan los

elementos individuales para producir una estimación del tiempo de desarrollo en horas, días o semanas.

En general, aquellos proyectos que se desarrollan a tiempo tienen las siguientes características en común:

- Una buena estimación de los requerimientos de tiempo.
- Un medio para monitorear el avance.
- Un medo para comparar el desempeño planeado con el real.
- La información suficiente para enfrentarse a los problemas cuando estos surjan.

Negociación.- Al inicio del proyecto, cuando se esta trabajando sobre la *estimación,* es necesario hacer negociaciones con el cliente para ajustar el presupuesto y el tiempo. Durante las negociaciones, es muy importante concentrarse en lo que se puede hacer y evitar caer en discusiones.

Lo mejor es dar opciones al cliente, por ejemplo: con el equipo actual de trabajo del que disponemos podemos entregar un mes más tarde de lo que se desea con todos los servicios solicitados, sin embargo podemos eliminar algunos servicios para alcanzar el tiempo deseado, o podría incorporar más personal, lo que implicaría un costo adicional, para tener todos los servicios en el tiempo que usted desea.

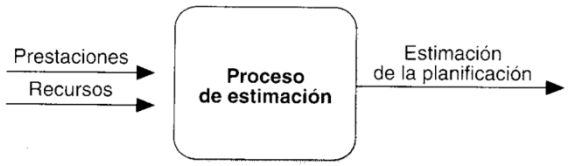


Figura 5.4: Es posible negociar las entradas, pero no es posible modificar la salida sin cambiar las entradas (McConnell, 1997)

La estimación debe ser preparada por alguien calificado, algunas veces se cae en la situación absurda de que los clientes, que no tienen idea de cómo construir un sistema de software, insisten en determinar cuanto tiempo se llevará su construcción. Es necesario insistir en que la estimación la prepare una persona o un grupo de personas con los conocimientos adecuados. En algunas empresas ha funcionado contratar a un grupo independiente de expertos que haga la estimación, esto es efectivo porque entonces no influyen intereses como entregar el producto en el menor período de tiempo posible, ni evitar que se trabaje eficientemente y a un buen ritmo.

Hay que comprometerse más con los servicios que dará el sistema más que con el tiempo de entrega, no se puede decir cuanto costará un nuevo sistema hasta que éste se conozca bastante. En el momento de la estimación inicial es mejor dejar un margen en el tiempo de entrega el cual se irá precisando conforme avanza el proyecto.

En el caso de que existan cambios en los requerimientos, será necesario volver a estimar el tiempo de entrega. Probablemente el cliente, los directivos o el equipo de marketing se opongan a cambiar la estimación sin cambiar también el conjunto de servicios o recursos, sin embargo es preferible mantenerse firme con la estimación, argumentando que se ha estudiado el caso cuidadosamente y que dar un tiempo más corto para la entrega del producto no haría que su desarrollo tardará menos tiempo, sino que solo podría asegurarse que el sistema se entregaría con retraso. Es muy recomendable ofrecer a cambio algunas de las opciones propuestas por McConnell para lograr una negociación exitosa (como se ilustra en la *figura* 5.5):

- Pasar algunas de los servicios deseados a la versión 2. Pocas personas necesitan exactamente todo lo que pidieron en el momento exacto en que lo hicieron.
- Entregar el producto por etapas, por ejemplo, versiones 0.7, 0.8, 0.9 y 1.0, incluyendo las funciones más importantes al principio.
- Eliminar completamente algunos servicios del sistema. Los servicios que consumen tiempo para su implementación y suelen ser negociables son por ejemplo; el nivel de integración con otros sistemas, el nivel de compatibilidad con sistemas anteriores y el rendimiento.
- Pulir menos algunos servicios; implementarlos hasta cierto punto, pero hacerlos menos bonitos.
- Relajar los requerimientos detallados para cada servicio. Procurar acercarse a los requerimientos al máximo posible a través del uso de componentes comerciales preconstruidos.

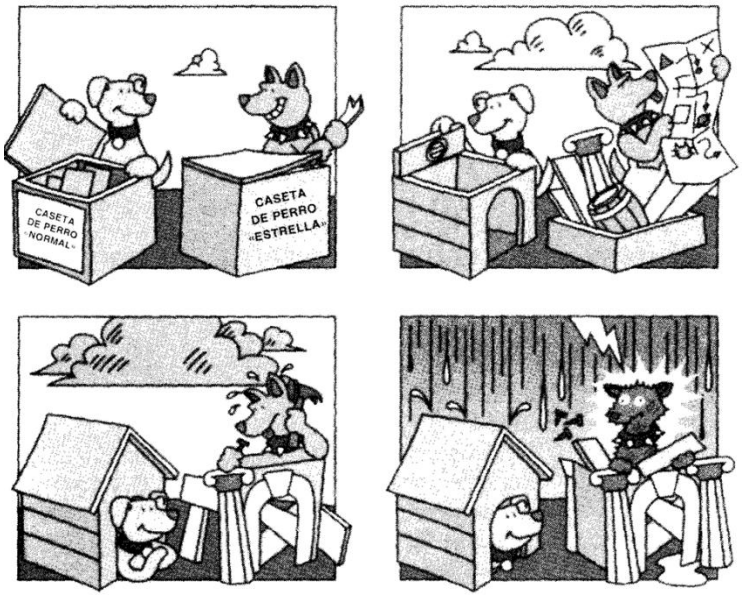


Figura 5.5: Filtrado de requerimientos (McConnell, 1997).

Para determinar el esfuerzo y el tiempo requeridos (en semanas-persona o mes-persona) dependiendo de las líneas de código, hay libros que contienen docenas de fórmulas y ecuaciones con las que el administrador puede necesitar varios días para aprender a realizar una estimación simple. Si bien existen programas de estimación que ayudan a hacer estimaciones, éstos tienden a ser caros (varios miles de dólares). La tabla de la *figura 5.6* es muy útil para hacer estimaciones sencillas de cuanto tiempo y cuantas personas se necesitarán para hacer un programa de un tamaño determinado. La tabla está elaborada bajo la hipótesis de que el equipo de desarrolladores tiene un talento de medio a alto, los miembros del equipo están familiarizados con el lenguaje de programación y medianamente familiarizados con el ambiente de desarrollo. La tabla contiene la estimación para tres clases de proyectos:

- *Software de sistemas.*- Comprende sistemas operativos, controladores de dispositivos, compiladores, bibliotecas de código, sistemas en tiempo real y software científico.
- *Software de gestión.*- Aquellos que se utilizan en una empresa, por ejemplo sistemas de control de nóminas, contabilidad y control de almacén.
- *Software "pret*-à-porter".- Se vende comercialmente para un propósito específico, como: procesador de texto, hoja de cálculo, programas de análisis financiero, escritura de guiones, gestión de información legal, etc.

	Productos de sistemas		Productos de gestión		Productos «prêt-à-porter»	
Tamaño del sistema (líneas de código)	Planificación (meses)	Esfuerzo (personas-mes)	Planificación (meses)	Esfuerzo (personas-mes)	Planificación (meses)	Esfuerzo (personas-mes)
10.000	10	48	6	9	7	15
15.000	12	76	7	15	8	24
20.000	14	110	8	21	9	34
25.000	15	140	9	27	10	44
30.000	16	185	9	37	11	59
35.000	17	220	10	44	12	71
40.000	18	270	10	54	13	88
45.000	19	310	11	61	13	100
50.000	20	360	11	71	14	115
60.000	21	440	12	88	15	145
70.000	23	540	13	105	16	175
80.000	24	630	14	125	17	210
90.000	25	730	15	140	17	240
100.000	26	820	15	160	18	270
120.000	28	1.000	16	200	20	335
140.000	30	1.200	17	240	21	400
160.000	32	1.400	18	280	22	470
180.000	34	1.600	19	330	23	540
200.000	35	1.900	20	370	24	610
250.000	38	2.400	22	480	26	800
300.000	41	3.000	24	600	29	1.000
400.000	47	4.200	27	840	32	1.400
500.000	51	5.500	29	1.100	35	1.800

Fuentes: Confeccionado a partir de los datos de *Software Engineering Economics* (Boehm, 1981), «An Empirical Validation of Software Cost Estimation Models» (Kemerer, 1987), *Applied Software Measurement* (Jones, 1991), *Measures for Excellence* (Putnam y Myers, 1992) y *Assessment and Control of Software Risks* (Jones, 1994).

Figura 5.6: Esfuerzo estimado en función del tipo de proyecto y del tamaño del sistema (McConnell, 1997).

La estimación del esfuerzo para llevar a cabo un proyecto será de mayor calidad mientras mejor sea su definición, la estimación puede darse con valores puntuales, por ejemplo: 100 personas-mes, o con rangos: 80-120 personas-mes.

V.2.3 Recursos.

La administración de los recursos tiene como fin optimizar el personal, el dinero y el tiempo empleados para lograr los objetivos del proyecto. Para aumentar los recursos de un proyecto, McConnell propone las siguientes prácticas:

- Incorporar más desarrolladores, si se está al comienzo de la planificación.
- Incorporar desarrolladores de alto rendimiento (por ejemplo. expertos en áreas del dominio).
- Incorporar más probadores.
- Incorporar más soporte administrativo.

- Aumentar el grado de apoyo al desarrollador. Asignarle oficinas tranquilas, más aisladas, computadoras más rápidas, soporte técnico a mano para las redes y equipos, aprobación para utilizar servicios más caros de ayuda al desarrollador, etc.
- Eliminar el papeleo de la compañía. Establecer un proyecto "en la sombra".
- Aumentar el nivel de participación de los usuarios finales. Hacer que un usuario final se dedique de tiempo completo al proyecto, autorizarlo a tomar decisiones sobre el conjunto de servicios del producto.
- Aumentar el nivel de participación de los directivos.

V.3 Medidas, métricas e indicadores

Primero definiremos cada uno de los términos estableciendo sus diferencias:

Las medidas: es la recopilación de datos, los cuales son valores independientes y cuantitativos. Las medidas se hacen sobre el producto, el proceso y el proyecto.

Las métricas: se basan en las medidas para obtener datos cuantitativos sobre la calidad y productividad del proceso y del producto.

Los indicadores: evalúan las métricas y arrojan conclusiones sobre el estado del proyecto. Los indicadores le dan la pauta al administrador del proyecto par ajustar lo necesario y/o emprender acciones correctivas.

Las medidas, métricas e indicadores ofrecen múltiples beneficios, en primer lugar son el punto de partida para poder hacer mejores estimaciones y para diseñar planes de trabajo. Además permiten evaluar el proyecto actual y compararlo con proyectos anteriores, las medidas mejoran la visibilidad del progreso y ayudan a disminuir riesgos. Además, las medidas sientan las bases para la mejora del proceso a largo plazo, ya que se pueden comparar los proyectos y analizar los métodos que funcionan y los que no.

Podemos clasificar las medidas de acuerdo con la *figura 5.7*:

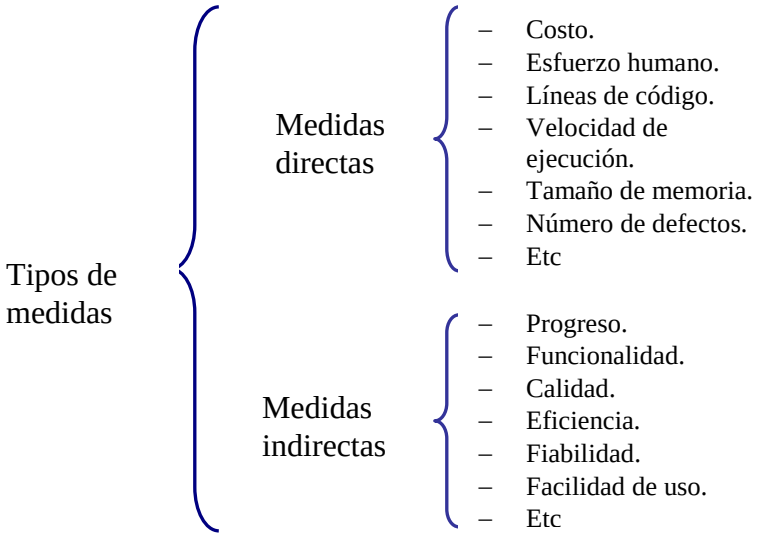


Figura 5.7 Medidas directas e indirectas (Pressman, 2006).

Es necesario poner especial cuidado al elegir los factores que se van a medir y a la interpretación de las mediciones, evitando caer en la sobrevaloración de la medida de un solo factor (por ejemplo, el de la *figura* 5.8).

Las medidas deben ser:

- **Exactas**
- **Precisas**: No se debe perder información en los redondeos ya que la información se desvirtúa.
- **Consistentes**: Una medición de un atributo debe dar el mismo valor independientemente de la medición.
- **Comparables**: Para ello, debe estar normalizada.

La adopción de métricas en un proceso de desarrollo de software tiene dos fases: la de aprendizaje y la de uso. Durante la fase de aprendizaje se emplea mucho esfuerzo con poco beneficio para detectar cuales son las mediciones útiles, pero una vez superada esta fase, cuando finalmente se obtienen métricas útiles, el beneficio aumenta considerablemente. Una forma práctica de entrar rápidamente a la fase de uso es el establecimiento de una "línea de base de métricas" la cual consiste en datos recopilados en proyectos previos de desarrollo de software, estos pueden ir desde los datos más simples, como una tabla sencilla, hasta una compleja base de datos con decenas de medidas de proyectos anteriores.

A pesar de sus beneficios evidentes, también es posible cometer el error de hacer mal uso de las medidas, por ejemplo al evaluar a los empleados. Las medidas del desempeño de las personas ayudan a que éstas se concentren sobre objetivos específicos, por ejemplo, la medida del número de errores cometidos por persona ayudará a que éstas trabajen con mayor cuidado, la medida del número de módulos terminados ayudará a que las personas no divaguen y se concentren en producir resultados concretos.

La medida del progreso es una de las más importantes, ya que no saber que hay retraso es aún más grave que el retraso en sí mismo.

"La mala noticia es que usted fue el que menos ha trabajado y la buena noticia es que usted fue el que menos errores cometió".

Figura 5.8: medición de los errores (Glasbergen, 1996: www.glasbergen.com)

Las *métricas* proporcionan los elementos para:

- Indicar la calidad del producto.
- Evaluar la productividad de las personas.
- Evaluar los beneficios derivados del uso de nuevos métodos y herramientas.
- Establecer una línea de base para la estimación.
- Justificar el uso de nuevas herramientas y la necesidad de formación.

Los *indicadores* proporcionan los elementos para:

- Evaluar el estado del proyecto.
- Hacer un seguimiento de los riesgos.
- Detectar las áreas problemáticas.
- Ajustar el flujo y las tareas del trabajo para evitar retrasos.
- Evaluar la habilidad del personal.
- Evaluar la calidad del producto.

Las métricas del software se clasifican según la *figura 5.7* En la *figura 5.9* podemos apreciar ejemplos de las medidas orientadas al tamaño:

Medidas orientadas al tamaño

- Esfuerzo humano (personas-mes)
- Costo.
- Número de errores.
- Páginas de documentación.
- Etc.

Figura 5.9 Medidas orientadas al tamaño (Pressman, 2006).

Las *métricas orientadas al tamaño* se deben normalizar, ya sea con respecto a Líneas De Código (LDC), por ejemplo: costo/LDC o No. de errores/LDC, o con respecto al esfuerzo humano medido generalmente en personas-mes (PM), por ejemplo: costo/pm o No. de errores/pm.

Las *medidas orientadas a la función* evalúan el proyecto a partir de un parámetro denominado *punto de función*. Obtener estas medidas no es fácil, ya que éstas son indirectas y dependen de varios factores.

Un *punto de función* es una unidad de medida para expresar la cantidad de funcionalidad de negocio un sistema de información proporciona a un usuario. El costo (en dólares u horas) de una sola unidad se calcula a partir de proyectos anteriores.

Existe un método de estimación de costos mediante los puntos de función llamado "Análisis de Puntos de Función", éste método cuantifica la funcionalidad que hay que entregar al usuario al construir una aplicación. Este método de puntos de función se utiliza para medir el tamaño del software, pretende medir la funcionalidad entregada al usuario independientemente de la tecnología utilizada para la construcción y explotación del software, y también ser útil en cualquiera de las fases de vida del software, desde el diseño inicial hasta la explotación y mantenimiento.

Las métricas orientadas a Puntos de Función se caracterizan por:

- Tener un **componente empírico**, basado en la experiencia de muchos proyectos.
- Tener **en cuenta la complejidad**, aunque esta sea muy difícil de determinar en un proyecto.
- Ser **independientes del entorno tecnológico** y de las metodologías aplicadas.
- Utilizar **medidas indirectas**, que se caracterizan por ser subjetivas y difíciles de calcular, sin embargo el resultado obtenido es fácilmente comparable.

En la *figura 5.10* podemos observar los 14 Factores que contribuyen a la complejidad de una aplicación y que se utilizan para el cálculo de los puntos de función.

Factores de complejidad (FC)	0-5	Factores de complejidad (FC)	0-5
Comunicación de datos		Funciones distribuidas	
Rendimiento		Gran carga de trabajo	
Frecuencia de transacciones		Entrada *on-line* de datos	
Requisitos de manejo del usuario final		Actualizaciones *on-line*	
Procesos complejos		Utilización con otros sistemas	
Facilidad de mantenimiento		Facilidad de operación	
Instalación en múltiples lugares		Facilidad de cambio	

Figura 5.10: Factores de complejidad de una aplicación (Pucha, L.)

Cada uno de estos valores debe evaluarse del 0 al 5 según la tabla de la *figura 5.11*.

Valor	Significado del valor
0	Sin influencia, factor no presente
1	Influencia insignificante, muy baja
2	Influencia moderada o baja
3	Influencia media, normal
4	Influencia alta, significativa
5	Influencia muy alta, esencial

Figura 5.11: Valores posibles de los factores de complejidad.

La complejidad de los puntos de función se establece mediante fórmulas y criterios publicados por el Grupo Internacional de Usuarios de los Puntos de Función (IFPUG: por sus siglas en inglés), para mayores referencias consultar: [IFPUG, Symons, 1991].

Existen también métricas Orientadas a Objetos, y métricas de Calidad que pueden consultarse con más detalle en [Pressman, 2006], quien además hace la siguiente cita:

"Las métricas nos permiten reconocer cuando reír y cuando llorar". Tom Gilb.

"No todo lo que puede ser contado cuenta, y no todo lo que cuenta puede ser contado." Albert Einstein.

V.4 El Modelo Constructivo de Costos (COCOMO)

Los costos se pueden calcular como una función matemática basada en atributos de productos, proyectos y procesos, cuyos valores son calculados por administradores de proyectos. COCOMO (COnstructive COst MOdel) es un modelo matemático empírico obtenido gracias a la recolección de datos que provienen de proyectos de software anteriores. Después de analizar estos datos se descubrieron fórmulas que se ajustaron a las observaciones. COCOMO pretende establecer una relación matemática con la que sea posible estimar el esfuerzo (hombre-mes) y el tiempo requerido para desarrollar un proyecto.

Barry Boehm, en su libro sobre "Economía de la Ingeniería del Software", menciona una escala de modelos de estimación de software con el nombre de COCOMO. La estimación es más precisa a medida que se toman en cuenta mayor cantidad de factores que influyen en el desarrollo de un producto de software [Pressman, 1993].

Para hacer una estimación efectiva es necesario estimar primero el tamaño del software y después la planeación. Existen varias formas para estimar el tamaño de un proyecto, entre ellas:

- Utilizar un enfoque algorítmico, como los puntos de función, para estimar el tamaño del programa a partir de los requerimientos.

- Utilizar un software para estimar el tamaño del programa, a partir de la descripción de los requerimientos del programa (interfaces interactivas, funcionalidad lógica, persistencia de datos, etc.)

Una vez que se tiene la **estimación del tamaño**, se puede pasar al segundo paso, que es la **estimación del esfuerzo**. Ésta es necesaria para poder saber a cuántas personas hay que incorporar en el proyecto; además, con una ésta se facilita la **estimación de la planeación**.

La estimación del esfuerzo requerido para un proyecto usando COCOMO se basa en la experiencia, es decir, se fundamenta en la observación de los hechos anteriores. El dato clave para hacer las estimaciones es el número de líneas de código fuente: LDC. La cantidad de Líneas de Código se debe estimar por experiencia, por analogía con otros proyectos semejantes, o por otros datos que se posean.

El modelo original COCOMO se público por primera vez en 1981 por Barry Boehm y reflejaba las prácticas en desarrollo de software de aquel momento, El término *constructive* se refiere a al hecho de que el modelo permite al estimador obtener un mejor entendimiento de la complejidad del desarrollo software. En la década y media siguiente las técnicas de desarrollo software cambiaron drásticamente. Estos cambios incluyen el gasto del esfuerzo en diseñar y gestionar el proceso de desarrollo software así como de la creación del producto software. Un giro total desde los mainframe que trabajan con procesos batch nocturnos hacia los sistemas en tiempo real y un énfasis creciente en la reutilización de software ya existente y en la construcción de nuevos sistemas que utilizan componentes software a medida. Para que COCOMO fuera apto para ser aplicado en proyectos vinculados a tecnologías como orientación a objetos, desarrollo incremental, composición de aplicación, y reingeniería se hizo un nuevo modelo al que se le llamó COCOMO II, el cual es un modelo que permite estimar el costo, el esfuerzo y el tiempo cuando se planifica una nueva actividad de desarrollo software. Está asociado a los ciclos de vida modernos. Para evitar confusión, el modelo COCOMO original fue designado con el nombre COCOMO' 81. Así todas las referencias de COCOMO encontradas en la literatura antes de 1995 se refieren a lo que ahora llamamos COCOMO'81. La mayoría de las referencias publicadas a partir de 1995 se refieren a COCOMO II [Center of Systems and Software Engineering].

La escala de modelos de Boehm contiene tres niveles: Básico, Intermedio y Avanzado. Esta familia de modelos COCOMO son una alternativa para la estimación del Esfuerzo personas-mes y de la Planeación en meses a partir del Tamaño del software en miles de líneas de código (KLDC) y de otros aspectos relacionados con los costos.

Los indicadores de planificación que se pueden obtener con este método son:

- Esfuerzo (hombre-mes)
- Tiempo de desarrollo (meses)
- Personal necesario (hombres)
- Productividad (inst/hombre-mes)
- Costo (pesos)

La unidad de esfuerzo Hombre-Mes supone un total de 152 horas de trabajo por persona, en base a la experiencia práctica y a consideraciones sobre vacaciones, permisos, enfermedad, etc. Además, se tiene que:

Hombres-Mes x 152 = Hombres-Hora
Hombres-Mes x 19 = Hombres-Día
Hombres-Mes / 12 = Hombres-Año

En el modelo COCOMO establece tres *niveles* que son:

- Nivel básico.
- Nivel intermedio.
- Nivel detallado.

Pero además se tienen tres *modos* los cuales son:

- Orgánico o familiar.
- Semilibre o Semiacoplado.
- Fuertemente restringido o empotrado.

En el ***modo orgánico o familiar***, el equipo de desarrollo es relativamente pequeño. La gran mayoría de la gente relacionada con el proyecto tiene una amplia experiencia en otros proyectos relacionados con la misma organización por lo que se tiene una idea bastante buena de la manera en la que el sistema bajo desarrollo, contribuirá a los objetivos de su organización. La mayoría de las personas puede contribuir de forma efectiva a que cada una de las etapas se termine a tiempo sin que se requieran grandes necesidades de comunicación para determinar con precisión las tareas que cada uno debe desarrollar en el proyecto. Además, el equipo de trabajo puede negociar con facilidad la modificación de algunas de las especificaciones para hacer más fácil este desarrollo. La motivación para terminar el proyecto de software antes de lo previsto es mínima. Los proyectos son de un tamaño relativamente pequeño, como máximo 50 KDSI (miles de instrucciones) y generalmente la productividad es alta en este modo de desarrollo.

El ***modo semilibre o semiacoplado*** representa un estado intermedio entre el modo orgánico y el modo fuertemente restringido, en este modo todos los miembros del equipo de diseño tienen un nivel medio de experiencia en sistemas relacionados con el proyecto. El equipo de desarrollo esta formado por una mezcla de gente experta e inexperta. El equipo de desarrollo del proyecto tiene un nivel medio de experiencia en proyectos similares. El proyecto tiene fuertes restricciones, que pueden estar relacionadas con los requerimientos (funcionales y no funcionales) del software o con los requerimientos del hardware como el procesador y las diversas interfaces con hardware externo. Los proyectos generalmente son sistemas interactivos distribuidos, donde la persistencia y las transacciones de datos son de gran importancia. El tamaño de los proyectos puede llegar hasta las 300 KDSI.

En el ***modo fuertemente restringido o empotrado*** los proyectos deben desarrollarse dentro de limitaciones sumamente estrictas. Dependiendo del problema, el grupo puede incluir una mezcla de personas experimentadas y no experimentadas. El producto debe operar dentro de un entorno muy acoplado de hardware, software, normativas y

procedimientos operativos, tales como sistemas de transferencia electrónica de fondos o control de tráfico aéreo. En estos proyectos no existe la posibilidad de negociar fácilmente cambios en el software y en tal caso precisará un mayor tiempo para acomodar o asegurar que los cambios cumplan las especificaciones (mayor costo de verificación, validación y de gestión de la configuración). Los costos de cambiar algo son tan elevados que sus características se consideran inmodificables, por lo tanto el software debe desarrollarse siguiendo estrictamente todas las especificaciones. Estos proyectos se desarrollan en áreas generalmente desconocidas, lo cual lleva inicialmente a equipos pequeños de analistas y a una sobrecarga de comunicación durante el desarrollo. Este modo se puede aplicar a proyectos de cualquier tamaño.

A continuación se describen los tres niveles de COCOMO [Center of Systems and Software Engineering, Pressman, 2006; González, 2001] y algunos ejemplos de aplicación.

V.4.1 COCOMO Básico

El modelo COCOMO básico calcula el esfuerzo y el costo del desarrollo de software en función del tamaño del programa, expresado en las líneas estimadas de código (LDC). No toma en cuenta los otros factores que afectan al proyecto, sin embargo, es adecuado para realizar estimaciones de forma rápida aunque sin gran precisión. Este modelo únicamente se basa en el Tamaño del software en KLDC, se suele aplicar en los desarrollos de productos pequeños/medios, desarrollados por personal de una organización propia o empresa en modo orgánico.

Expresión para la estimación del Esfuerzo personas-mes.

$$E = a_b KLDC^{b_b}$$

Donde:

KLDC es el número de Línea de Códigos en miles, a_b y b_b son el coeficiente y exponente, para este modelo y E es el Esfuerzo estimado.

Expresión para la estimación la Planeación en meses.

$$D = c_b E^{d_b}$$

Donde:

E es el Esfuerzo personas-mes estimado, c_b y d_b son el coeficiente y exponente, para este modelo y D es la Planeación en Meses estimada.

Coeficientes y exponentes para el modelo COCOMO básico. [Fenton, 1991].

Proyecto de software	a_b	b_b	c_b	d_b
Orgánico (pequeños en tamaño y complejidad)	2.4	1.05	2.5	0.38
Semiacoplado (intermedio en tamaño y complejidad)	3.0	1.12	2.5	0.35
Empotrado (desarrollados para un tipo de hardware particular con fuertes restricciones operativas)	3.6	1.20	2.5	0.32

Ejemplo.- Queremos desarrollar un software de gestión de información, se ha estimado tendrá 32 KLDC y en base a las características de la aplicación se ha decidido manejarlo en el modo orgánico.
¿Cuáles serán el esfuerzo, tiempo y recursos requeridos para desarrollar dicha aplicación?
Esfuerzo:

$$E = a_b KLDC^{bb}$$

$$E = 2{,}4\ (32)^{1{,}05}$$

$$\underline{E = 91 \text{ personas-mes}}$$

Tiempo:

$$D = c_b E^{db}$$

$$D = 2{,}5(91)^{0{,}38}$$

$$\underline{D = 14 \text{ Meses}}$$

Empleados:

E/D
91 / 14
<u>E/D = 6,5 Personas</u>

V.4.2 COCOMO Intermedio

El modelo COCOMO intermedio calcula el esfuerzo del desarrollo de software en función del tamaño del programa y de un conjunto de "conductores de costo" que incluyen la evaluación subjetiva del producto, del hardware, del personal y de los atributos del proyecto. Los factores como: calidad, experiencia del personal, restricciones de hardware, utilización de técnicas modernas y herramientas de desarrollo se consideran como adicionales al costo total del proyecto. El modelo intermedio incorpora 15 variables de predicción que influyen en el costo del proyecto. Estas variables se agrupan en cuatro categorías: atributos del producto software, atributos del equipo de computo, atributos de personal y atributos del proyecto.

Atributos del Producto Software

RELY: Fiabilidad requerida del software.

Se puede definir como la probabilidad de que el software realice sus funciones satisfactoriamente en su próxima ejecución durante un periodo dado de tiempo. La influencia se clasifica en: Muy Alto, Alto, Nominal, Bajo y Muy Bajo, en función del efecto que tenga un fallo del producto. Un rango Muy bajo, se usa cuando el error de codificación puede ser eliminado por los programadores sin que tenga alguna consecuencia, cuando exista un caso en que vidas humanas dependan del software la influencia debe clasificarse como Muy Alto.

DATA: Tamaño de la base de datos.

Esta variable indica el tamaño y complejidad de la base de datos. Se expresa mediante la proporción:

Data = Tamaño de la base de datos en caracteres / Tamaño del programa en DSI.

Este parámetro puede tomar los valores Bajo, Nominal, Alto y Muy alto.

CPLX: Complejidad del Producto.

Mide la complejidad en función de las funciones de control, cálculos, gestión de datos y operaciones dependientes de dispositivos. El rango de este parámetro puede variar desde Muy bajo si el módulo utiliza expresiones matemáticas simples a Extra Alto si se emplean varios módulos con ejecución dinámica.

Atributos del equipo de cómputo.

TIME: Limitaciones en el Tiempo de Ejecución.

Se refiere a las limitaciones de uso de máquina del producto considerado. Se expresa en términos del porcentaje de tiempo de ejecución disponible que se espera sea usado por el sistema o subsistema. Es Nominal cuando se usa menos del 50% del tiempo y Extra alto cuando se consume el 95%.

STOR: Limitaciones de Memoria Principal.

Se expresa en términos de las restricciones de almacenamiento principal. El rango varía desde Nominal si se espera una restricción de memoria de menos del 50% hasta Extra alto si la reducción es del 95%.

VIRT: Volatilidad de la Máquina Virtual.

Se entiende por máquina virtual el conjunto de hardware y software que el producto utiliza para realizar su tarea. Durante el desarrollo esta máquina puede sufrir cambios. El rango de su variabilidad va desde Bajo hasta Muy Alto, en función de estos cambios.

TURN: Tiempo de respuesta experimentado por el equipo que desarrolla el proyecto.

Está definido por el tiempo medio de respuesta en horas desde que el desarrollador introduce un trabajo en la computadora hasta que obtiene los resultados del proceso. Estos factores han perdido parte de su importancia en los entornos actuales de desarrollo y explotación ya que estas limitaciones se producen sólo en el desarrollo de productos en que no es posible utilizar herramientas de productividad o desarrollos en entornos batch. El rango varía desde Bajo para un sistema interactivo hasta Muy Alto cuando el tiempo de respuesta es mayor de 12 horas.

Atributos de Personal

ACAP: Capacitación de los Analistas.
Expresa en términos de percentiles con relación al conjunto de analistas los siguientes atributos:

- Habilidad para el análisis.
- Eficiencia y calidad en el trabajo.
- Habilidad para comunicarse y cooperar.

Se evalúa su eficiencia trabajando en equipo. Cuanto más capaz sea el equipo de analistas, menor será el esfuerzo necesario. El rango de este parámetro puede variar entre Muy Bajo y Muy Alto.
AEXP: Experiencia en Aplicaciones.
Indica el nivel de experiencia en aplicaciones del equipo de desarrollo de proyectos. El rango varía desde Muy Bajo (menos de cuatro meses de experiencia) y Muy Alto (más de 12 años).

PCAP: Capacitación de los Programadores.
Expresa similares atributos que el parámetro ACAP pero para los programadores.

VEXP: Experiencia en la Máquina Virtual.
Es el tiempo de experiencia en el entorno Hardware y Software del equipo que desarrolla el software. No se considera el lenguaje de programación. Este parámetro puede tomar los valores desde Muy Bajo (si la experiencia en la máquina es menor de un mes) hasta Alto (si es mayor de tres años).

LEXP: Experiencia en el Lenguaje de Programación.
Es la experiencia del equipo de programadores en un lenguaje de programación determinado. El rango de este parámetro puede variar desde Muy Bajo hasta Alto para un equipo con un mes hasta tres años de experiencia.

Atributos del Proyecto

MODP: Prácticas Modernas de Programación.
Señala el grado de utilización de prácticas modernas de programación entendiendo por tal:

- Análisis de Requisitos y Diseño Top-Down
- Diseño Estructurado.
- Desarrollo Incremental.
- Revisiones o Inspecciones de Diseño y Código.
- Programación Estructurada.
- Librerías de Programas.

Se valorará el grado de utilización de estas prácticas desde Muy Bajo hasta Muy Alto.

TOOL: Uso de herramientas para el Desarrollo de Software.
Señala el grado de utilización de herramientas en el desarrollo al software, existen cinco niveles de herramientas:

- Herramientas básicas de microprocesador.
- Herramientas básicas de microcomputador.
- Herramientas potentes de microcomputador.
- Herramientas potentes de servidor/computador central.
- Herramientas avanzadas.

El rango de este parámetro varía entre Muy Bajo cuando sólo se usan herramientas básicas, hasta Muy Alto cuando se usan herramientas de propósito especial como CASE (Computer Aided Software Engineering).

SCED: Limitaciones en la planificación.
Se define mediante el porcentaje de retraso o aceleración con respecto a la planificación nominal impuesta al equipo de desarrollo. Cualquier aceleración (Muy Bajo) o retraso (Muy Alto) requerirá mayor esfuerzo.

Estimación con el Modelo Intermedio.- Las 15 variables anteriormente mencionadas influyen en la estimación de esfuerzo. El esfuerzo calculado se ajusta al multiplicarlo por el resultado del producto de los valores obtenidos de las tablas de atributos en función de los valores identificados en la definición del proyecto.

Tabla de Multiplicadores de esfuerzo [Fenton, 1991].

Variable	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Extra
RELY	0.75	0.88	1.0	1.15	1.40	
DATA		0.94	1.0	1.08	1.16	
CPLX	0.70	0.85	1.0	1.15	1.30	1
TIME			1.0	1.11	1.30	1
STOR			1.0	1.06	1.21	1
VIRT		0.87	1.0	1.15	1.30	
TURN		0.87	1.0	1.07	1.15	
ACAP	1.46	1.19	1.0	0.86	0.71	
AEXP	1.29	1.13	1.0	0.91	0.82	
PCAP	1.42	1.17	1.0	0.86	0.70	
VEXP	1.21	1.10	1.0	0.90		
LEXP	1.14	1.07	1.0	0.95		
MODP	1.24	1.10	1.0	0.91	0.82	
TOOL	1.24	1.10	1.0	0.91	0.83	
SCED	1.23	1.08	1.0	1.04	1.10	

En el COCOMO intermedio, la ecuación para calcular el tiempo de desarrollo es la misma que la del COCOMO básico. La ecuación para calcular el esfuerzo es:

$$E = a * \text{KLOC}^{b} * \text{EAF}$$

Donde ***E*** es el esfuerzo en personas-mes, KLOC es el número estimado de miles de líneas de código. ***EAF*** es el *coeficiente de adaptación del esfuerzo.*
Coeficientes y exponentes para el modelo COCOMO Intermedio [Fenton, 1991].

Proyecto de software	a	b
Orgánico (pequeños en tamaño y complejidad)	3.2	1.05
Semiacoplado (intermedio en tamaño y complejidad)	3	1.12
Empotrado (desarrollados para un tipo de hardware particular con fuertes restricciones operativas)	3.8	1.2

Ejemplo.- Se debe desarrollar un software de complejidad elevada de 10 KDSI para un equipo de cómputo comercial. El tipo de software es de comunicaciones, la codificación será demasiado compleja. Durante la planeación del proyecto se contrato a personal

altamente capacitado y mucha experiencia, los costos de los salarios se incrementaran debido a las capacidades del equipo pero gracias a esto los costos debidos a la complejidad reducirán.

Se debe determinar el esfuerzo y el costo de desarrollo si el precio medio es de 25,000 USD personas-mes.

Aplicaremos el Modo Orgánico para calcular el esfuerzo, dado que el número de líneas de código no excede las 50 KLOC.

El Coeficiente de adaptación del esfuerzo se calcula de la siguiente forma [Fenton, 1991]:

Factor	Situación	Ratio	Multiplicador
RELY	*Normal*	NOMINAL	1
DATA	*20.000 bytes*	BAJO	0.94
CPLX	*Comunicaciones*	MUY ALTO	1.3
TIME	*70% de uso*	ALTO	1.11
STOR		ALTO	1.06
VIRT		NOMINAL	1
TURN		NOMINAL	1
ACAP	*Analistas con experiencia*	ALTO	0.86
AEXP	*4 años*	NOMINAL	1
PCAP	*Programadores con experiencia.*	ALTO	0.86
VEXP	*Poca experiencia*	BAJO	1.1
LEXP	*3 meses.*	NOMINAL	1
	1 año y buenas prácticas de programación	ALTO	0.91
TOOL	*Herramientas CASE e IDE*	BAJO	1.1
SCED	*Nueve meses*	NOMINAL	1

Coeficiente de adaptación del esfuerzo:

EAF = (1 * 0.94 * 1.3 * 1.11 * 1.06 * 1 * 1 * 0.86 * 1 * 0.86 * 1.1 * 1 * 0.91 * 1.1 * 1)

EAF = 1.17

Esfuerzo:

$E = a * KLOC^{b} * EAF$

$E = 3.2 * 10^{1.05} * 1.17$

E = 42.0083709291 personas-mes.

Tiempo:

$D = c_b E^{db}$

$D = 2{,}5(42)^{0{,}38}$

D = 10.3 Meses

Empleados:

E/D
42 / 10.3
E/D = 4.07 Personas

Costo:

USD = (42 personas-mes) (25000 USD) = 1 050 000 USD.

V.4.3 COCOMO Detallado

El modelo COCOMO Detallado incorpora todas las características de la versión intermedia y lleva a cabo una evaluación del impacto de los conductores de costo en cada fase del transcurso de ingeniería del software (análisis, diseño, etc.). Se considera que los factores a tomar en cuenta afectan a cada una de las fases del proyecto.
El modelo intermedio tiene dos limitaciones que pueden ser significativas en la estimación detallada de costo en grandes proyectos de software:

- La distribución de esfuerzo por fases puede ser inadecuada.
- Puede ser muy engorroso utilizarlo en un producto con muchos componentes.

El modelo detallado presenta dos funcionalidades que resuelven las limitaciones de COCOMO Intermedio:

- Multiplicadores de esfuerzo por fases:

En el modelo COCOMO Intermedio, la distribución de esfuerzo por fase se determina únicamente por el tamaño del producto. En la práctica, factores como la fiabilidad requerida, la experiencia en aplicaciones y desarrollos interactivos afectan a unas fases más que a otras, por lo tanto ahora en el modelo detallado se proporcionan un conjunto de multiplicadores de esfuerzo para cada atributo los cuales determinaran el esfuerzo requerido para completar cada fase.

- Descomposición jerárquica del producto a tres niveles.

En el COCOMO Intermedio, el coeficiente de adaptación del esfuerzo se calcula para distintos componentes del software producto. Este proceso puede ser muy tedioso e innecesariamente repetitivo si ciertos componentes son agrupados en subsistemas con prácticamente el mismo coeficiente.
El COCOMO Detallado evita este problema proporcionando una jerarquización del producto a tres niveles:

1.- El Nivel módulo se describe por el número de instrucciones (DSI) producidas por aquellos factores que tienden a modificar dicho nivel: complejidad del módulo y adaptación a partir del software existente, de la capacidad y de la experiencia de los programadores que desarrollarán el módulo en el lenguaje y en la máquina virtual.

2.- El Nivel subsistema queda descrito por los restantes factores (limitaciones en

tiempo y memoria, capacidad de los analistas, herramientas, planificación etc.) que tienden a variar de un subsistema a otro, pero que son iguales para todos los módulos dentro de un subsistema.

3.- El Nivel sistema se define mediante los factores correspondientes al proyecto, como son el esfuerzo nominal y la planificación de tiempos.

Este modelo incorpora todas las características de la versión intermedia y se centra en el impacto que tendrá el costo en cada paso del proceso de desarrollo de software (análisis, diseño, etc.). En estas notas no se desarrolla este modelo dado que su aplicación queda reservada a proyectos muy grandes, los cuales no son muy frecuentes. El desarrollo completo de este modelo se encuentra en el libro "Software Engineering Economics" de B. Boehm, editorial Prentice Hall, 1981.

V.5 Errores clásicos relacionados con el producto

A continuación se muestran los errores clásicos relacionados con la forma en la que se define el producto según [McConnell, 1997]:

- **Exceso de requerimientos**.- Algunos proyectos tienen desde el inicio más requerimientos de los que necesitan. Un ejemplo es “la eficiencia”, que se fija como requisito más a menudo de lo que es necesario, y puede generar una planificación del software innecesariamente larga. Los usuarios tienden a interesarse menos en las funcionalidades complejas que en las de las secciones de marketing o de desarrollo. Las complejas alargan desproporcionadamente el plan de desarrollo.
- **Cambio de los requerimientos**.- Los proyectos sufren muy a menudo cambios en los requerimientos a lo largo de su vida, lo que puede ser fatal para su entrega a tiempo.
- **Desarrolladores meticulosos**.- Los desarrolladores encuentran fascinante la nueva tecnología, y a veces están ansiosos por probar nuevas funcionalidades de su lenguaje o entorno, o por crear su propia implementación de una utilidad bonita que han visto en otro producto, la necesite o no su producto. El esfuerzo requerido para diseñar, implementar, probar, documentar o mantener estas funcionalidades innecesarias alarga el plan.
- **Estire y afloje en la negociación.**- Se presenta cuando un directivo aprueba un retraso en el plan de un proyecto que progresa más lento de lo esperado, pero entonces añade tareas completamente nuevas después de un cambio en el plan. Esto es curioso, puesto que si el directivo aprueba el retraso en el plan lo hace sabiendo implícitamente que el plan estaba equivocado, no obstante, una vez

que se corrige, la misma persona realiza acciones explícitas para volver a equivocarse. Esto sólo puede ir en contra del plan.

- **Desarrollo orientado a la investigación**.- Si el proyecto fuerza los límites de la informática porque necesita la creación de nuevos algoritmos o de nuevas técnicas de computación, no estamos desarrollando software; estamos investigando en software. Los planes de desarrollo de software son razonablemente predecibles; los planes en la investigación sobre software ni siquiera son predecibles teóricamente. Cuando el producto tiene objetivos que pretenden aumentar los conocimientos existentes, como algoritmos, velocidad, utilización de la memoria y demás, debemos asumir que la planificación es altamente especulativa.

[McConnell, 1997] hace una analogía de lo que sucede frecuentemente en la administración de proyectos, con una antigua serie de televisión llamada *La isla de Gilligan*: "Al principio de cada episodio, Gilligan, el Capitán o el Profesor llegaban con un plan tonto para salir de la isla. El plan parecía funcionar inicialmente, pero, como revelaba el episodio, algo iba mal, y al final del episodio los náufragos volvían donde habían empezado, perdidos en la isla. De igual forma, la mayoría de las compañías descubre al final de cada proyecto que han cometido otro error clásico y que han entregado otro proyecto fuera de plazo, con mayor costo, o ambas cosas".

Es recomendable elaborar una lista propia de *malos hábitos* para procurar no caer en ellos la próxima vez, se puede comenzar con la los errores señalados por [McConnell, 1997] incluidos en este documento.

Capítulo VI Administración basada en riesgos

VI.1 El proceso de la gestión de riesgos

El desarrollo de proyectos complejos de software lleva implícita la existencia de ciertos riesgos que si no se toman en cuenta y se analizan con cuidado, retrasarán considerablemente la entrega del producto o incluso podrían llegar a causar la cancelación del proyecto.

Un riesgo se puede definir de manera sencilla como:

- "la probabilidad de que una circunstancia adversa ocurra" [Sommerville, 2006]
- "un problema potencial que puede ocurrir o no" [Pressman, 2006]

Definición de administración basada en riesgos.-"*La administración o gestión de riesgos consiste en identificar éstos y crear planes para minimizar sus efectos en el proyecto*." [Sommerville, 2006].

La gestión de riesgos consiste en identificar los riesgos, evaluar la probabilidad de que ocurran, estimar su impacto y establecer un plan de contingencia en caso de que el problema se presente [Pressman, 2006].

La gestión de los riesgos puede clasificarse en varios niveles, según [Presmman, 1993]:

Nivel 1: *Control de crisis*.- Esta etapa es equivalente a "apagar el fuego", es decir, controlar los riesgos sólo cuando se ha convertido en problemas.

Nivel 2: *Arreglar cada error*.- Detectar y reaccionar rápidamente ante cualquier riesgo, pero sólo después de que se haya producido.

Nivel 3: *Mitigación del riesgo*.- Planificar con anticipación el tiempo que se necesitaría para cubrir riesgos en el caso de que estos ocurrieran, sin intentar eliminarlos inicialmente.

Nivel 4: *Prevención*.- Crear y llevar a cabo un plan como parte del proyecto de 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 ocasionar algún riesgo.

Para que se logre una reducción, supervisión, y administración del riesgo de manera correcta y eficiente es necesario trabajar en los niveles *4* y *5*, ya que trabajar con los niveles *1*,*2* y *3* implica una mala administración del proyecto.

Figura 6.1: Todavía existen organizaciones que concentran sus esfuerzos en arreglar sus errores en lugar de prevenirlos (Briseño, 2003).

Como se aprecia en la *figura 6.1* Todavía existen organizaciones que concentran sus esfuerzos en arreglar sus errores en lugar de prevenirlos. "La función de la gestión de riesgos del software es identificar, estudiar y eliminar las fuentes de riesgo antes de que empiecen a amenazar la finalización satisfactoria de un proyecto de software"[McConnell, 1997].

Existen métodos que ayudan a identificar en que momento alguno de los factores que intervienen en el desarrollo de un proyecto podría causar problemas críticos, con el apoyo de estos métodos el gerente o administrador del proyecto puede planear tiempos de holgura y algunas actividades que ayuden a minimizar los problemas, el uso de estos métodos es básico en la gestión de riesgos del software.

La *figura 6.2* muestra cuales son los elementos del proceso de gestión de riesgos según [McConnell, 1997]:

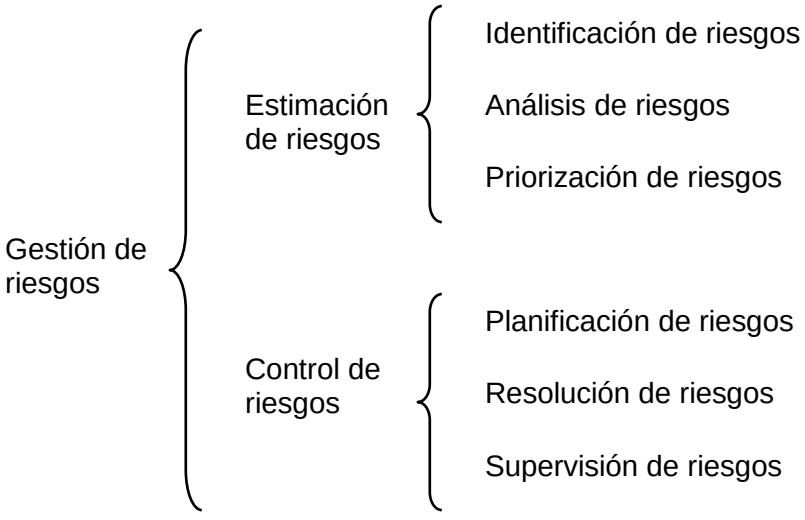


Figura 6.2: La gestión de riesgos se compone de estimación y control de riesgos (McConnell, 1997; Sommerville, 2006; Bohehm, 1989)

Estimación de riesgos.

La *identificación de riesgos* consiste en reconocer cuales son los posibles riesgos para el proyecto, el producto y los negocios. Una práctica recomendada es solicitar a los responsables del proyecto y a los líderes técnicos que hagan su propia lista de riesgos.

El *análisis de riesgos* consiste en valorar las probabilidades y consecuencias de los riesgos identificados. Las listas elaboradas por cada uno de los responsables se discuten en grupo para analizar cada uno de éstos.

Con la *priorización de riesgos* se unifican criterios y se obtiene una lista con los riesgos más importantes ordenados en función de la probabilidad de que ocurran y de la gravedad de sus consecuencias.

Control de riesgos.

La *planificación de riesgos* consiste en crear planes para abordar los riesgos, ya sea para evitarlos o minimizar sus efectos en el proyecto.

La *resolución de riesgos* es la ejecución del plan para resolver cada uno de los riesgos significativos.

La *supervisión de riesgos* consiste en valorar los riesgos de forma constante y revisar los planes para su mitigación tan pronto como su información esté disponible. Esta actividad incluye la identificación de nuevos riesgos para volver a considerarlos en el proceso de su gestión.

El proceso de gestión de riesgos puede visualizarse como se indica en la *figura 6.3*.

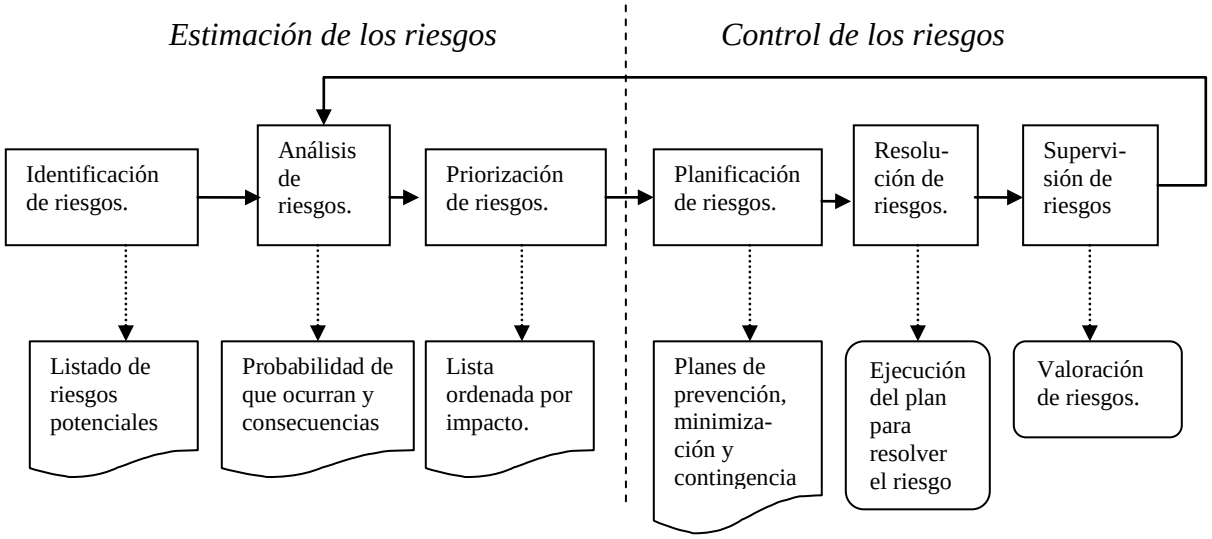


Figura 6.3: El proceso de gestión de riesgos (Sommerville, 2006).

VI.2 Estimación de los riesgos

VI.2.1 Identificación de riesgos

Una de las tareas más importantes del administrador del proyecto es anticipar los riesgos que podrían afectar a la programación del proyecto o a la calidad del software a desarrollar y emprender acciones para evitar esos riesgos.

La identificación de los riesgos es un intento sistemático encaminado a especificar las amenazas del proyecto. Cuando se identifican los riesgos conocidos y predecibles hay oportunidad para evitarlos cuando es posible y controlarlos cuando es necesario.

Estimar los riesgos significa valorar cuales son los riesgos que podrían presentarse durante el desarrollo del proyecto, analizándolos y distinguiendo cuales serían los más graves, sin minimizar el impacto que pudieran tener los menos graves. A continuación estudiaremos con más detalle en que consiste cada una de las actividades que componen la estimación de los riesgos.

Cuando se analizan los riesgos es importante cuantificar el grado de incertidumbre y el grado de pérdida asociado a cada riesgo. [Pressman, 2006] distingue tres categorías de riesgos:

- *Riesgos del proyecto*.- Amenazan al plan del proyecto, si se vuelven reales es probable que se altere la calendarización y que aumenten los costos. Estos riesgos son principalmente: problemas potenciales de presupuesto, calendarización, personal, recursos, participantes y requerimientos.

- *Riesgos técnicos*.- Ocurren porque el problema es más difícil de resolver de lo que en un principio se pensó que sería. Amenazan la calidad y actualidad del software que se producirá. Si un riesgo técnico se vuelve real, la implementación se vuelve difícil o imposible. Estos riesgos son principalmente: problemas potenciales de diseño, implementación, interfaz, verificación y mantenimiento. Además también son factores de riesgo técnico: la ambigüedad de la especificación, incertidumbre técnica, la obsolescencia técnica y la tecnología "de punta".

- *Los riesgos de negocios*.- Amenazan la viabilidad del software que se construirá. Estos riesgos ponen en peligro el proyecto o el producto, los cinco riesgos de negocio más importantes son:
  1. Construcción de un excelente sistema que en realidad nadie quiere. (riesgo de mercado).
  2. Construcción de un producto que ya no encaja en la estrategia comercial global de la compañía (riesgo de estrategia).
  3. Construcción de un producto que no se sabe como vender (riesgo de ventas).

4. Pérdida del apoyo de los altos ejecutivos debido a un cambio en el enfoque o en el personal (riesgo administrativo).
5. Pérdida del presupuesto o del personal asignado (riesgo presupuestal).

A continuación se muestran varios puntos con los que los clientes pueden crear riesgos en la planificación según [McConnell, 1997]:

• Los clientes no saben lo que quieren.
• Los clientes no quieren comprometerse a tener un conjunto de requerimientos escritos.
• Los clientes insisten en establecer nuevos requerimientos una vez que se han fijado la planificación y el costo.
• La comunicación con los clientes es lenta.
• Los clientes no participan en las revisiones o son incapaces de hacerlas.
• Los clientes no están preparados técnicamente.
• Los clientes no dejan realizar el trabajo a la gente.
• Los clientes no entienden el proceso de desarrollo de software.
• Un cliente nuevo es una entidad desconocida, y se desconocen los riesgos específicos.

Establecer buenas relaciones con los clientes permite identificar mejor los riesgos y controlarlos durante el desarrollo del proyecto.

Un método para identificar riesgos consiste en crear una lista de verificación. Con ésta se identifican los riesgos, luego se enfoca la atención sobre un subconjunto de los conocidos y predecibles y se clasifican según las siguientes sub categorías según [Pressman, 2006]:

- *Tamaño del producto*.- riesgos asociados con el tamaño global del software que se construirá o modificará.
- *Impacto en el negocio*.- riesgos asociados con las restricciones que impone la gerencia o el mercado.
- *Características del cliente*.- riesgos asociados con la sofisticación del cliente y la habilidad del desarrollador para comunicarse con él en una forma oportuna.
- *Definición del proceso*.- riesgos asociados con el grado en el que se ha definido el proceso de software y en que le da seguimiento la organización que lo desarrolla.
- *Entorno de desarrollo*.- riesgos asociados con la disponibilidad y calidad de las herramientas que se usarán en la construcción del producto.
- *Tecnología que construir*.- riesgos asociados con la complejidad del sistema que se construirá y la "novedad" de la tecnología que está empaquetada en el sistema.
- *Tamaño y experiencia de la plantilla de personal*.- riesgos asociados con la experiencia técnica, experiencia en el proyecto y con los ingenieros de software que harán el trabajo.

VI.2.2 Análisis y priorización de riesgos

Durante el análisis se considera cada riesgo identificado para evaluar la probabilidad de que éste ocurra y su gravedad. Estas consideraciones se hacen en base al criterio y experiencia del administrador del proyecto. [Sommerville, 2006] recomienda los valores que se presentan en la tabla de la *figura 6.4*.

Probabilidad de que ocurra	Nivel
< 10%	Muy bajo
10-25%	Bajo
25-50%	Moderado
50-75%	Alto
< 75%	Muy alto

Figura 6.4: Niveles de probabilidad de que el riesgo ocurra.

Además de evaluar la probabilidad de que ocurra el riesgo es necesario evaluar sus consecuencias, [Sommerville, 2006] distingue 4 niveles de gravedad de los riesgos, que son:

1. Catastrófico.
2. Serio.
3. Tolerable.
4. Insignificante.

La idea fundamental de la administración de proyectos de software basada en riesgos cuando se usan modelos de desarrollo iterativos, tales como el modelo en espiral o el Proceso Unificado, es planear las iteraciones conforme a la prioridad de los riesgos. De esta manera, los riesgos de mayor prioridad se hacen frente en las primeras iteraciones, mientras que los de menor prioridad se dejan para después.

Una vez identificados y analizados los riesgos, se elabora una lista ordenada según su grado de importancia. El número exacto de riesgos a supervisar depende del proyecto. [Bohem, 1988] recomienda una lista de 10, sin embargo la lista podría ser de 15 o de 5, la idea principal es que el número elegido de riesgos sea una cantidad manejable.

Algunos de los riesgos que hay que tomar siempre en cuenta se ilustran en la *figura 6.5*:

Riesgo	Probabilidad	Efecto
Problemas de reclutamiento.- Imposible reclutar el personal con las habilidades requeridas para el proyecto.	Alta	Catastrófico
Problemas financieros de la organización.- Fuerzan a reducir el presupuesto del proyecto.	Baja	Catastrófico
Reestructuración organizacional.- La organización se reestructura de tal forma que cambia el grupo de gestión del proyecto.	Alta	Serio
Tiempo de desarrollo subestimado.- El tiempo requerido para desarrollar el software esta subestimado.	Alta	Serio
Rendimiento de la base de datos.- La base de datos que se utiliza en el sistema no puede procesar muchas transacciones por segundo como se esperaba.	Moderada	Serio
Enfermedad del personal.- El personal clave está enfermo y no disponible en momentos críticos.	Moderada	Serio
Cambios en los requerimientos.- Éstos provocan que se tenga que rehacer el diseño.	Moderada	Serio

Figura 6.5: Ejemplo del análisis de riesgos [Sommerville, 2006]

VI.3 Control de riesgo

Una vez que ya se han valorado los riesgos, se pasa a la etapa de *control de riesgo*, la cual se compone de las siguientes tres actividades: planificación, resolución y supervisión de riesgos.

VI.3.1 Planificación de riesgos

El objetivo de la planificación de riesgos es desarrollar un plan que supervise cada uno de los riesgos enlistados proponiendo una estrategia para enfrentarlos en caso de que ocurran.

En la *figura 6.6* se muestra un ejemplo de un plan para controlar los riesgos.

Riesgo	Estrategia
Problemas de reclutamiento.	Alertar al cliente de las dificultades potenciales y los posibles retrasos. Investigar la compra de componentes ya hechos.
Problemas financieros de la organización.	Preparar un breve documento para el gestor principal que muestre que el proyecto hace contribuciones muy importantes a las metas del negocio.
Reestructuración organizacional.	Preparar un breve documento para el nuevo gestor que muestre las contribuciones importantes del proyecto a las metas del negocio.
Tiempo de desarrollo subestimado.	Incluir en el plan del proyecto algunas semanas de margen para ”retrasos inesperados” y aumentar una o dos semanas más para los *inesperados* "retrasos inesperados”.
Rendimiento de la base de datos	Investigar la posibilidad de comprar una base de datos de alto rendimiento.
Enfermedad del personal.	Reorganizar al equipo de tal forma que haya solapamiento en el trabajo y que los integrantes comprendan lo que están haciendo los demás.
Cambios en los requerimientos	Analizar con mucho cuidado los requerimientos (invertir el tiempo necesario) para que no haya información oculta que impida visualizar cambios radicales en el diseño.

Figura 6.6: Ejemplo de estrategia de gestión de riesgos.

No existe un plan que permita establecer los planes de gestión de riesgos, el plan depende del juicio y la experiencia del administrador del proyecto. [Sommerville, 2006] distingue tres categorías en las estrategias para enfrentar los riesgos.

1. *Estrategias de prevención*.- Siguiendo estas estrategias, la probabilidad de que el riesgo aparezca se reduce.
2. *Estrategias de minimización*.- Siguiendo estas estrategias, se reducirá el impacto del riesgo. Un ejemplo de este tipo es la estrategia que se presenta en la *figura 6.6* para enfrentar una eventual enfermedad del personal.
3. *Planes de contingencia*.- Seguir estas estrategias es estar preparado para lo peor y tener una estrategia para cada caso. Un ejemplo de este tipo de estrategia es la que se presenta en la *figura 6.6* para afrontar los problemas financieros.

VI.3.2 Resolución de riesgos

La resolución de un riesgo concreto depende del riesgo en cuestión, [McConnell, 1997] propone las siguientes estrategias a seguir:

- *Trasladar el riesgo de una parte del sistema a otra*.- A veces lo que es un riesgo en una parte del proyecto no lo es en otra parte del mismo. Por ejemplo, se puede trasladar la revisión de una parte del diseño con la que un desarrollador no está familiarizado de tal manera que la revise personal que ya tiene experiencia al respecto.
- *Eliminar el origen del riesgo*.- En lo posible, evitar actividades que pongan en riesgo el proceso. Por ejemplo, si el diseño de una parte del sistema es demasiado arriesgado, cambiarla a un proyecto de investigación y eliminarla de la versión que se está desarrollando.
- *Comunicar el riesgo*.- Hacer saber al personal involucrado: la dirección, marketing, clientes, la presencia del riesgo y sus consecuencias.
- *Controlar el riesgo*.- Aceptar que el riesgo puede ocurrir y desarrollar planes para afrontarlo, por ejemplo, si hay riesgo de mudarse de oficina durante el desarrollo del proyecto, puede plantearse como estrategia contratar personal temporal para empacar y desempacar y realizar la mudanza en fin de semana.
- *Recordar el riesgo*.- Crear una colección de planes de gestión de riesgos que se pueda utilizar en proyectos futuros.

VI.3.3 Supervisión de riesgos

El proceso de gestión de riesgos, como otros de planificación de proyectos, es un proceso iterativo que se aplica a lo largo de todo el proyecto. Una vez que se genera un conjunto de planes iniciales, se supervisa la situación. En cuanto surja más información acerca de los riesgos, éstos deben analizarse nuevamente y se deben establecer nuevas prioridades. La prevención de riesgos y los planes de contingencia se deben modificar tan pronto como surja nueva información de los riesgos.

Los resultados del proceso se deben documentar en un plan de gestión de riesgos. Éste debe incluir un estudio de los riesgos a los que se enfrenta el proyecto y un análisis de éstos. Si es necesario, puede incluirse algunos planes específicos de contingencia que se activan al aparecer dichos riesgos.

Capítulo VII Herramientas para aumentar la productividad

VII.1 Las herramientas para el soporte de la productividad

Las herramientas para aumentar la productividad son un factor importante en la administración de proyectos, su objetivo principal es mejorar la productividad en el desarrollo y mantenimiento del software, sin embargo el poseer la herramienta no garantiza que se obtendrán resultados de manera inmediata o a largo plazo. El criterio para usarlas y la creatividad también son factores decisivos.

Hay una gran variedad de herramientas para la administración de proyectos. Estas herramientas pueden clasificarse en dos categorías:

Orientadas a las personas y al proceso – Tienen el fin de mejorar la planeación y la administración de los recursos de los proyectos. Las personas son los creadores de software y tienen un efecto importante sobre la producción. Existen herramientas que permiten tomar decisiones inteligentes basadas en los indicadores tales como los factores que obstaculizan el proyecto, por ejemplo, la disponibilidad de recursos.

La forma en cómo trabaja la gente tiene un impacto significativo sobre el resultado final. Existe una amplia gama de herramientas tanto de paga como software libre que sirven para definir tareas, duración y personal involucrado, fechas de entrega, gestión de recursos y cálculo de costos. Desde el punto de vista del administrador de proyectos, estas herramientas sirven para planear y monitorear los proyectos, y desde el punto de vista de los miembros del equipo de desarrollo las herramientas sirven para visualizar sus tareas y registrar sus avances.

Las herramientas que ayudan a monitorear el proceso facilitan la detección de problemas y permiten poner en práctica las estrategias para resolverlos. El uso de este tipo de herramientas ayuda a los administradores a visualizar cuan exitosa es la empresa tanto a nivel del portafolios de proyectos como a nivel de la planificación detallada. Además permiten visualizar fácilmente los datos por medio de informes, tablas y gráficas.

Las herramientas de comunicación facilitan el trabajo conjunto del administrador con los miembros de su equipo para una administración más eficiente de documentos y de proyectos. Permiten al administrador compartir documentos, notificar acerca de cambios de estatus o invitar a los miembros del equipo a contribuir con sus sugerencias para el proyecto.

Orientadas al desarrollo – Su principal objetivo es facilitar el uso de las distintas metodologías propias de la ingeniería del software. Las herramientas que facilitan el diseño,

la implementación automática de parte del código con un diseño dado, la compilación automática y la documentación o detección de errores pueden acelerar el proceso de desarrollo de un sistema de software.

VII.2 Principales tipos de herramientas que existen en el mercado.

El objetivo general de las herramientas es acelerar el proceso para el que han sido diseñadas, es decir, para automatizar o apoyar una o más fases del ciclo de vida del desarrollo de sistemas. Hay herramientas semiautomatizadas y automatizadas diseñadas para un uso específico, éstas se enlistan a continuación [Barzanallana, 2010].

VII.2.1 Herramientas orientadas a las personas y al proceso.

- *Modelado de procesos y herramientas de administración***.-** Se utilizan para representar los elementos clave del proceso de modo que sea posible entenderlo mejor.

- *Herramientas de planificación de proyectos*.- Sirven para calcular el esfuerzo estimado en número de personas y tiempo requerido y para definir las tareas.

- *Herramientas de administración de proyectos*.- Da seguimiento al plan del proyecto.

- *Herramientas para el análisis de riesgos*.- Guían al administrador en la construcción tablas de riesgos para mejorar su identificación y análisis.

- *Herramientas de seguimiento de requerimientos*.- Cuando se desarrollan sistemas grandes, el producto final suele no satisfacer los requisitos especificados por el cliente. El objetivo de este tipo de herramientas es proporcionar un enfoque sistemático para el aislamiento de requisitos, comenzando por las especificaciones del cliente.

- *Herramientas de métricas y gestión*.- Mejoran la capacidad del administrador para controlar y coordinar el proceso del software y la capacidad del ingeniero para dar calidad al software que produce. Las herramientas orientadas a la gestión capturan métricas especificas del proyecto (por ejemplo: LDC/persona-mes, defectos por punto de función) que proporcionan una indicación global de productividad o de calidad.

- *Herramientas de administración de documentación*.- Dado que la mayor parte de las organizaciones dedicadas al desarrollo de software invierte una cantidad de tiempo considerable en el desarrollo de documentos, y en muchos casos el proceso de documentación en si resulta bastante deficiente, este tipo de herramientas son de gran ayuda.

- *Herramientas de control de calidad*.- Son herramientas métricas que hacen una auditoria del código fuente para determinar si se ajusta o no a ciertos estándares del lenguaje. Otras herramientas extraen métricas técnicas como base para medir la calidad del software que se esta construyendo.

VII.2.2 Herramientas orientadas al desarrollo

- *Herramientas de codificación de cuarta generación*.- Los lenguajes de cuarta generación, los generadores de código y los generadores de aplicaciones, permiten que un ingeniero de software especifique un sistema a un nivel muy alto de abstracción.

- *Herramientas de mantenimiento*.- Las *herramientas de ingeniería inversa a especificaciones*, toman el código fuente como entrada y generan modelos de diseño y análisis estructurado, listas de utilización y otra información con el diseño. Las *Herramientas de reestructuración y análisis de código*, analizan la sintaxis del programa, generan un grafo de flujo de control y un programa estructurado. Las *Herramientas interactivas de reingeniería de sistema*, se utilizan para modificar sistemas de base de datos.

- *Herramientas de gestión de configuración de software*.- Son de gran ayuda para el control de versiones y el control de cambios. Proporcionan un mecanismo para identificar todos los elementos de configuración y relacionarlos con otros elementos. Los elementos de configuración individuales facilitan el proceso de auditoria; otra función es la contabilidad de estados, la cual brinda la información de los cambios a todos aquellos que necesiten conocerlos.

- *Herramientas de análisis y diseño*.- Capacitan al ingeniero del software para crear modelos del sistema que haya que construir. Los modelos contienen una representación de los datos, de la función y del comportamiento (en el nivel de análisis), así como caracterizaciones del diseño de datos, arquitectura, procedimientos e interfaz.

- *Herramientas de prototipos y simulación*.- proporcionan al ingeniero del software la capacidad de predecir el comportamiento de un sistema en tiempo real antes de llegar a construirlo. Además, capacitan al ingeniero del software para desarrollar simulaciones del sistema de tiempo real que permitirán al cliente obtener ideas acerca de su funcionamiento, comportamiento y respuesta antes de la verdadera implementación

- *Herramientas de desarrollo y diseño de interfaz*.- Son un conjunto de primitivas de componente de programas tales como menús, botones, estructuras de ventanas, iconos, mecanismos de desplazamiento, controladores de dispositivos, etc. que permiten construir pantallas de interfaces de usuario.

- *Herramientas de generación de prototipos*.- Permiten al ingeniero de software definir rápidamente la disposición de pantalla para aplicaciones interactivas y crear un diseño de datos, acoplado con las disposiciones de la pantalla y de los informes.

- *Herramientas de programación*.- Son los compiladores, editores y depuradores que están disponibles para prestar su apoyo en la mayoría de los lenguajes de programación convencionales.
- *Herramientas de integración y comprobación*.- Adquieren datos que se utilizaran durante la comprobación. Analizan el código fuente sin ejecutar casos de prueba. Analizan el código fuente durante la ejecución. Simulan las funciones del hardware o de otros elementos externos y prestan su asistencia en la planificación, desarrollo y control de las comprobaciones.

- *Herramientas de reingeniería* .- Las *herramientas de ingeniería inversa para producir especificaciones* toman el código fuente como entrada y generan modelos gráficos de análisis y diseño estructurados, listas de utilización y otras informaciones de diseño. Las *herramientas de reestructuración y análisis de código* analizan la sintaxis del programa y generan una gráfica de control de flujo y un programa estructurado. Las *herramientas de reingeniería para sistemas en línea* se utilizan para modificar sistemas de bases de datos en línea.

VII.3 Adquisición y uso de herramientas para el soporte de la productividad

Las herramientas para mejorar la productividad se pueden comprar, sin embargo también existen herramientas de software libre, las cuales están más limitadas. A continuación mencionaremos algunas de éstas.

VII.3.1 OpenSched

OpenSched es una herramienta gratuita para la gestión de proyectos, toma como entrada un archivo que describe el proyecto y genera descripciones del plan del proyecto, diagramas de Gantt y diagramas de red. Utiliza PDFLatex para mostrar los archivos de salida y programa automáticamente los recursos para un proyecto. Sin embargo, solo se puede utilizar en plataforma Linux, aún tiene defectos y su uso es complicado, ya que es por medio de línea de comandos (http://mtechit.com/download/sched/).

VII.3.2 Planner

Planner es también una herramienta libre escrita en lenguaje de programación C para gestión de proyectos, diseñada para el escritorio de GNOME, sus funcionalidades son:

- Gestión de calendarios.
- Gestión de recursos.
- Seguimiento del avance del proyecto.
- Enlazar tareas.
- Exportación a diferentes formatos.

Planner puede almacenar sus datos en archivos XML o en una base de datos PostgreSQL. Los proyectos se pueden imprimir en PDF o exportar a HTML para una visualización simple desde cualquier navegador web. Tiene capacidad para importar proyectos desde Microsoft Project (solo formato XML). Su interfaz con el usuario es más amigable que la de *OpenSched*, existen versiones en distintos idiomas, es multiplataforma, tiene licencia GPL y es de fácil instalación. Sin embargo, solo es factible para empresas pequeñas y aún tiene defectos. (http://www.planner-software.com/planner/, http://www.alcancelibre.org/article.php/conoce-planner).

VII.3.3 GanttProject

GanttProject es un software gratuito para la administración de proyectos. Nos permite planificar todas las tareas y actividades de un proyecto en el tiempo acordado, facilitando una visualización del estado de progreso de cada actividad, tiene la capacidad de dividir los proyectos en subtareas, cada una con sus fechas de inicio y previsión de finalización. Cada actividad puede ser enlazada a todas las demás actividades, con sus respectivas relaciones de dependencia, recursos adyacentes, comentarios, etc. (http://www.ganttproject.biz/). Su interfaz con el usuario es muy sencilla y en general, sus funcionalidades son mejores que las de planner. A continuación se muestra un ejemplo del formato del reporte producido con *GanttProject*.

Informe GanttProject

Proyecto : presentacion

Inicio : 1/02/11

Fin : 23/02/11

Organización : uam-c

Página web :

Descripción :

ANALISIS DE HERRAMIENTA PARA ADMINISTRACION DE PROYECTOS

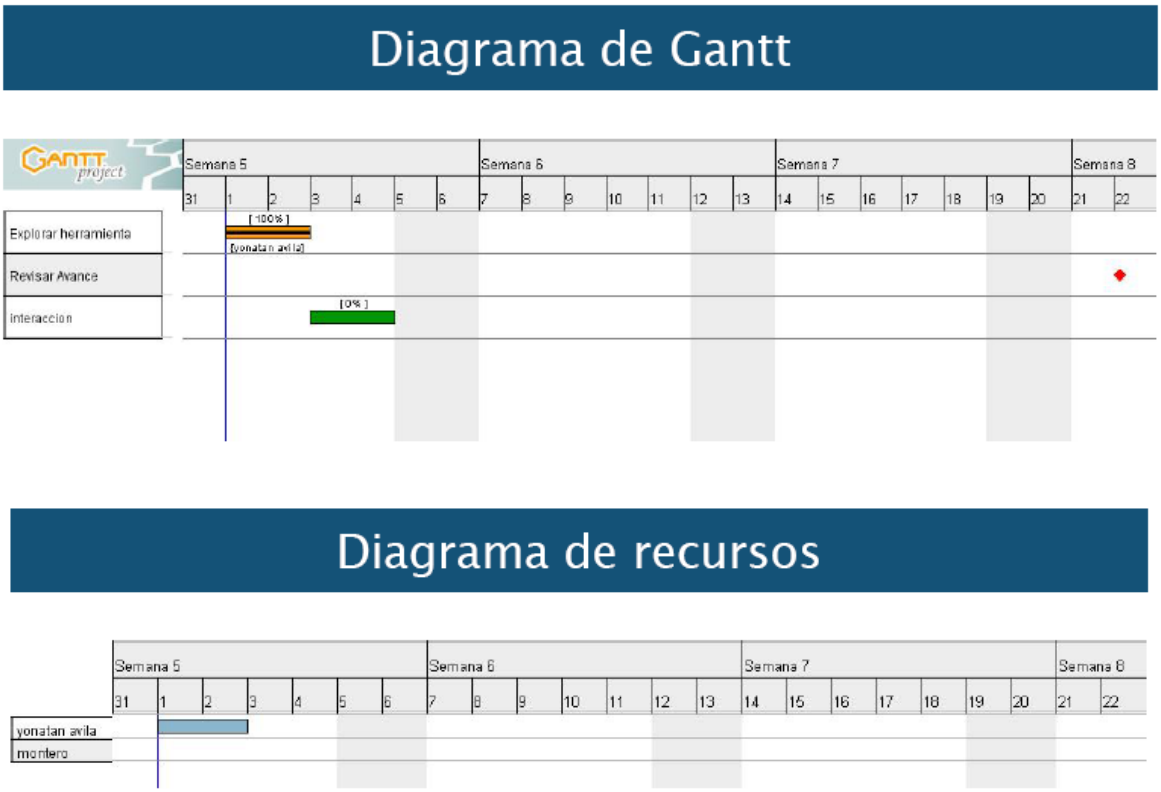
Date : 08-mar-2011 11:04:16

Lista de tareas

Nombre	Inicio	Fin	Hito	%	Recursos	Notas
Explorar herramienta	1/02/11	3/02/11	false	100	yonatan avila	la herramienta es nueva para el equipo
Revisar Avance	22/02/11	23/02/11	true	100		
interaccion	3/02/11	5/02/11	false	0		

Lista de recursos

Nombre	Función	E-Mail	Teléfono
Name:yonatan avila	Encargado del proyecto	yonqheas@hotmaiil.com	56789034
Name:montero	Indefinido	correo@hotkeys.com	56789043



VII.3.4 Microsoft Project

Microsoft Project Es una herramienta que requiere de la compra de su licencia. *Microsoft Project* permite planear sistemáticamente las fases y tareas de un proyecto. Sus principales funcionalidades son:

- Organizar la lista de actividades en una estructura jerárquica.
- Asignar recursos y costos a las diferentes actividades.
- Obtener la gráfica de red del proyecto.
- Imprimir una gran variedad de informes.

Para comenzar un nuevo proyecto en *Microsoft Project* es necesario contar con la siguiente información previa:

1. Título del proyecto.
2. Calendario: Días laborables y no laborables en el proyecto (Receso, sábados y domingos).
3. Lista de tareas principales.
4. Dividir cada tarea en subtareas (Pasos para ejecutar la Tarea principal)

5. Determinar la duración de cada tarea y subtarea.

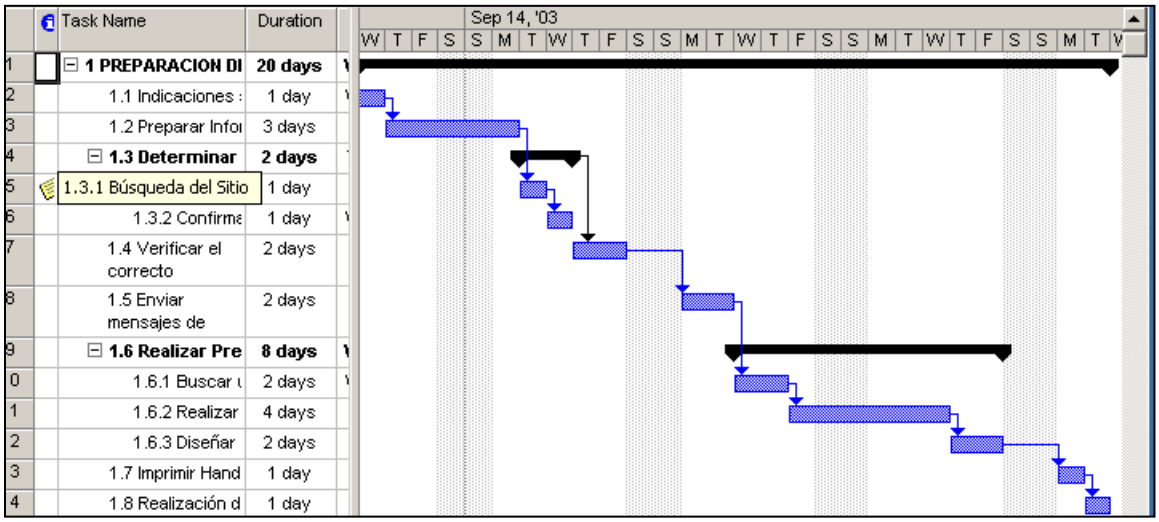
A continuación se presenta un ejemplo de tareas y subtareas para el proyecto "Organización de un seminario."

- Indicaciones sobre el seminario.
- Preparar Información Básica.
- Determinar fecha, hora y lugar. ← Tarea principal
    - Búsqueda de la sede.
    - Confirmación de la sede. } ← Tareas secundarias
- Verificar el correcto funcionamiento del programa en el sitio.
- Enviar mensajes de Invitación al personal administrativo.
- Realizar Presentación en Power Point
    - Diseñar ejemplos y ejercicios.
    - Hacer un bosquejo de la presentación.
    - Diseñar la presentación.
- Imprimir Manual y sacar copias.
- Realización del seminario.

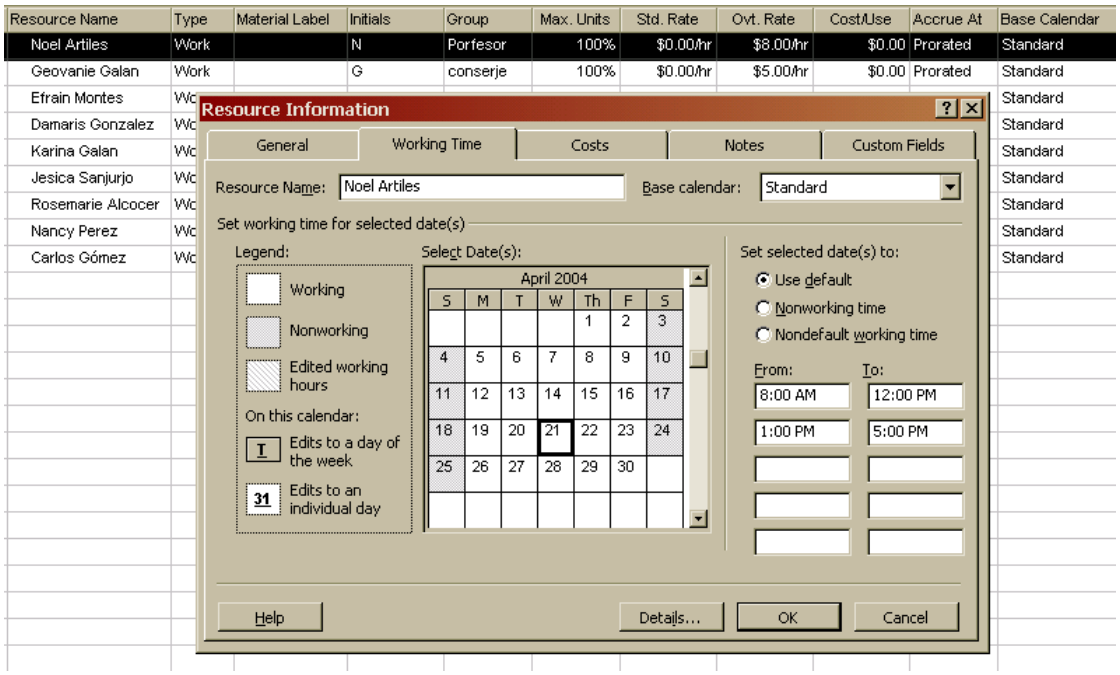
Con *Microsoft Project* se pueden jerarquizar las tareas y definir tiempos con una interfaz de usuario muy similar a la de Word y Excel, a continuación se muestra un ejemplo de la hoja de tareas, en donde se especifican las tareas, las subtareas, fecha de inicio y finalización. La precedencia sirve para indicar que tarea es necesario finalizar primero para poder comenzar con otra.

Task Name	Duration	Start	Finish	Prede
1 PREPARACION DE SEMINARIO	**20 days**	**Wed 9/10/03**	**Tue 10/7/03**	
1.1 Indicaciones sobre el seminario	1 day	Wed 9/10/03	Wed 9/10/03	
1.2 Preparar Información Básica	3 days	Thu 9/11/03	Mon 9/15/03	2
1.3 Determinar Fecha, Hora y Lugar	**2 days**	**Tue 9/16/03**	**Wed 9/17/03**	
1.3.1 Búsqueda del Sitio	1 day	Tue 9/16/03	Tue 9/16/03	3
1.3.2 Confirmación del sitio	1 day	Wed 9/17/03	Wed 9/17/03	5
1.4 Verificar el correcto funcionamiento del programa en el sitio	2 days	Thu 9/18/03	Fri 9/19/03	4
1.5 Enviar mensajes de Invitación al personal administrativo.	2 days	Mon 9/22/03	Tue 9/23/03	7
1.6 Realizar Presentación en Power Point	**8 days**	**Wed 9/24/03**	**Fri 10/3/03**	
1.6.1 Buscar un ejemplo de aplicacion	2 days	Wed 9/24/03	Thu 9/25/03	8
1.6.2 Realizar la presentacion en borrador	4 days	Fri 9/26/03	Wed 10/1/03	10
1.6.3 Diseñar la presentacion	2 days	Thu 10/2/03	Fri 10/3/03	11
1.7 Imprimir Handouts y Fotocopias	1 day	Mon 10/6/03	Mon 10/6/03	12
1.8 Realización del seminario	1 day	Tue 10/7/03	Tue 10/7/03	13

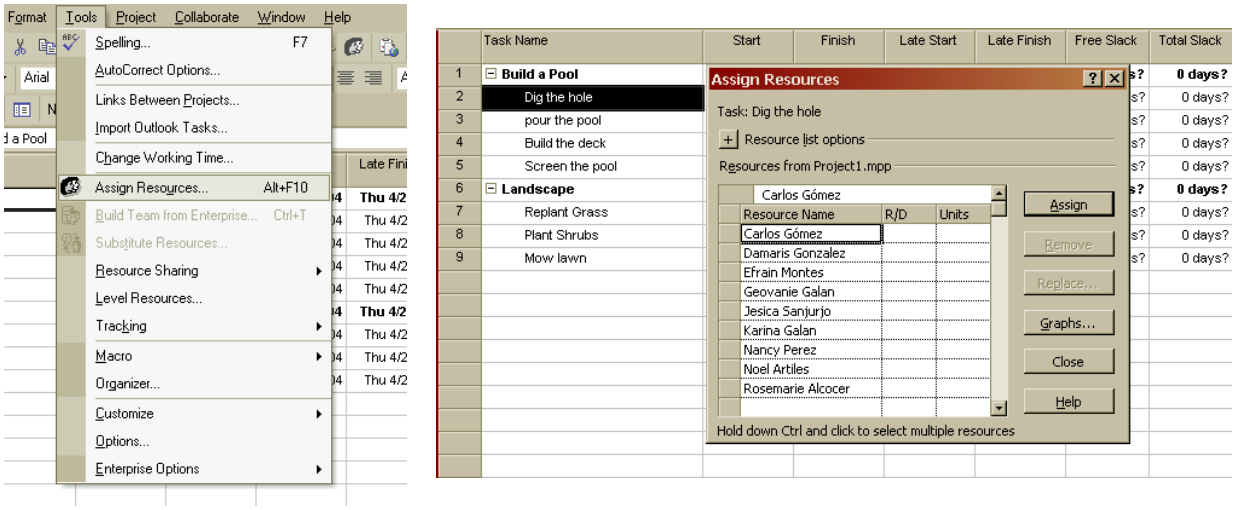
Esta herramienta elabora el diagrama de Gantt directamente de la información de la hoja de tareas.



Microsoft Project también cuenta con una hoja para administrar los recursos materiales y humanos, la hoja de recursos permite introducir información detallada de cada recurso, a continuación se muestra un ejemplo:



La asignación de tareas a los recursos humanos se hace de una manera sencilla, como se muestra en las siguientes figuras:



Con *Microsoft Project,* se pueden crear informes de varios tipos con una muy buena presentación.

En Internet hay una amplia información acerca de la manera de trabajar con *Microsoft Project*, por ejemplo:

http://www.ese.upenn.edu/seniordesign/resources/MS_Project_Tutorial.pdf

http://www.tutorial-lab.com/tutoriales-microsoft-project/id351-gestion-proyectos-con-microsoft-project.aspx

http://www.youtube.com/watch?v=2O_Dr88KbIs

http://www.programatium.com/microsoft-project.htm

VII.3.5 Métrica.

La metodología Métrica Versión 3 proporciona un conjunto de métodos y técnicas que sirven de guía a los distintos profesionales de Sistemas y Tecnologías de la Información y Comunicaciones en la obtención de los diversos productos de los procesos del ciclo de vida de un proyecto de software, su portal se encuentra en: http://administracionelectronica.gob.es/?_nfpb=true&_pageLabel=P60085901274201580632&langPae=es . EL objetivo principal de esta metodología es mejorar la productividad de los participantes y asegurar la calidad. La mayoría de las técnicas propuestas están soportadas por herramientas disponibles en el mercado que automatizan en mayor o menor grado su utilización. Sin embargo, no todos los productos resultantes de cada tarea son

susceptibles de obtenerse de forma automatizada. Métrica 3, divide las actividades de Gestión de Proyectos en tres tipos:

- Actividades de Inicio del Proyecto (GPI), que permiten estimar el esfuerzo y establecer la planificación del proyecto.
- Actividades de Seguimiento y Control (GPS), supervisando la realización de las tareas por parte del equipo de proyecto y gestionando las incidencias y cambios en los requisitos que puedan presentarse y afectar a la planificación del proyecto.
- Actividades de Finalización del Proyecto, cierre y registro de la documentación de gestión

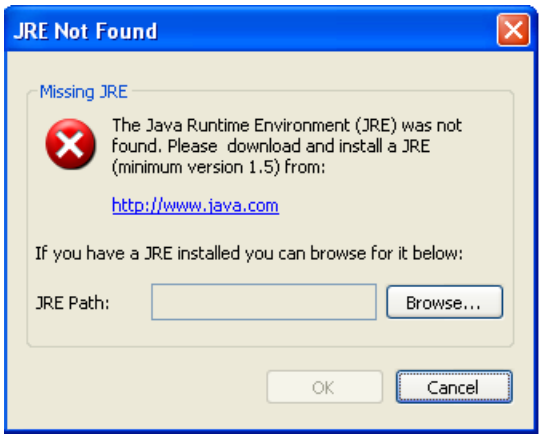
Estas actividades pueden requerir, en función de la complejidad del proyecto, el soporte de herramientas comerciales de gestión de proyectos.

VII.3.6 OpenProj.

OpenProj es la mejor alternativa a MSProject en versión libre.

Requerimientos de hardware y software.
OpenProj requiere de una versión 1.5 de java como mínimo; aunque se prefiere la versión 1.6. Podemos observar que versión tenemos en nuestra PC en la siguiente página:
http://www.java.com/en/download/help/testvm.xml
En caso que no tengamos ninguna versión la podemos adquirir en:
http://www.java.com/en/download/index.jsp
En caso de tener otro sistema operativo, por ejemplo Mac, la versión de java 1.5 debe estar preinstalada.

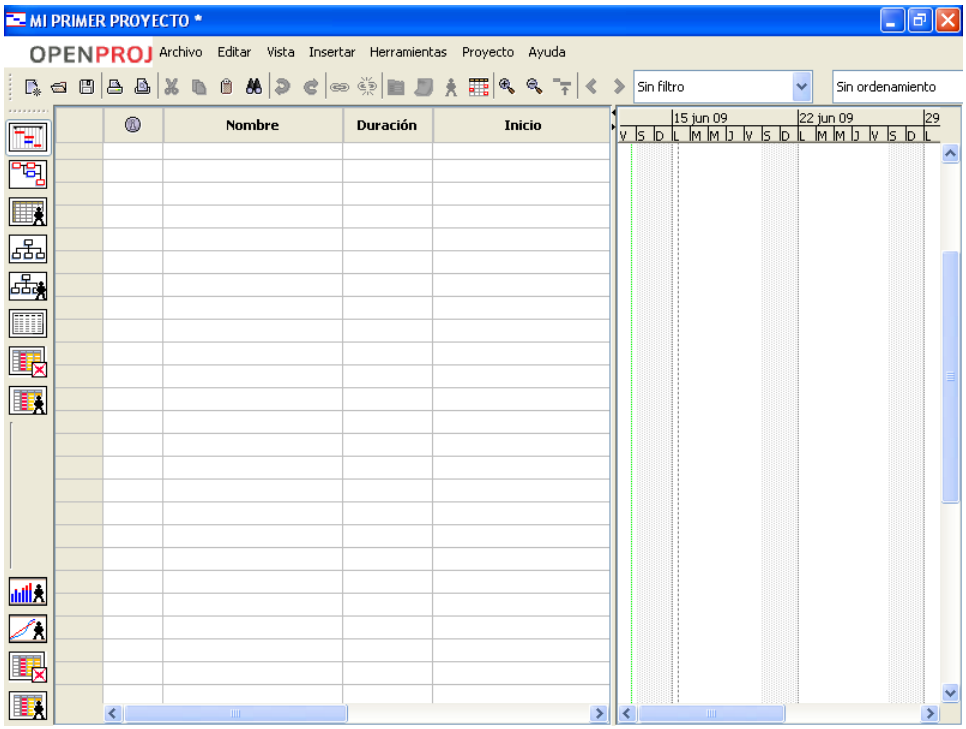
Instalación del software.
Se debe descargar el archivo ejecutable de OpenProj de la página http://openproj.org/ o en http://sourceforge.net/project/showfiles.php?group_id=199315. En esta última se encuentran las versiones para los diferentes sistemas operativos (Linux, Mac). En caso de que nuestra computadora no contenga todos los programas requeridos para el normal funcionamiento pueden emerger ventanas como la siguiente:



La versión de Java que falta la podemos conseguir en http://www.java.com

Pantalla Inicial

A continuación se muestra la pantalla de inicio de OpenProj:



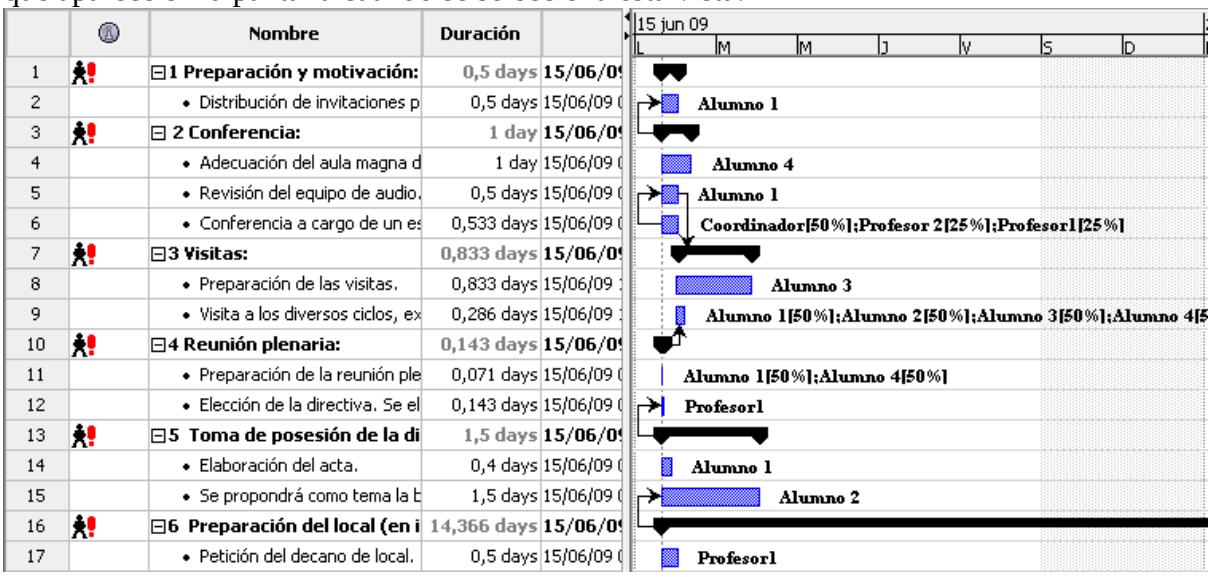
Vistas en OpenProj.

Las vistas de OpenProj permiten crear una combinación de ventanas con diversos campos de acuerdo a las necesidades del proyecto. Las vistas permiten el ingreso de los datos, así como la visualización en mayor detalle de la información que contiene el proyecto. Las vistas con las que cuenta OpenProj son las siguientes:

- Vista Diagrama de Gantt.
- Vista Diagrama de Red.
- Vista Recursos.
- Vista WBS.
- Vista RBS.
- Vista Uso de Tareas.
- Vista Uso de Recursos

Vista Diagrama de Gantt.- Esta es la vista inicial por defecto, en la izquierda de esta vista se enumeran las tareas, y en la derecha aparece el diagrama de Gantt. Con esta vista se pueden apreciar la estructura de las tareas y su ordenamiento en el tiempo. Esta vista se activa dando clic en el icono **Gantt:**

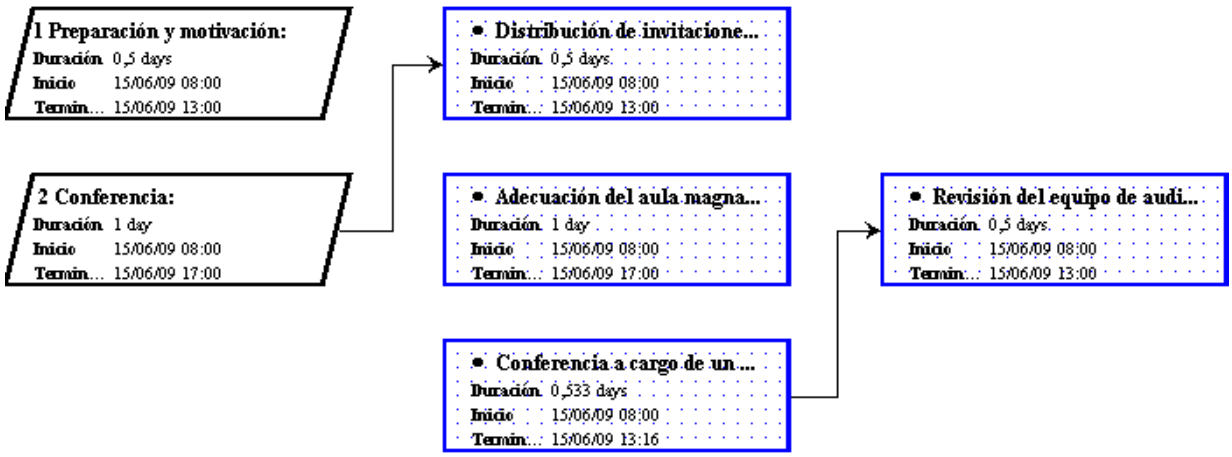
O también por medio del menú **vista: Gantt.** A continuación se muestra un ejemplo de lo que aparece en la pantalla cuando se selecciona esta vista:



	Nombre	Duración	
1	1 Preparación y motivación:	0,5 days	15/06/09
2	• Distribución de invitaciones p	0,5 days	15/06/09
3	2 Conferencia:	1 day	15/06/09
4	• Adecuación del aula magna d	1 day	15/06/09
5	• Revisión del equipo de audio.	0,5 days	15/06/09
6	• Conferencia a cargo de un es	0,533 days	15/06/09
7	3 Visitas:	0,833 days	15/06/09
8	• Preparación de las visitas.	0,833 days	15/06/09
9	• Visita a los diversos ciclos, ex	0,286 days	15/06/09
10	4 Reunión plenaria:	0,143 days	15/06/09
11	• Preparación de la reunión ple	0,071 days	15/06/09
12	• Elección de la directiva. Se el	0,143 days	15/06/09
13	5 Toma de posesión de la di	1,5 days	15/06/09
14	• Elaboración del acta.	0,4 days	15/06/09
15	• Se propondrá como tema la b	1,5 days	15/06/09
16	6 Preparación del local (en i	14,366 days	15/06/09
17	• Petición del decano de local.	0,5 days	15/06/09

Vista Diagrama de Red.- A este diagrama también se le llama diagrama PERT y muestra las dependencias entre tareas usando una gráfica de actividades por nodos, para mostrar la red del proyecto. Esta vista se activa dando clic en el icono **Red,** que se muestra a continuación**.**

O también por medio del menú **Vista: Red.** A continuación se muestra un ejemplo de lo que aparece en la pantalla cuando se selecciona esta vista:



Vista Recursos.- Muestra una tabla que contiene los recursos utilizables para el proyecto. A diferencia de MS Project, OpenProj cuenta con un campo adicional que es el de Dirección de Correo electrónico. Los campos que encontramos en esta lista son:

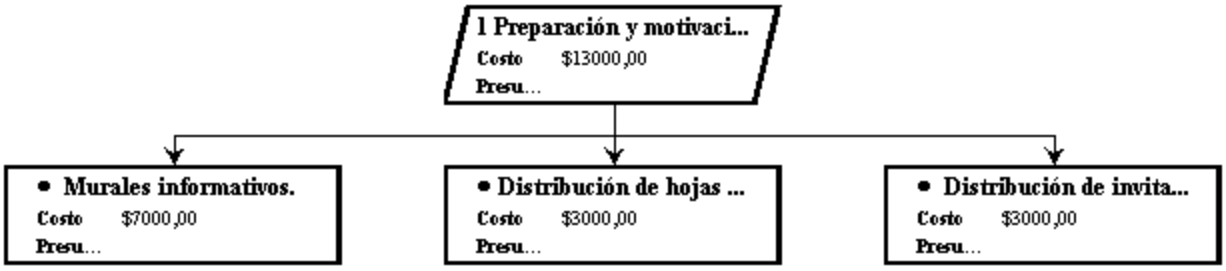
- Número de identificación del recurso
- Nombre
- RBS: nombre del cargo o jerarquía del recurso
- Tipo: trabajo o Material
- Dirección de correo electrónico
- Etiqueta material
- Iniciales
- Grupo: puede utilizarse para hacer agrupaciones (en la esquina superior izquierda del panel)
- Unidades máximas.
- Tasa estándar.
- Tasa sobretiempo
- Calendario base

Esta vista se activa dando clic en el icono **uso de recursos,** o en el menú **vista: Detalle uso de recursos.**

	Nombre	Trabajo	Entorno de Trabajo	Atraso asig...	Demora		15 jun 09 V	S	D	L
1	**Coordinador**	54,412 hor...				Trabajo	0h	0h	0h	6,412h
	• Conferencia a cargo de	2,134 horas	Plano	0 days	0	Trabajo				2,134h
	• Visita a los diversos ciclo	1,143 horas	Plano	0 days	0	Trabajo				1,143h
	4 Reunión plenaria:	0,009 horas	Plano	0 days	0	Trabajo				0,009h
	7 Compra y colocación de	24 horas	Plano	0 days	0	Trabajo				
	• Compra de muebles y ma	8 horas	Plano	0 days	0	Trabajo				
	• Compra de computadora	8 horas	Plano	0 days	0	Trabajo				
	• Adecuación de los mater	8 horas	Plano	0 days	0	Trabajo				
	8 Contratación de una per	3,127 horas	Plano	0 days	0	Trabajo				3,127h
2	• Coordinación con el deca	0 horas				Trabajo	0h	0h	0h	0h
3	• Entrevistas con candidat	0 horas				Trabajo	0h	0h	0h	0h
4	• Elección de asesores.	0 horas				Trabajo	0h	0h	0h	0h
5	• Conferencia a cargo de un	0 horas				Trabajo	0h	0h	0h	0h
6	• Contratación	0 horas				Trabajo	0h	0h	0h	0h
7	• Elección del/la secretario/a	0 horas				Trabajo	0h	0h	0h	0h
8	• Preparación de la reunión	0 horas				Trabajo	0h	0h	0h	0h
9	• Elección de la directiva. Se	0 horas				Trabajo	0h	0h	0h	0h
10	Profesor 2	17,223 horas				Trabajo	0h	0h	0h	6,218h

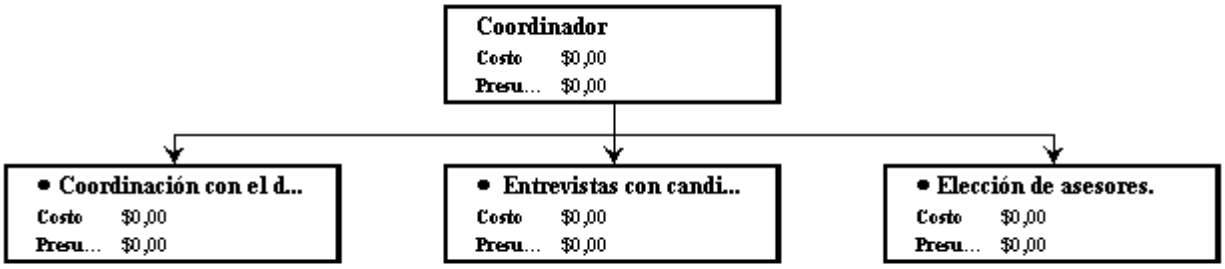
Vista WBS.- Muestra una jerarquía de tareas del proyecto. Esta jerarquía se relaciona con la que se observa en el gráfico Gantt. Esta vista se activa dando clic en el icono **WBS** que se muestra a continuación

O también por medio del menú **Vista: WBS.** A continuación se muestra un ejemplo de lo que aparece en la pantalla cuando se selecciona esta vista:

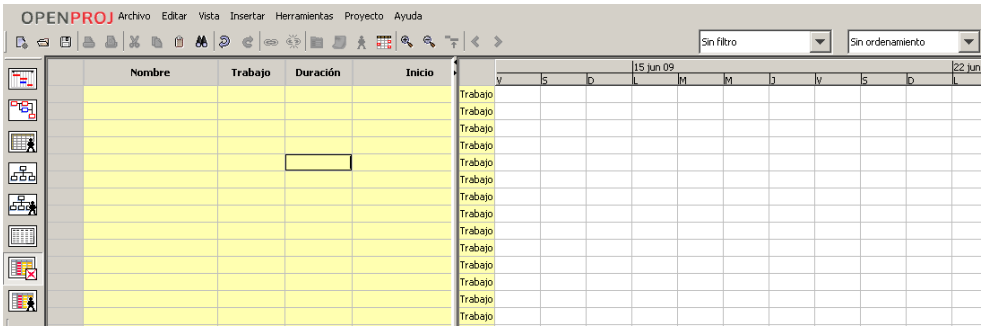


Vista RBS.- Esta vista es muy similar a la WBS. La gráfica RBS muestra los recursos en un árbol jerárquico que corresponde a la estructura definida en la Hoja de Recursos. Esta opción es útil sólo cuando se dispone de muchos recursos para el proyecto y se activa dando clic en el icono **RBS,** que se muestra a continuación:

O también por medio del menú **Vista: RBS.** A continuación se muestra un ejemplo de lo que aparece en la pantalla cuando se selecciona esta vista:



Vista Uso de Tareas.- Hay dos formas para acceder a la *vista uso de tareas*. La primera es seleccionar desde el menú **vista** la opción **detalle uso de tarea.** Donde aparece la siguiente ventana:



La otra forma para acceder a la **vista detalle uso de tareas,** es dando clic en el icono **tareas usadas:**

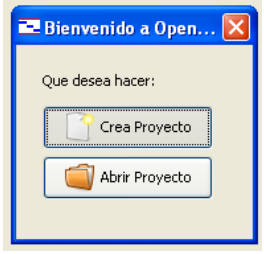
Esta vista muestra el Detalle de Uso de Tareas sincronizando sus contenidos con la parte de arriba de la pantalla.

Vista Uso de Recursos.-Esta vista muestra el Detalle del Uso de Recursos, sincronizando sus contenidos con la parte de arriba de la pantalla. Para acceder a esta vista se debe hacer clic en el menú **vista: Recursos.** A continuación se muestra un ejemplo:

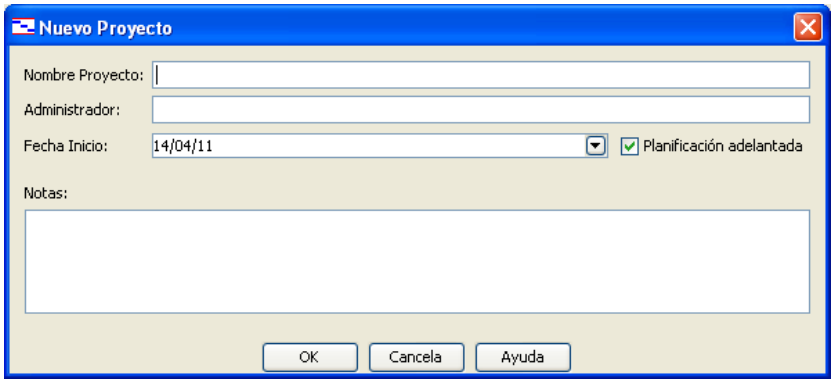
		Nombre	RBS	Tipo	Dirección de Correo elec...	Etiqueta material
1		⊟**Coordinador**		**Trabajo**		
2		• Coordinación con el decai		Trabajo		
3		• Entrevistas con candidat		Trabajo		
4		• Elección de asesores.		Trabajo		
5		• Conferencia a cargo de un		Trabajo		
6		• Contratación		Trabajo		
7		• Elección del/la secretario/a		Trabajo		
8		• Preparación de la reunión		Trabajo		
9		• Elección de la directiva. Se		Trabajo		
10		Profesor 2		Trabajo		

Creación de un proyecto.

Para crear un proyecto es necesario seleccionar la opción **Crear Proyecto** en la ventana emergente de la opción **Archivo**

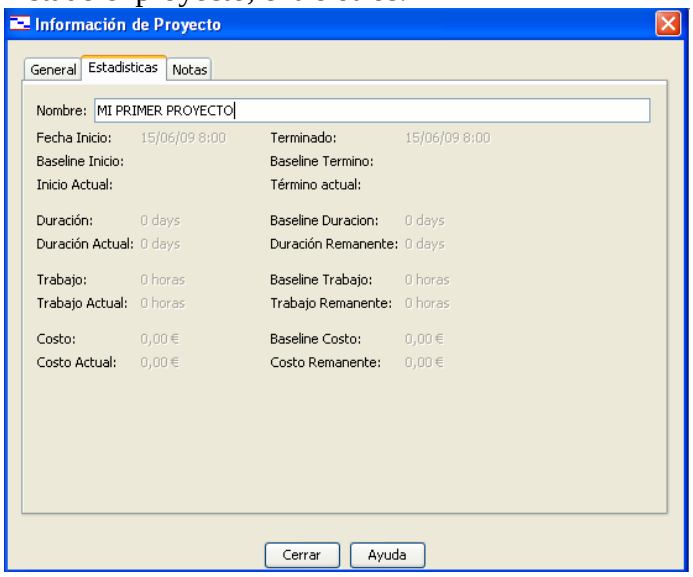


Proporcionar los datos requeridos en el siguiente cuadro de diálogo, las notas son opcionales.



Si se elige la **Planificación adelantada** aparece la **Fecha Inicio** en el caso contrario aparecerá la **Fecha Término.** Una vez que se almacena esta información, si seleccionamos **Proyecto-Información Proyecto** Aparecerá la información almacenada anteriormente más otras opciones almacenadas en tres pestañas

En la primera pestaña **General** encontraremos información como: Prioridad, Tipo de Proyecto, Riesgo y Estado el proyecto, entre otros.

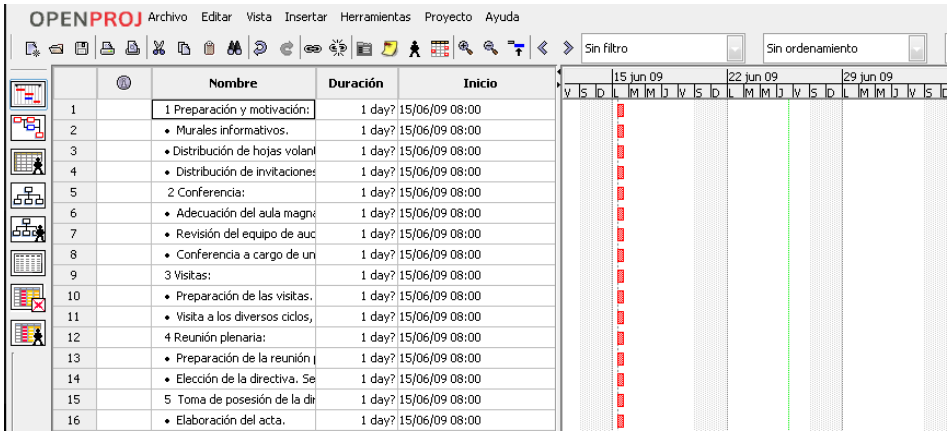


En la pestaña Estadísticas encontraremos datos como Fecha de Inicio, Fecha tentativa de término, Duración, Duración actual, Duración remanente, Trabajo, Trabajo actual, Trabajo remanente, Costo, Costo actual, Costo remanente; esta información indica el inicio lo actual y lo que hace falta para terminar el proyecto. Y por último, en la pestaña **Notas** se pueden incluir datos o notas pertinentes.

TAREAS.

Para realizar el proceso de creación de tareas, es importante tener en cuenta que este se puede hacer desde la *vista Gantt*, o desde *la vista uso de tareas*.

Creación de tareas desde la *"vista Gantt"*.- La *vista Gantt* contiene del lado izquierdo la tabla Gantt, que es donde se introducen los datos, y del lado derecho el diagrama de Gantt, que es donde aparece el gráfico de las tareas con sus tiempos y relaciones. En la tabla Gantt, aparecen una serie de campos en los cuales debe introducirse la información acerca de las tareas. El primer campo es el de **nombre,** en el cual debe escribirse el nombre de las tareas (se toma como ejemplo la creación de un club de ciencia) de la siguiente manera:



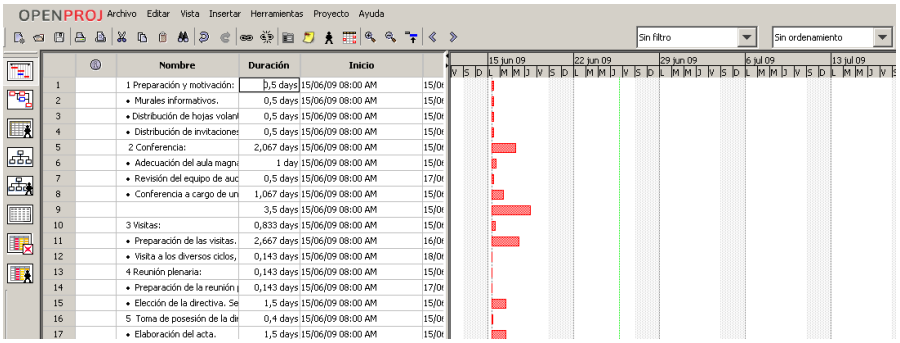
	Nombre	Duración	Inicio
1	1 Preparación y motivación:	1 day?	15/06/09 08:00
2	• Murales informativos.	1 day?	15/06/09 08:00
3	• Distribución de hojas volant	1 day?	15/06/09 08:00
4	• Distribución de invitaciones	1 day?	15/06/09 08:00
5	2 Conferencia:	1 day?	15/06/09 08:00
6	• Adecuación del aula magna	1 day?	15/06/09 08:00
7	• Revisión del equipo de auc	1 day?	15/06/09 08:00
8	• Conferencia a cargo de un	1 day?	15/06/09 08:00
9	3 Visitas:	1 day?	15/06/09 08:00
10	• Preparación de las visitas.	1 day?	15/06/09 08:00
11	• Visita a los diversos ciclos,	1 day?	15/06/09 08:00
12	4 Reunión plenaria:	1 day?	15/06/09 08:00
13	• Preparación de la reunión	1 day?	15/06/09 08:00
14	• Elección de la directiva. Se	1 day?	15/06/09 08:00
15	5 Toma de posesión de la dir	1 day?	15/06/09 08:00
16	• Elaboración del acta.	1 day?	15/06/09 08:00

Duración de las tareas.- El segundo campo que se puede ver en la tabla Gantt, es el de duración, en el cual se debe especificar el tiempo que se le asignará a cada una de las tareas. La duración de una tarea en OpenProj puede estar dada en horas o días. La convención para asignar la duración de cada una de las tareas es la siguiente:

Convención	Tiempo de duración
H	**Horas**
D	**Días**
Ed	**Día transcurrido**

Así, por ejemplo, si se requiere que una tarea tenga duración continua, en un período no laborable, se debe añadir al tiempo de duración **ed.**

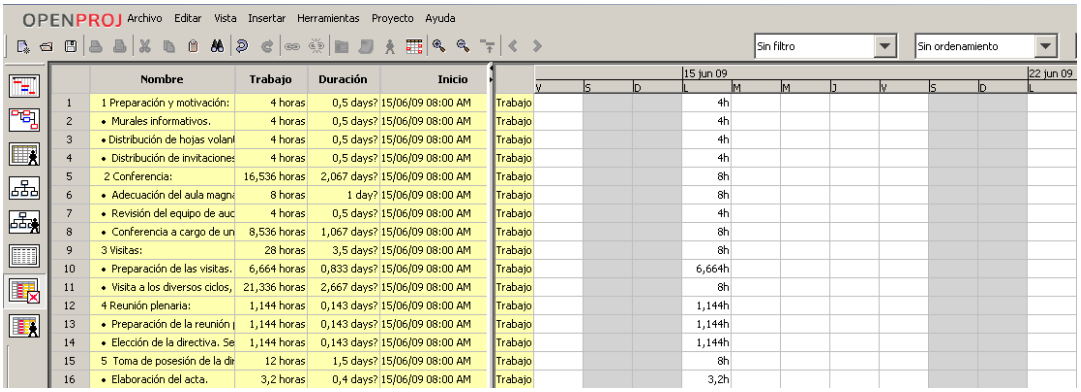
Para introducir la duración de las tareas, es necesario escribir en el campo duración, frente a cada tarea, el número que indique la medida temporal y la convención necesaria. Por ejemplo, si una tarea tiene como duración 3 meses, se debe escribir 60d. O si una tarea debe durar más de 1 día laboral en forma continua, se escribe 1ed.



A medida que se introduce el tiempo de duración de una tarea, se modifica el tamaño de su barra correspondiente en el **diagrama de Gantt** que esta del lado derecho de la pantalla. OpenProj calcula las fechas de inicio tomando como referencia la fecha de inicio del proyecto. Las fechas de finalización se calculan en base a la duración establecida para cada tarea.

Existen *tareas críticas* y *tareas no críticas*. Una *tarea crítica* es aquella que al variar su duración, afecta la duración del proyecto completo. Estas tareas pueden definirse con alguna convención particular, de tal manera que en el *diagrama de Gantt* aparezca esta tarea de manera diferenciada, puede ser con otro color de barra. Una *tarea no crítica* es aquella que hasta cierto punto su duración puede variar sin que modifique la fecha de determinación del proyecto.

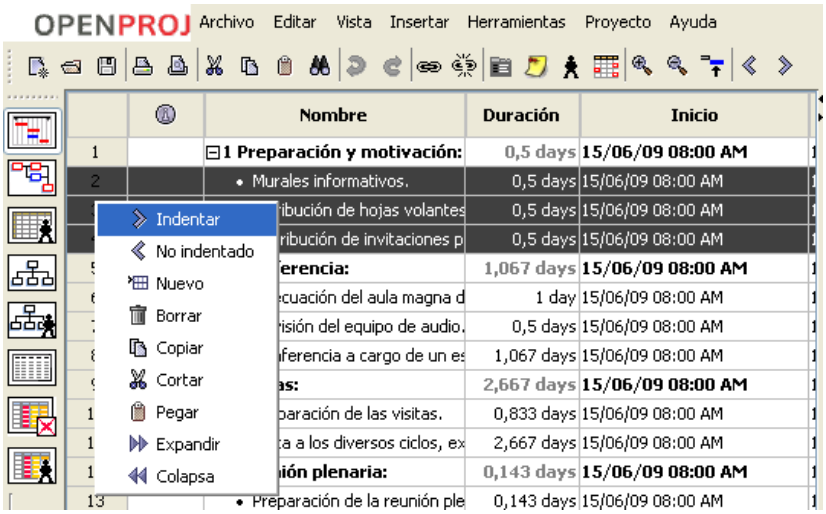
Creación de tareas desde la *"vista uso de tareas"*.- Para crear tareas desde la *vista uso de tareas* se procede a ingresar los nombres de las tareas en el **campo nombre,** seguido de su duración en el **campo trabajo,** y los campos siguientes aparecen automáticamente por los datos ingresados, a continuación se muestra un ejemplo:



Tareas resumen y subtareas.- En un proyecto se pueden asignar diferentes niveles a cada una de las tareas, es decir, que una tarea puede contener otras tareas de tal manera que la ejecución de una tarea, depende de las subtareas que tenga asociadas. Una *tarea resumen* se compone de subtareas y contiene la duración, el costo y otro tipo de información de las

tareas que subordina. Las tareas resumen pueden representar las diversas fases de un proyecto y aparecen en negrilla, sus barras en el *diagrama de Gantt***,** son diferentes a las de una tarea subordinada. Para establecer las tareas resumen, se debe crear el esquema de un proyecto, es decir, determinar previamente sus fases, con las tareas asociadas a cada una de ellas, e ingresarlas en el proyecto, de tal manera que la *tarea resumen* se encuentre al inicio, y las *subtareas* se encuentren ubicadas debajo de ésta.

Para establecer las tareas *resumen* y *subordinadas*, se deben seleccionar la tarea resumen junto a sus subordinadas, dar clic derecho y elegir la opción **indentar** como se muestra en el siguiente ejemplo:



Otra forma de establecer la subordinación de las tareas es seleccionando la tarea resumen junto a las subtareas, y dando clic en el icono llamado **sangría:**

RECURSOS

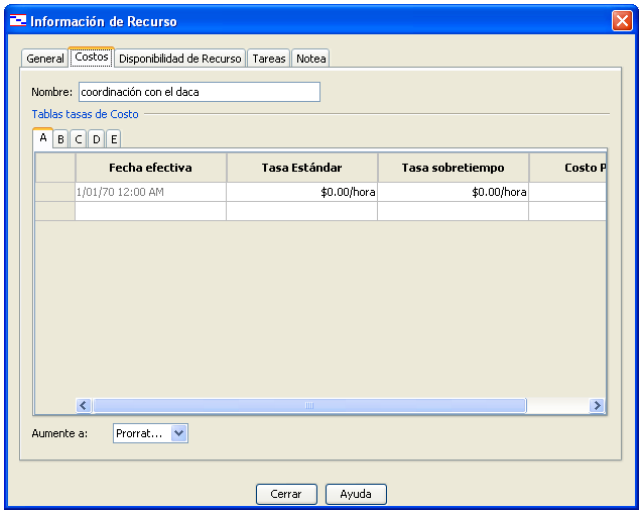
Asignación de recursos.- Para asignar los diferentes recursos de un proyecto, es necesario que se primero se creen de la siguiente manera: Dar clic en la vista **uso de recursos:**

La *vista uso de recursos* sirve para ingresar cada uno de los recursos con sus respectivas características:

Para crear un recurso basta con escribir el nombre del recurso en el campo **nombre** y escribir en el campo **trabajo** la duración de su trabajo asignado. A continuación se muestra un ejemplo:

	Nombre	Trabajo	Entorno de Trabajo	Atraso asig...
1	**Coordinador**	164,997 ho...		
	Coordinación con el deca	*3 horas*	*Plano*	*0 days*
	Entrevistas con candidat	*12 horas*	*Plano*	*0 days*
	Elección de asesores.	*1 hora*	*Plano*	*0 days*
	Conferencia a cargo de	*0,8 horas*	*Plano*	*0 days*
	Contratación	*3 horas*	*Plano*	*0 days*
	Elección del/la secretario	*3 horas*	*Plano*	*0 days*
	Preparación de la reunió	*0,428 horas*	*Plano*	*0 days*
	Elección de la directiva.	*0,428 horas*	*Plano*	*0 days*
2	**Profesor 2**	65,57 horas		
	Compra de computado	*6 horas*	*Plano*	*0 days*
	Preparación de la reun	*0,428 horas*	*Plano*	*0 days*
	Publicar el aviso en el	*12 horas*	*Plano*	*0 days*
	Conferencia a cargo d	*1,6 horas*	*Plano*	*0 days*
	Elección de asesores.	*1 hora*	*Plano*	*0 days*
	Elección de la directiva	*0,428 horas*	*Plano*	*0 days*

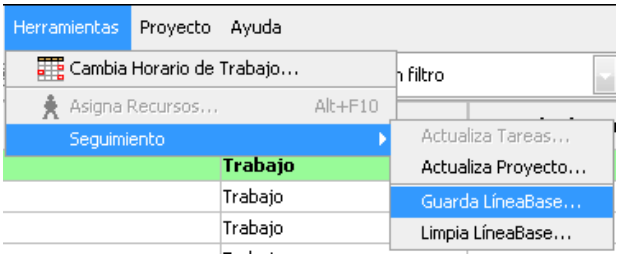
Se tienen 5 tablas de costos predeterminadas: A, B, C, D y E, se debe seleccionar una de ellas según las necesidades del proyecto. Por defecto se selecciona la tabla A. Para crear las tablas de costos se debe dar doble clic en el campo **Tabla Factores de costo,** y emergerá la siguiente ventana, donde deben hacerse las modificaciones respectivas para la información general del recurso, el costo, la disponibilidad, las tareas y las notas.



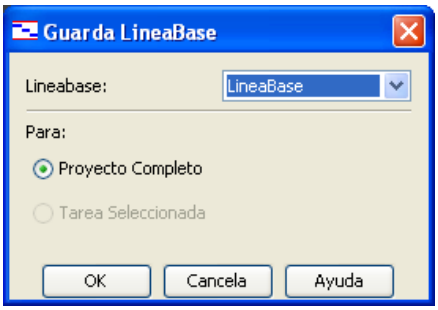
Para asignar recursos a cada una de las tareas es necesario ubicarse en la vista **Diagrama de Gantt** en el campo **Nombres del recurso** y escribir el nombre del recurso respectivo. Cuando se cuenta con más de un recurso, los nombres deben separarse por comas, como se muestra en el siguiente ejemplo:

		Nombre	Duración	Inicio	Terminado	Prede...	Nombres del Recurso
1		**1 Preparación y motivación:**	0,5 days	**18/05/09 08:00**	**18/05/09 11:00**		
2		• Murales informativos.	0,5 days	18/05/09 08:00	18/05/09 11:00		Alumno 2;Cartulina para mu...
3		• Distribución de hojas volantes ir	0,5 days	18/05/09 08:00	18/05/09 11:00	2SS	Alumno 1
4		• Distribución de invitaciones par	0,5 days	18/05/09 08:00	18/05/09 11:00	3SS	Alumno 3
5		**2 Conferencia:**	2,067 days	**18/05/09 11:00**	**19/05/09 16:24**	**1**	
6		• Adecuación del aula magna de l	1 day	18/05/09 11:00	19/05/09 09:00	4	Alumno 4
7		• Revisión del equipo de audio.	0,5 days	18/05/09 11:00	18/05/09 15:00	6SS	Alumno 1
8		• Conferencia a cargo de un espe	1,067 days	19/05/09 09:00	19/05/09 16:24	6;7	Coordinador[50%];Profesor...
9		**3 Visitas:**	3,5 days	**19/05/09 16:24**	**22/05/09 13:24**	**5**	
10		• Preparación de las visitas.	0,833 days	19/05/09 16:24	20/05/09 13:24	8	Alumno 2
11		• Visita a los diversos ciclos, expli	2,667 days	20/05/09 13:24	22/05/09 13:24	10	Alumno 3
12		**4 Reunión plenaria:**	0,143 days	**22/05/09 13:24**	**22/05/09 14:15**	**9**	
13		• Preparación de la reunión plena	0,143 days	22/05/09 13:24	22/05/09 14:15	11	Coordinador[50%];Profesor...
14		• Elección de la directiva. Se eleg	0,143 days	22/05/09 13:24	22/05/09 14:15	13SS	Coordinador[50%];Profesor...

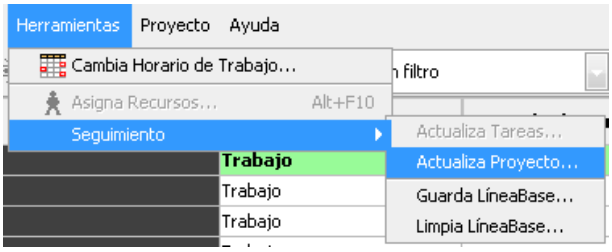
Seguimiento del proyecto y Línea de Base.- La línea de base es el punto de partida para realizar el *seguimiento del proyecto*. Es la estimación inicial del diseñador del proyecto, permite evaluar como avanza la ejecución con respecto a la planeación. La *línea de base* es una imagen guardada del proyecto en un momento inicial específico que permite comparar posteriormente lo que se tenía planeado con lo que está ocurriendo con el proyecto. Para la creación de una línea de base, se deben seguir los siguientes pasos: Clic en el menú **Herramientas, seguimiento, Guardar Línea base** como se muestra a continuación:



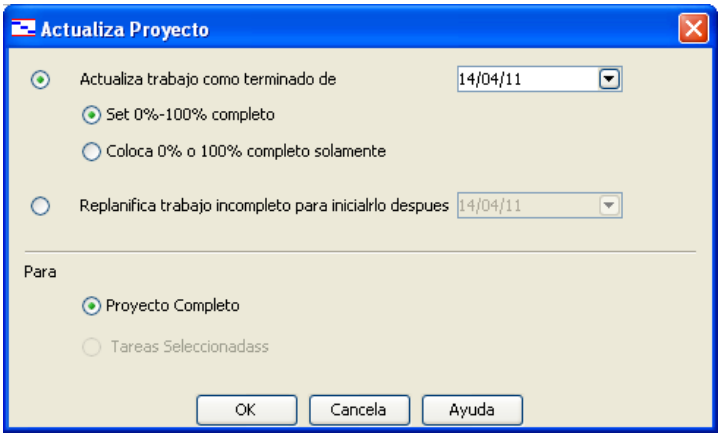
Al seleccionar lo anterior aparece la siguiente ventana en donde se debe guardar la línea de base e indicar si es para el **proyecto completo,** o para una **tarea seleccionada,** y dar clic en **OK.**



Después de establecer la *línea base* como punto de comparación, se puede llevar a cabo el *seguimiento* del proyecto, el cual consiste en comparar la planeación y la ejecución real del proyecto para tomar las medidas correctivas que sean necesarias. Para dar *seguimiento* hay que seleccionar el menú **Herramientas, seguimiento,** donde aparecen las opciones de **actualizar proyecto, guardar lineabase y limpia lineabase.**



Para iniciar el seguimiento del proyecto primero se debe *actualizar*. Para *Actualizar proyecto* usando la opción "trabajo terminado de" se hace referencia a la fecha de corte, la cual supone que todas las tareas se completaron, así:



La actualización por porcentaje completado permite registrar el proyecto o las tareas previamente seleccionadas con el porcentaje de cumplimiento de las tareas o el proyecto, respecto a una fecha propuesta. La opción **coloca 0% o 100% completo solamente** le asignará dichos porcentajes de acuerdo a la terminación de la tarea. Si una tarea se ha terminado completamente, se le asigna el 100%, de lo contrario le asigna el 0%.

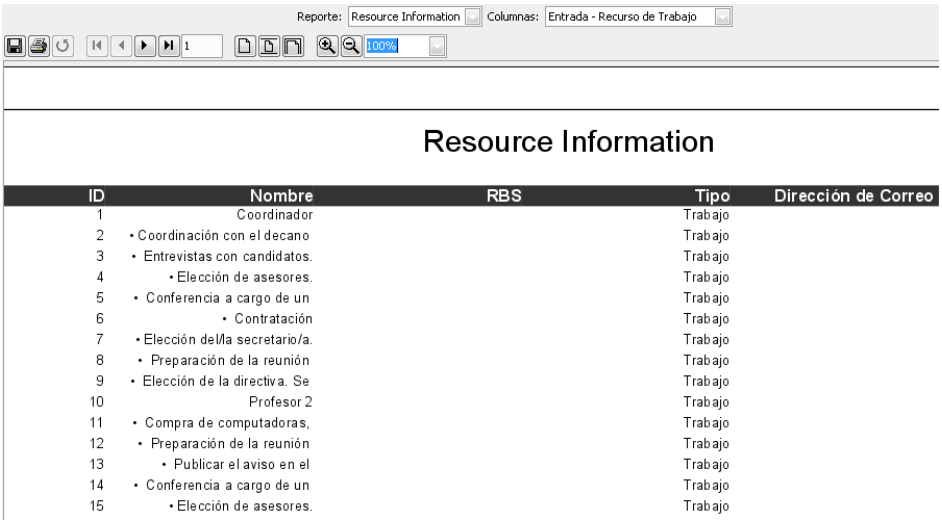
Informes.

A diferencia de MS Project, los informes en OpenProj no son muchos, sin embargo su contenido es similar al de algunos de los informes de MS Project. Para ver los informes en OpenProj, se debe ir a la vista **Informes o reportes** o al icono siguiente:

Los reportes que se generan son para visualización o para impresión. Existen 4 formas preestablecidas para la generación de informes:

Informe de Detalles del proyecto (Project Detail).- En este informe se encuentran los detalles del proyecto, así como la información más relevante del mismo. Este tipo de informes es muy útil para los directivos de nivel superior por su presentación concisa de los datos generales del proyecto, a continuación un ejemplo:

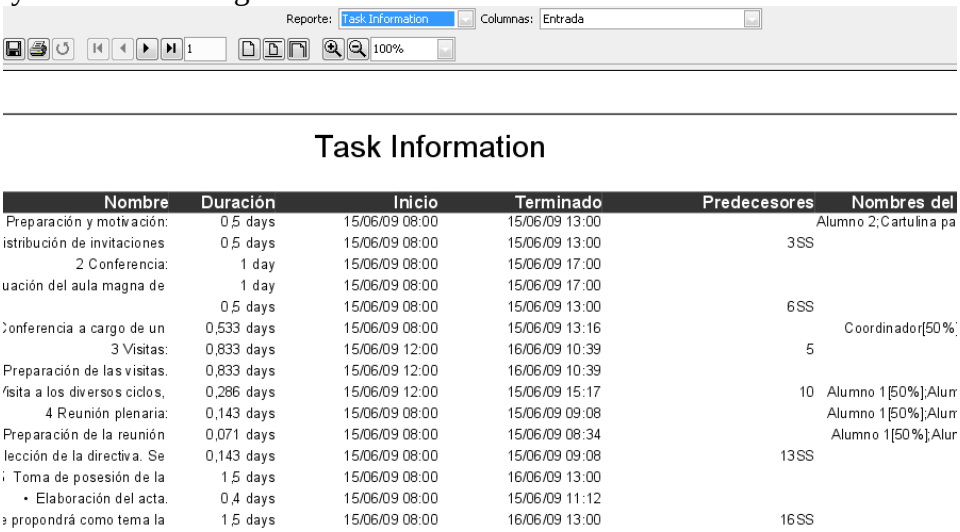
Información de los recursos (Resource information).- Este informe sirve para observar los detalles de los recursos que se utilizan en el proyecto. Como en el siguiente ejemplo:


Reporte: Resource Information Columnas: Entrada - Recurso de Trabajo

Resource Information

ID	Nombre	RBS	Tipo	Dirección de Correo
1	Coordinador		Trabajo	
2	• Coordinación con el decano		Trabajo	
3	• Entrevistas con candidatos.		Trabajo	
4	• Elección de asesores.		Trabajo	
5	• Conferencia a cargo de un		Trabajo	
6	• Contratación		Trabajo	
7	• Elección del/la secretario/a.		Trabajo	
8	• Preparación de la reunión		Trabajo	
9	• Elección de la directiva. Se		Trabajo	
10	Profesor 2		Trabajo	
11	• Compra de computadoras,		Trabajo	
12	• Preparación de la reunión		Trabajo	
13	• Publicar el aviso en el		Trabajo	
14	• Conferencia a cargo de un		Trabajo	
15	• Elección de asesores.		Trabajo	

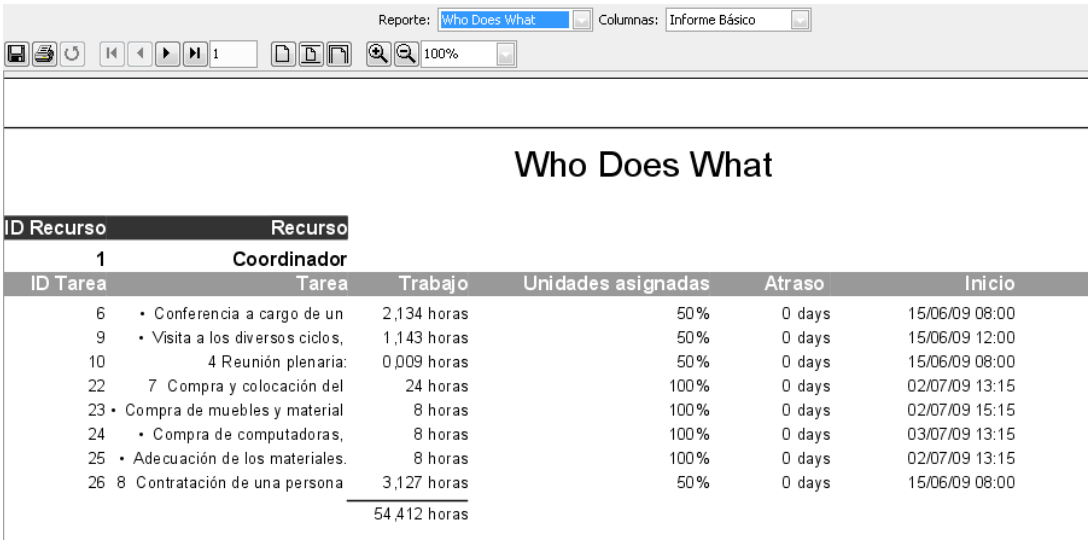
Información de tareas (Task information).- Este informe sirve para ver las tareas del proyecto y sus recursos asignados.


Reporte: Task Information Columnas: Entrada

Task Information

Nombre	Duración	Inicio	Terminado	Predecesores	Nombres del
Preparación y motivación:	0,5 days	15/06/09 08:00	15/06/09 13:00		Alumno 2;Cartulina pa
istribución de invitaciones	0,5 days	15/06/09 08:00	15/06/09 13:00	3SS	
2 Conferencia:	1 day	15/06/09 08:00	15/06/09 17:00		
uación del aula magna de	1 day	15/06/09 08:00	15/06/09 17:00		
	0,5 days	15/06/09 08:00	15/06/09 13:00	6SS	
Conferencia a cargo de un	0,533 days	15/06/09 08:00	15/06/09 13:16		Coordinador[50%]
3 Visitas:	0,833 days	15/06/09 12:00	16/06/09 10:39	5	
Preparación de las visitas.	0,833 days	15/06/09 12:00	16/06/09 10:39		
/isita a los diversos ciclos,	0,286 days	15/06/09 12:00	15/06/09 15:17	10	Alumno 1[50%];Alum
4 Reunión plenaria:	0,143 days	15/06/09 08:00	15/06/09 09:08		Alumno 1[50%];Alum
Preparación de la reunión	0,071 days	15/06/09 08:00	15/06/09 08:34		Alumno 1[50%];Alur
lección de la directiva. Se	0,143 days	15/06/09 08:00	15/06/09 09:08	13SS	
Toma de posesión de la	1,5 days	15/06/09 08:00	16/06/09 13:00		
• Elaboración del acta.	0,4 days	15/06/09 08:00	15/06/09 11:12		
e propondrá como tema la	1,5 days	15/06/09 08:00	16/06/09 13:00	16SS	

Quién hace qué (Who Does what).- En este informe se puede visualizar el avance y cumplimiento de las tareas asignadas a un recurso en particular.


Reporte: Who Does What Columnas: Informe Básico

100%

Who Does What

ID Recurso	Recurso
1	Coordinador

ID Tarea	Tarea	Trabajo	Unidades asignadas	Atraso	Inicio
6	• Conferencia a cargo de un	2,134 horas	50%	0 days	15/06/09 08:00
9	• Visita a los diversos ciclos,	1,143 horas	50%	0 days	15/06/09 12:00
10	4 Reunión plenaria:	0,009 horas	50%	0 days	15/06/09 08:00
22	7 Compra y colocación del	24 horas	100%	0 days	02/07/09 13:15
23	• Compra de muebles y material	8 horas	100%	0 days	02/07/09 15:15
24	• Compra de computadoras,	8 horas	100%	0 days	03/07/09 13:15
25	• Adecuación de los materiales.	8 horas	100%	0 days	02/07/09 13:15
26	8 Contratación de una persona	3,127 horas	50%	0 days	15/06/09 08:00
		54,412 horas			

Si se desea profundizar más en este, se pueden consultar las siguientes referencias, que fueron la fuente donde se obtuvo la información presentada en esta sección:

- Manual de MS Project de e-magister.com
- Ayuda en línea de OpenProj
- OpenProj Univesidad Nacional de Colombia
- www.Serena.com
- www.openproj.org

Capítulo VIII Recuperación de proyectos.

VIII.1 Introducción

Un proyecto de desarrollo de SW puede quedar suspendido o abandonado por varias razones, entre las que comúnmente se encuentran: un retraso excesivo, incumplimiento de los objetivos o un alto incremento de costos. En algunos casos, a pesar de que se presenten algunos de estos problemas, el proyecto todavía sigue en marcha aunque su estado sea tan incierto que posiblemente se haya pensado en abandonarlo. Hay casos en los que es posible retomar estos proyectos y llevarlos a buen término aplicando estrategias para la recuperación de los mismos.

La recuperación de un proyecto no siempre es posible. Para tomar una decisión sobre recuperarlo o no, es necesario identificar sus problemas y hacer una predicción de lo que se espera de él para decidir entonces si tiene remedio o no. En caso afirmativo, habrá que llevar a cabo una nueva planificación con nuevas estimaciones del esfuerzo, tiempo y costo. Además habrá que cerciorarse de que al retomarlo, se apliquen correctamente: la gestión de riesgos, la gestión de calidad y la gestión de recursos.

La Ingeniería del Software es una rama de la Ingeniería en Computación dedicada al desarrollo y mantenimiento de aplicaciones de software, a través de la aplicación y uso de principios, técnicas y métodos de la ingeniería, las ciencias de la computación, la administración, la gestión de proyectos, el dominio de aplicación y otros campos relacionados [Pressman, 1998]. El desarrollo de software es un proceso complejo [Pressman, 1998, Sommerville, 2006], en el cual inciden, entre otras, las siguientes dificultades:

- La complejidad inherente al dominio del problema.
- La dificultad en el manejo del proceso de desarrollo.
- El interés por desarrollar software que sea altamente reutilizable y extensible.
- Las técnicas y procesos que mejor funcionan para un desarrollador o para un pequeño grupo no suelen “escalar” fácilmente al desarrollo de sistemas de gran envergadura.
- El ritmo de evolución del hardware y del propio software en las condiciones de un mercado muy competido provoca una demanda y un conjunto de expectativas crecientes que son difíciles de cubrir con sistemas de software de alta calidad.
- El ritmo de evolución del hardware ha sido siempre muy superior al del software.

Tales dificultades comúnmente generan riesgos, los cuales hacen peligrar la exitosa culminación del proceso de desarrollo y en diferentes escenarios conllevan al abandono parcial o definitivo del proyecto. En el desarrollo de proyectos de software el concepto “riesgo” [Sommerville, 2006; Pressman, 1998] se refiere comúnmente a problemas de

comunicación con el cliente, requerimientos poco comprensibles, arquitecturas poco comprensibles, problemas con la tecnología subyacente, problemas que pueden emerger en el proceso de desarrollo, falta de experiencia en el equipo de desarrollo, estimaciones erróneas de tiempo, esfuerzo y costo, entre otros.

Los proyectos de desarrollo de software con un alto riesgo de abandono se caracterizan por una combinación de síntomas y causas, encontrándose comúnmente entre éstos los siguientes:

- Selección inadecuada del ciclo de vida y metodología de desarrollo.
- Insuficiente comprensión sobre el alcance del proyecto y los requerimientos del dominio del problema.
- Insuficiente planeación y estimación del proyecto.
- Carencia de comunicación con la directiva de la organización y con el cliente del proyecto.
- Incomprensión sobre la arquitectura requerida para la aplicación.
- Desconocimiento sobre las tecnologías, herramientas y lenguajes requeridos en el proceso de desarrollo

En la sección VIII.4 de este capítulo se presenta la propuesta de un plan de recuperación de proyectos [Soto-Galindo & González, 2010; González & Soto-Galindo, 2010], inspirado en el proceso de diagnóstico médico, el cual debe constituir un valioso apoyo para el desarrollador de software, líder de proyecto o equipo de desarrollo en la compleja tarea de recuperación de proyectos de software.

VIII.2 Panorama del Desarrollo de Proyectos de Software: Conclusión vs. Abandono

A mediados de los noventas, el Standish Group publicó su CHAOS report [Standish Group, 2009], abarcando una muestra aproximada de 8000 proyectos de software distribuidos en más de 350 empresas ubicadas en América del Norte. De dicho estudio se aprecia que sólo el 16.2% y el 9% de las pequeñas y grandes empresas respectivamente, terminaron el desarrollo dentro de los costos y de los plazos establecidos en la planeación. El 52,7% de los proyectos finalizó excedido en el presupuesto (sobre el 189%) y con grandes retrasos en los tiempos de entrega (sobre el 122%), además de ofrecer mucho menos funcionalidad de la que se había pactado dentro del alcance. Finalmente, el 31.1% de los proyectos fueron cancelados (lo que correspondía a aproximadamente 81 billones de dólares desperdiciados). Entre las principales causas que llevaron al incumplimiento de las estimaciones de tiempo y presupuesto, o incluso al abandono de los proyectos se encontraron las siguientes: requerimientos incompletos, falta de involucramiento del cliente, recursos insuficientes, expectativas poco realistas, carencia de soporte ejecutivo, requerimientos cambiantes,

planeación deficiente, falta de administración de las tecnologías de la información, y desconocimiento tecnológico.

Daniel Piorun [Piorun, 2001] en su publicación ¿Por qué fracasan los proyectos?, propone como los principales factores determinantes de un desarrollo fallido, los siguientes tres puntos: cambios en los objetivos establecidos a nivel estratégico, la pobre utilización de metodologías de trabajo, y problemas humanos de conducción, comunicación y conflictos entre las personas. Uno de los últimos estudios efectuados por The Standish Group - CHAOS 2009 muestra un retroceso en el número de proyectos que se logra concluir con éxito. Las estadísticas arrojan que sólo el 32% de los desarrollos son entregados en tiempo y forma, mientras que un 44% presenta una desviación negativa en el alcance cubierto y desfase en tiempos de entrega. El restante 24% de los proyectos se cancela.

El panorama nacional no luce muy diferente al observado en el resto del mundo. La información oficial acerca de los proyectos de software con problemas en México es escasa, se sabe que gran número de las consultorías constituidas en el país presentan o han presentado algún tipo de complicación, que van desde retrasos en tiempos de entrega, hasta el abandono del proyecto. En la mayoría de estos casos, los problemas presentados exceden los costos estimados por los equipos de desarrollo y han incidido directamente en pérdidas económicas tanto del lado del cliente como del proveedor. Si tomamos en consideración que la mayor parte de estos desarrollos fallidos son atendidos usando técnicas secuenciales sin atender la sintomatología de fondo, podremos comprender el por qué este tipo de situaciones resultan normalmente en desastre.

VIII.3 Recuperación de Proyectos de Software

VIII.3.1 La importancia de las estrategias y planes para la recuperación de proyectos de software

Sería prácticamente imposible relacionar la vasta cantidad de textos publicados a la fecha, dedicados a presentar las diferentes metodologías (estrategias, enfoques metodológicos, etc.) existentes para guiar el proceso de desarrollo de software. Una gran parte de estas metodologías y enfoques metodológicos se centra mucho más en las fases del propio proceso de construcción de software que en las actividades encaminadas a la administración, planeación, estimación y aseguramiento de la calidad del desarrollo de software. En este caso, se aprecia un marcado énfasis en la definición y descripción de métodos, actividades, modelos y herramientas para ejecutar las principales fases del proceso de construcción de software, tales como análisis de requerimientos, diseño arquitectónico, diseño de componentes e implementación. Por otra parte, están aquellas metodologías y enfoques metodológicos orientados más a los aspectos de administración, estimación, planeación y calidad del desarrollo de software y que asumen que las fases de construcción pudieran ejecutarse a través de alguna de las tantas estrategias existentes. Sin embargo, es notable que un aspecto tan importante como la recuperación de proyectos de

software no sea comúnmente abordado como parte de estas metodologías de desarrollo como lo son los aspectos de construcción y de administración-gestión.

Entre los escasos enfoques para la recuperación de proyectos difundidos a la fecha, es necesario hacer mención aquí al plan de recuperación propuesto por [McConell, 1997], el cual está orientado a la recuperación de proyectos con serios problemas. En dicho enfoque, la recuperación de un proyecto se basa en la combinación de tres estrategias fundamentales:

- Reducción del tamaño del software.
- Incremento de la productividad del proceso de desarrollo.
- Ajuste de la planeación, considerando el tiempo realmente requerido para el desarrollo del producto.

Estas estrategias están directamente relacionadas con tres elementos claves del proyecto de software: el producto, las personas y el proceso, respectivamente. Antes de iniciar el plan de recuperación, es necesario evaluar la situación para determinar el tipo de plan específico que se requiere, es decir, ¿cuál o cuáles de estas estrategias aplicar? Para cada estrategia, el plan de recuperación evalúa y aplica un conjunto de directrices, dirigidas al elemento clave en cuestión. Por ejemplo, si la estrategia a ejecutar está dirigida al producto, entonces el conjunto de directrices abarca acciones tales como: la estabilización de los requerimientos, la disminución del conjunto de prestaciones, la eliminación de las partes del producto caracterizadas por una baja calidad, la reducción del número de defectos, entre otras.

La verdadera robustez de este plan de recuperación radica en los tres enfoques que combina y en la amplia gama de acciones a ejecutar encaminadas a la recuperación del proyecto con problemas. Sin embargo, la evaluación de la situación para identificar las características y causas que generaron los problemas en el proyecto no queda definida de forma clara, al no proponer el plan de forma detallada los pasos metodológicos a seguir.

Como se analizó en la anterior sección, el incumplimiento y abandono de proyectos de software iniciados es un factor que cada vez afecta más la industria del software tanto en México como a nivel mundial. Varían de proyecto a proyecto la gama de causas que pueden generar su incumplimiento y/o abandono, los signos visibles que indican que el mismo está en problemas, así como los diferentes factores que se involucran en esta trama. Es una realidad que muchos de estos proyectos en problemas podrían salvarse si la directiva, equipos de trabajo y desarrolladores contaran con oportunas estrategias y planes para guiar el proceso de recuperación de los mismos. Hacia esta necesidad va la propuesta de plan para la recuperación de proyectos de software que se describe en VIII.4.

VIII.3.2 Identificación de los problemas

Para recuperar un proyecto es necesario identificar las razones por las cuales no está funcionando adecuadamente. Posteriormente habrá que plantear soluciones para salvarlo o retomarlo, sin dejar de tomar en cuenta que no todos los proyectos son susceptibles de ser recuperados.

Según [McConell, 1997], las características de un proyecto que fracasa son las siguientes:

- No se tiene ni la más remota idea de cuando se podrá terminar.
- El producto esta lleno de defectos.
- Pérdida del control por parte de los directivos del estado del proyecto.
- Pérdida de la confianza del cliente en el equipo de desarrollo.
- Desmotivación del personal (exceso de horas extra trabajadas, baja moral).

Para recuperar un proyecto con problemas ante todo es necesario hacer un serio análisis de dichos problemas, detectar las causas que los originan y establecer medidas correctivas que las eliminen. Es necesario revisar procesos, organización, personas y herramientas. Entre los criterios para evaluar un proyecto se encuentran: el ambiente de negocio, el ciclo de vida del proyecto y el marco de la gerencia del proyecto es decir, el "ambiente político" en el que se desarrolla el proyecto. Además de tomar en cuenta los aspectos técnicos, hay que considerar si el fracaso del proyecto se debe a otros factores. Identificar a los que pudieran estar saboteando el proyecto puede ser la solución. Por ejemplo, una empresa detecta que tiene pérdidas debido a robos de su producto, por lo que decide encargar a un equipo de desarrolladores "A" que desarrollen un subsistema hardware-software que detecte estos eventos, el subsistema "A" se instala dentro del "sistema de alerta" de la empresa y resulta que no funciona. Para que las autoridades constaten que se están tomando medidas al respecto, se encarga a un equipo de desarrolladores "B" que elabore un subsistema bajo principios de funcionamiento diferentes al subsistema "A", este subsistema "B" se instala también en el "sistema de alerta" de la empresa y resulta que tampoco funciona. Cuando, después de un tiempo de pérdidas considerables, la empresa pretende contratar a un grupo de desarrolladores para un subsistema "C" que verifique el correcto funcionamiento de los subsistemas anteriores. Alguien con visión concluye que mientras los sistemas sean instalados en el "sistema de alerta" de la empresa ningún subsistema será capaz de detectar los robos, y no porque éstos no funcionen, sino porque muy probablemente los ladrones son los que controlan el "sistema de alerta" de la empresa y el dinero que está en juego es tanto que hay suficiente para corromper a quien sea necesario y acallar las alertas de todos los subsistemas que se implanten.

Cuando se pretende recuperar un proyecto hay que verificar que los requerimientos estén claramente definidos, podría pensarse que un sistema se especifica correctamente mediante casos de uso, sin embargo esto no siempre es cierto. Los proyectos de software pueden agruparse en varios tipos, [Wiegers, 2006] los clasifica en: aplicaciones interactivas para el usuario final, aplicaciones computacionalmente intensivas, almacenamiento de datos, procesamiento por lotes y productos hardware con software embebido. Los proyectos de aplicaciones interactivas para usuarios, incluyendo aplicaciones web y kioscos de ventas, son los que pueden especificarse, casi siempre, mediante casos de uso. Uno de los aspectos que hay que atacar al principio de la recuperación de un proyecto es revisar la Especificación de los Requerimientos de Software (ERS). Hay que tener en cuenta que no todas las funcionalidades se ajustan adecuadamente a los casos de uso, sobre todo si se trata de un proyecto diferente a una aplicación interactiva para el usuario. Si en el proyecto a recuperar se elaboró una ERS con casos de uso artificiales, lo más probable es que los

diseñadores nunca tuvieron claros esos requerimientos. Para mejorar y clarificar estos requerimientos [Wiegers, 2006] sugiere utilizar otro tipo de herramientas, tales como requerimientos funcionales complementarios a los casos de uso, tablas de evento-respuesta, diagramas de transición de estados y diagramas de secuencias. Otro aspecto que es necesario verificar durante la recuperación, es el protocolo de pruebas, hay que revisar que en el diseño estén contemplados suficientemente los casos no exitosos. Probar que todo funcione correctamente bajo las condiciones apropiadas no es suficiente, se necesita diseñar el sistema para que éste sea robusto y contar con pruebas que permitan verificar que efectivamente se logró esta robustez.

A lo largo de estas notas, se han ido mencionando los *errores clásicos* identificados por McConnell en su libro *Desarrollo y gestión de proyectos informáticos*, el cual obtuvo el premio JOLT 1996 al mejor libro para desarrolladores de software. Este trabajo se ha convertido en un "clásico" en la Administración de Proyectos y vale la pena aprovechar la sabiduría vertida en él. Utilizar la lista de errores clásicos a manera de *check-list* puede ser una ayuda invaluable para detectar cuales son los problemas del proyecto que se pretende recuperar.

Actualmente existen herramientas de diagnóstico tales como Project snapshot (http://www.projectsnapshot.com/) que sirven para evaluar proyectos de software y tener una idea de su viabilidad. La *fotografía del proyecto* (Project snapshot) reúne información que ilustra el estado de un proyecto. En la *figura 8.1* se ilustra la *fotografía del proyecto de software*, éste formato fue elaborado en la universidad de Princeton.

Fotografía del proyecto
Nombre del proyecto:_____

<table>
<tr><td colspan="2">Propósito del proyecto
¿Cuáles son las metas y los objetivos del proyecto? ¿Por qué se llevará a cabo? ¿Cuál es el problema a resolver o la oportunidad a aprovechar?</td></tr>
<tr><td>Entregables y tiempos de entrega (qué y cuándo)</td><td>Stakeholders (Responsable del proyecto, Administrador del proyecto, Clientes, otros grupos clave que pueden impactar o ser impactados por el proyecto)</td></tr>
<tr><td>Recursos (Dinero, personas, equipo, instalaciones, software, etc.)</td><td>Riesgos (limitación de recursos, fechas límite, presupuesto, tecnología, otras limitantes, etc.)</td></tr>
<tr><td>Interdependencias (con otros proyectos, grupos, interfaces del sistema, etc.)</td><td>Criterios de éxito (¿Cómo se puede saber si el proyecto es exitoso?)</td></tr>
</table>

Figura 8.1: fotografía del proyecto (Project snapshot) [web.princeton.edu/sites/ppo/ProjectSnapshot.doc]

Cuando se puede adquirir la "fotografía" de un proyecto, es porque se ha logrado obtener la información clave que determina sus características principales y sus probabilidades de éxito. Esta información también sirve de guía para elaborar la propuesta de acciones correctivas.

VIII.3.3 Planteamiento de acciones correctivas

Los elementos fundamentales que determinan la excelencia de un proyecto son: nivel de compromiso de los involucrados, equipo de alto desempeño, buena estimación, buena planificación y buen control. Es necesario tomar muy en cuenta cada uno de estos elementos al proponer acciones correctivas. McConnell propone una especie de lluvia de ideas, en la que cada uno de los desarrolladores identifica problemas del proyecto y propone ideas prácticas para salvar al proyecto, en base a esta información, el administrador elabora una lista de acciones correctivas, y además considera los siguientes aspectos:

- Subir la moral del personal.- Dándole tiempo para descansar. Aunque esto parezca ir en contra de la necesidad de dar velocidad al proyecto, hay que tomar en cuenta que el personal descansado tendrá un mejor desempeño. Es un error estratégico grave intentar recuperar un proyecto forzando al personal a trabajar horas extras.
- Estar dispuesto a cambiar el rumbo por completo, de ser necesario. Einstein decía: "La locura es hacer siempre lo mismo y esperar resultados diferentes".
- McConnell dice "Las partes arruinadas de un proyecto están así porque se ha ignorado consciente o inconscientemente los fundamentos del software". Hay que detectar estas partes y rehacerlas.

Existen una serie de medidas correctivas que pueden aplicarse, como por ejemplo establecer: un control de versiones, un sistema de seguimiento de defectos, un control de cambios o un análisis de riesgos. Todas estas acciones deben aplicarse según el caso a tratar. Sería muy difícil que un solo proyecto exhibiera a la vez todas estas carencias. También será necesario estabilizar los requerimientos cuando han ido cambiando demasiado a lo largo del desarrollo del proyecto. Cuando el cliente está dispuesto a recuperar un proyecto, normalmente significa que está dispuesto a negociar y posponer algunas funcionalidades. Si algún módulo está elaborado con tan baja calidad que cada vez que se arreglan sus defectos surgen otros, es más conveniente eliminarlo y comenzar de nuevo que seguir invirtiendo una cantidad de esfuerzo desproporcionada en éste.

McConnell propone la *creación de hitos miniatura detallados* con el fin de controlar el progreso del proyecto con precisión. Los hitos miniatura son pequeños y deben poder terminarse en uno o dos días. Uno de los principales problemas para establecer los hitos miniatura al comienzo de un proyecto es que no se tiene conocimiento suficiente para identificar todo el trabajo con detalle, sin embargo, cuando se trata de recuperar un proyecto, los desarrolladores ya conocen lo suficiente el producto como para poder decir en detalle lo que hay que hacer. Una vez establecidos los hitos miniatura, es responsabilidad

del administrador del proyecto controlar que todos se cumplan al 100% día a día, no debe quedar nada pendiente o por refinar, es decir, cada hito terminado debe estar listo para entregarse. Si algún desarrollador no está cumpliendo con sus hitos satisfactoriamente, será necesario reacomodar el plan tomando en cuenta su ritmo. McConnell también recomienda elaborar una lista de los diez riesgos principales y comprobarla a diario en una reunión pequeña en la que: se revisen los riesgos, se incluyan los nuevos aspectos que vayan surgiendo y se tomen decisiones a tiempo.

Además del reforzamiento de la gestión de riesgos, la implementación de medidas de recuperación de un proyecto debe incluir el apego a las normas de aseguramiento de la calidad, como los *checkpoints* de calidad y las inspecciones de la implementación.

Finalmente, recordar que el tiempo y el esfuerzo perdido no se pueden recuperar, solo podemos cumplir con los plazos reduciendo el alcance del proyecto.

Hay muy pocos trabajos en la actualidad relacionados con la recuperación de proyectos, además de la propuesta de McConnell, Soto y González [Soto y González, 2010] proponen una metodología para la recuperación de proyectos de software, la cual se expone en la siguiente sección.

VIII.4 Propuesta de un plan para la recuperación de proyectos de software.

El fundamento teórico de la propuesta metodológica que aquí se presenta es aquel que caracteriza el proceso de diagnóstico y tratamiento médico: identificación de signos y síntomas, interrogatorio, pruebas complementarias, tratamiento y seguimiento. En este sentido, vale la pena iniciar esta sección refiriéndonos a este estilo de solución de problemas y en particular a algunos de los términos que lo caracterizan.

Aquí se presentan los aspectos teórico-metodológicos del plan de recuperación propuesto, el mismo resultará en un ambiente de trabajo integrado (IDE, de las siglas en inglés) constituido por listas de chequeo, métodos, modelos, artefactos y herramientas que permitan al usuario:

i) La identificación de los signos de abandono del proyecto.
ii) La determinación de las causas de abandono del proyecto.
iii) El diagnóstico integral del proyecto de desarrollo de software.
iv) Seleccionar la estrategia de recuperación del proyecto en base al diagnóstico integral.
v) Seguir los métodos correspondientes a la ejecución de la estrategia de recuperación seleccionada.
vi) Efectuar el control y monitoreo de la ejecución de la estrategia de recuperación.

VIII.4.1 El diagnóstico y tratamiento médicos como marco de referencia

El diagnóstico médico es el proceso encaminado a la identificación o reconocimiento de una enfermedad sobre la base de los signos y síntomas que se manifiestan, y con el apoyo de los estudios de laboratorio y gabinete. El diagnóstico médico indica todo el proceso de investigación ejecutado sobre el paciente, a partir de las observaciones y razonamientos del médico para determinar la enfermedad.

Un tipo particular de diagnóstico médico es el diagnóstico diferencial, el cual está íntimamente relacionado con la precisión diagnóstica y por lo tanto, con la capacidad para poder agrupar enfermedades relacionadas ya sea como entidades nosológicas o síndromes. La capacidad diagnóstica de un buen médico está relacionada directamente con su experiencia para poder distinguir todas las posibilidades diagnósticas para un cierto grupo de manifestaciones clínicas (signos, síntomas, laboratorio y gabinete) y su capacidad para discernir, sobre la base de su experiencia, cual de todos estos padecimientos es con mayor certeza la causa del proceso mórbido. La meta del diagnóstico diferencial es "arribar" a un diagnóstico específico, sobre la base de la exclusión razonada de otras enfermedades que tienen en común gran parte de los síntomas.

Una vez expuestas estas consideraciones sobre el diagnóstico médico, y en particular sobre el diagnóstico diferencial, resulta muy apropiado considerar que un plan para la recuperación de proyectos de software pueda establecerse sobre la base de un proceso similar al que caracteriza a estos procedimientos médicos. Como se verá en la siguiente sección, un primer paso para lograr lo anterior será proporcionar una semántica en el dominio de la ingeniería del software a elementos de la terminología médica, que resultan parte indisoluble del diagnóstico médico, y que por lo tanto serán imprescindibles en el plan de recuperación, tales como signo, síntoma, resultados de laboratorio y gabinete, causa, síndrome, etc.

VIII.4.2 Fases, actividades y artefactos del plan de recuperación

El plan de recuperación propuesto abarca seis fases, las se muestran en la *figura 8.2*. En la *figura 8.3* se utiliza un diagrama orientado a objetos para representar los diferentes elementos del plan de recuperación y la relación entre los mismos. Como se puede apreciar en esta figura, una fase puede corresponder a una única actividad (homónima) o puede englobar varias actividades. Para su ejecución, las actividades se basan en la construcción y/o evaluación de diferentes artefactos definidos por la propia propuesta metodológica tales como tablas, listas de chequeo, cuestionarios, reglas de inferencia, etc. Finalmente, en la *figura 8.4* se presenta a través de un diagrama de casos de uso la interacción del usuario (ingeniero de software, líder de proyecto, consultor, etc.) con las diferentes actividades que tienen lugar cuando el plan de recuperación ha aplicado. En las próximas secciones se describen las cuatro primeras fases de plan de recuperación.

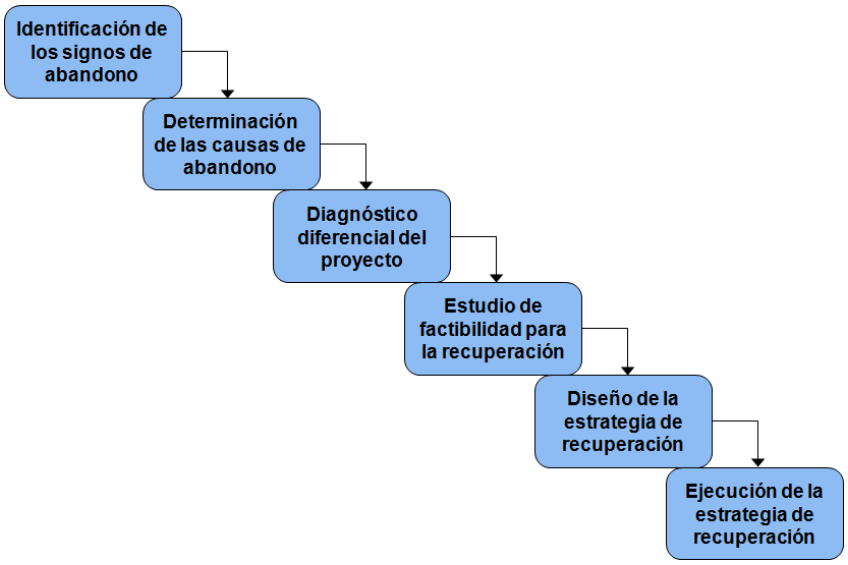


Figura 8.2: Las fases del plan de recuperación de proyectos de software.

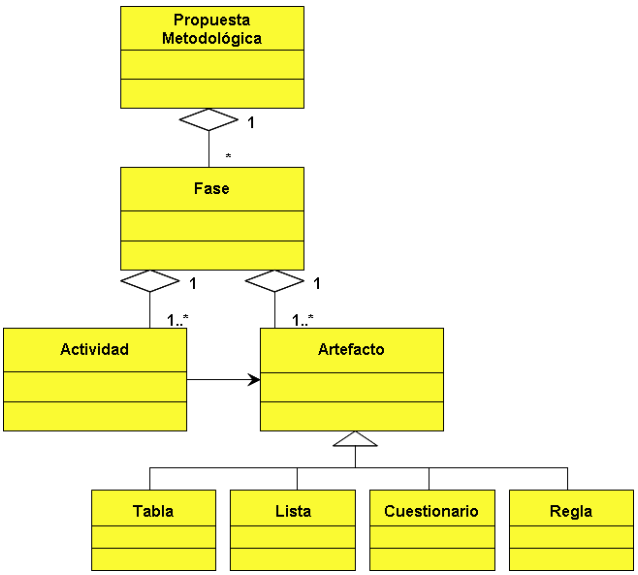


Figura 8.3: Fases, actividades y artefactos del plan de recuperación de proyectos de software.

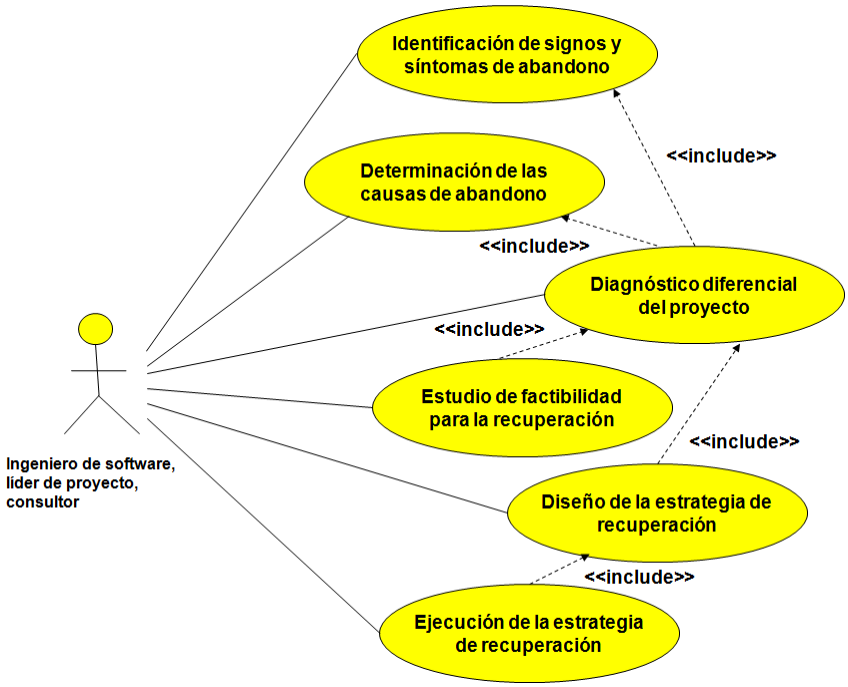


Figura 8.4: Diagrama de casos de uso que muestra la interacción del usuario con las diferentes actividades a ejecutar durante el seguimiento del plan de recuperación.

VIII.4.3 Identificación de los signos y síntomas de abandono del proyecto

Los proyectos de desarrollo de software con un alto riesgo de abandono, o de hecho ya abandonados, comúnmente exhiben una combinación de signos y síntomas negativos (o no deseables) agrupados en un conjunto de categorías que corresponden a las principales fases y elementos claves involucrados en el proceso de desarrollo de software. Como se muestra en la *figura 8.5*, las principales fuentes para la identificación de dicha sintomatología de abandono del proyecto son las siguientes:

- Documento de especificación del proceso de desarrollo (el cual, entre otra documentación, debe incluir planeación y estimación, análisis de requerimiento, diseño global, diseño de componentes, documento de implementación, etc.).
- Cuestionarios, entrevistas u otras fuentes que plasmen los puntos de vista y opiniones de los integrantes del equipo de desarrollo, líder de proyecto, directiva, etc.
- Dominio de referencia de los principales signos y síntomas que caracterizan un proyecto de software con problemas.

Para la identificación de tales signos y síntomas de abandono, se recorrerá el dominio de referencia propuesto en la *tabla 8.1*, constatando por cada categoría cuales de los signos y síntomas listados se manifiestan o caracterizan el proyecto de desarrollo de software.

Tabla 8.1. Categorías de signos y síntomas de abandono de un proyecto de software

Categoría	Signos y síntomas
Especificación del alcance del proyecto	SS-EA01: No se comprende claramente el alcance del sistema. SS-EA02: No se logra identificar con claridad los productos resultantes del proyecto. SS-EA03: Ni los integrantes del equipo de desarrollo ni el cliente logran especificar de forma apropiada el área de aplicación. SS-EA04: La relación alcance, tiempo y precio, está completamente del lado del cliente, dejando poco o nulo margen a la negociación del producto, estimación y planeación. SS-EA05: Documentación carente de un análisis de riesgos.
Estimación y planeación del proyecto	SS-EP01: No se cumplen las entregas en las fechas acordadas. SS-EP02: Retraso significativo en el desarrollo. SS-EP03: Percepción de un desarrollo lento. SS-EP04: Producto de baja calidad. SS-EP05: El desarrollo del proyecto no se adapta a la planeación y recursos disponibles. SS-EP06: Se trabaja bajo la presión de la fecha límite. SS-EP07: Deterioro de las relaciones entre desarrolladores, directivos, cliente, marketing y otros participantes. SS-EP08: No se están generando fechas de entrega reales. SS-EP09: El cliente ha perdido la confianza en lo que puede esperar del sistema y de quién lo provee. SS-EP10: Se percibe un ambiente de incertidumbre en el equipo y el seguimiento que se le puede dar al mismo es cada vez más complicado. SS-EP11: Se ha perdido la noción de avance, y es difícil o imposible identificar el estatus real del proyecto. SS-EP12: Los riesgos identificados han sido minimizados y no son considerados al llevar a cabo la planeación, o peor aún no se les ha identificado.
Ciclo de vida y metodología de desarrollo	SS-CV01: Imposibilidad de mostrarle al cliente signos evidentes de progreso. SS-CV02: Imposibilidad de mostrar a la directiva signos visibles de progreso. SS-CV03: Dificultad para efectuar modificaciones durante el proceso de desarrollo. SS-CV04: Imposibilidad de liberar al cliente versiones del producto para refinar éste a partir de sus comentarios. SS-CV05: Dificultad para centrar el desarrollo en los aspectos visibles del sistema y mostrar así avances al cliente. SS-CV06: Imposibilidad de liberar el sistema por etapas sucesivas a lo largo del proyecto. SS-CV07: Impresión de un desarrollo lento. SS-CV08: Incremento de los riesgos durante el proceso de desarrollo y ninguna gestión de los mismos.

Comunicación con la directiva y con el cliente	SS-CD01: Percepción de un desarrollo lento por parte del cliente. SS-CD02: Percepción de un desarrollo lento por parte de la directiva. SS-CD03: No existen los medios (artefactos que soporten y justifiquen el alcance) por los cuales la dirección del equipo de desarrollo y la del cliente puedan llegar a un acuerdo en cuanto a los tiempos de entrega y el contenido de los mismas. Esta situación genera un ambiente de desconfianza y malestar.
Equipo de desarrollo	SS-ED01: El equipo de desarrollo ha perdido la moral, lo que conlleva a entregas plagadas de errores. SS-ED02: El personal a cargo del equipo de desarrollo no tiene los medios ni la capacidad para exigir entregables con calidad. SS-ED03: El equipo de desarrollo está llevando jornadas de trabajo de más de 50 horas a la semana. SS-ED04:Las relaciones personales entre los miembros del equipo se han deteriorado. SS-ED05: Se ha perdido la confianza en el proyecto que se está desarrollando y el ambiente se ha vuelto de frustración y conformismo.
Arquitectura y componentes del sistema	SS-AC01: La arquitectura no ha resultado ser lo suficientemente robusta. SS-AC02: La arquitectura no cumple con los requerimientos del cliente, lo que ha forzado a estarla modificando en etapas bastante avanzadas del proyecto. SS-AC03: La serie de cambios efectuados ha creado una arquitectura completamente distinta a la que se tenía diseñada en un inicio. SS-AC04: Las vistas de la arquitectura no se pueden interpolar. SS-AC05: El equipo de desarrollo y en general los involucrados desconocen la arquitectura del sistema. SS-AC06: Los componentes que se han construido (llámense subsistemas o módulos) presentan problemas de integración. SS-AC07: La arquitectura no se comprende.
Uso de tecnologías, herramientas y lenguajes	SS-TH01: El lenguaje elegido para el desarrollo se ha hecho muy difícil de comprender para el equipo de trabajo. SS-TH02: Las pruebas unitarias están denotando un gran margen de error. SS-TH03: La tecnología usada no permite llevar a cabo la implementación de los requerimientos del usuario. SS-TH04: Las herramientas elegidas están creando más conflictos que los problemas que resuelven.
Control de la calidad	SS-CC01: Inexistente. SS-CC02: Los artefactos y demás entregables no son consistentes de versión a versión (se repiten errores y/o no hay formatos bases). SS-CC03: La especificación de la calidad sólo es seguida por una de las partes. SS-CC04: Resulta imposible exigir al equipo de desarrollo codificar de forma estándar, siguiendo un esquema de nombrado, documentación, etc. SS-CC05: Expresar ante el cliente los estándares que aseguran la

	calidad del proyecto desarrollado se vuelve una tarea difusa y complicada.

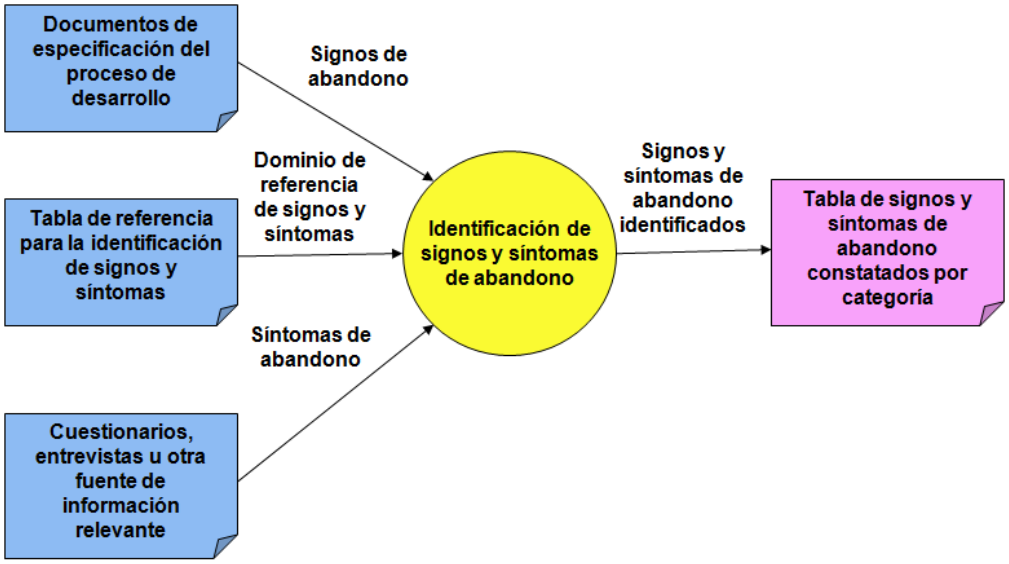


Figura 8.5: Diagrama de flujo de datos correspondiente al proceso de identificación de signos y síntomas de abandono.

VIII.4.4 Determinación de las causas de abandono del proyecto

Una vez identificados y clasificados los signos y síntomas de abandono del proyecto, y de ser necesario complementados con otras pruebas efectuadas, al igual que en el proceso de diagnóstico médico, la próxima fase del plan de recuperación se centra en la determinación de las causas que han originado esta sintomatología. Estas causas de abandono corresponden en el diagnóstico médico a la fase del diagnóstico etiológico, es decir, el estudio de las causas de la enfermedad.

Como se muestra en la *figura 8.6*, las principales fuentes para la identificación de las causas de abandono del proyecto son las siguientes:

- Documento de especificación del proceso de desarrollo (el cual, entre otra documentación, debe incluir planeación y estimación, análisis de requerimiento, diseño global, diseño de componentes, documento de implementación, etc.).
- Cuestionarios, entrevistas u otras fuentes que plasmen los puntos de vista y opiniones de los integrantes del equipo de desarrollo, líder de proyecto, directiva, etc.

- Tabla de signos y signos de abandono constatados por categoría (se obtuvo como resultado de la fase de identificación de signos y síntomas).
- Tabla de referencia para la asociación de causas probables a los signos y síntomas identificados.

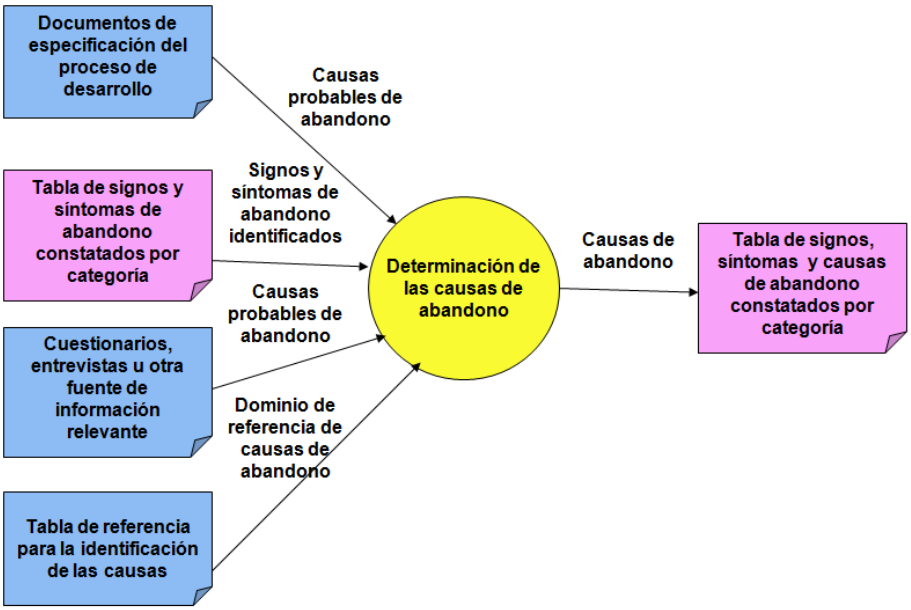


Figura 8.6: Diagrama de flujo de datos correspondiente al proceso de identificación de las causas de abandono.

Para la identificación de las posibles causas de abandono que originan la manifestación de los signos y síntomas previamente identificados, se recorrerá el dominio de referencia propuesto en la *tabla 8.2* para constatar cuales, de entre todas las causas probables asociadas a cada grupo de signos y síntomas, constituyen realmente la fuente que los ocasiona. Esta constatación de relación causal se sustenta en hechos y acontecimientos referidos por fuentes tales como:

- Documentos de especificación del proceso de desarrollo.
- Pruebas efectuadas sobre los diferentes productos del proceso de desarrollo (documento de planeación y estimación, especificación de análisis, especificación de diseño, programas, etc.)
- Resultados de las entrevistas a los miembros del equipo de desarrollo.

Tabla 8.2. Principales causas de abandono de un proyecto de software		
Categoría	**Signos y síntomas**	**Causas probables**
Especificación del alcance del	SS-EA01: No se comprende claramente el alcance del	CP-EA01: Pobre especificación de los requerimientos.

proyecto	sistema. SS-EA02: No se logra identificar con claridad los productos resultantes del proyecto. SS-EA03: Ni los integrantes del equipo de desarrollo ni el cliente logran especificar de forma apropiada el área de aplicación. SS-EA04: La relación alcance, tiempo y precio, está completamente del lado del cliente, dejando poco o nulo margen a la negociación del producto, estimación y planeación. SS-EA05: Documentación carente de un análisis de riesgos.	CP-EA02: No se ha definido con claridad el alcance del sistema. CP-EA03: No se cuenta con una definición estable del producto software a desarrollar. CP-EA04: Gran rapidez en el cambio de los requerimientos y/o poca estabilización de los mismos. CP-EA05: Negociación deficiente con poco énfasis en los requerimientos y sin noción de alcance. El estudio de la viabilidad del proyecto ha fallado. Puede no existir. CP-EA06: La metodología escogida no es la correcta, o se está omitiendo el/los artefacto/s que contemplan el análisis de riesgos. El líder de proyecto no tiene la capacidad para incluir un análisis del tipo.
Estimación y planeación del proyecto	SS-EP01: No se cumplen las entregas en las fechas acordadas. SS-EP02: Retraso significativo en el desarrollo. SS-EP03: Percepción de un desarrollo lento. SS-EP04: Producto de baja calidad. SS-EP05: El desarrollo del proyecto no se adapta a la planeación y recursos disponibles. SS-EP06: Se trabaja bajo la presión de la fecha límite. SS-EP07: Deterioro de las relaciones entre desarrolladores, directivos, cliente, marketing y otros participantes. SS-EP08: No se están generando fechas de entrega reales. SS-EP09: El cliente ha perdido la confianza en lo que puede esperar del sistema y de quién lo provee.	CP-EP01: Planeación excesivamente optimista. CP-EP02: Estimaciones poco realistas. CP-EP03: Estimación de recursos y planeación iniciales inaceptables. CP-EP04: Desequilibrio entre planeación, costo y producto. CP-EP05: Se ha subestimado la complejidad y alcance del proyecto. CP-EP06: Incremento de nuevos requerimientos en el proyecto. CP-EP07: La comunicación entre los diferentes involucrados en el proyecto es deficiente. CP-EP08: Los artefactos que se están usando en el control del proyecto no están aportando lo necesario, o se está fallando en su uso. CP-EP09: El equipo no está siendo retroalimentado y/o el seguimiento que se les da no es el suficiente.

	SS-EP10: Se percibe un ambiente de incertidumbre en el equipo y el seguimiento que se le puede dar al mismo es cada vez más complicado. SS-EP11: Se ha perdido la noción de avance, y es difícil o imposible identificar el estatus real del proyecto. SS-EP12: Los riesgos identificados han sido minimizados y no son considerados al llevar a cabo la planeación, o peor aún no se les ha identificado.	CP-EP10: Los hitos no son alcanzables o se están moviendo continuamente. CP-EP11: No existe un plan de calidad o no es lo suficientemente robusto. CP-EP12: El control de riesgos no está siendo atendido a cabalidad. CP-EP13: No existe documentación que soporte los riesgos del proyecto. CP-EP14: Estimación y planeación inexistentes.
Ciclo de vida y metodología de desarrollo	SS-CV01: Imposibilidad de mostrarle al cliente signos evidentes de progreso. SS-CV02: Imposibilidad de mostrar a la directiva signos visibles de progreso. SS-CV03: Dificultad para efectuar modificaciones durante el proceso de desarrollo. SS-CV04: Imposibilidad de liberar al cliente versiones del producto para refinar éste a partir de sus comentarios. SS-CV05: Dificultad para centrar el desarrollo en los aspectos visibles del sistema y mostrar así avances al cliente. SS-CV06: Imposibilidad de liberar el sistema por etapas sucesivas a lo largo del proyecto. SS-CV07: Impresión de un desarrollo lento. SS-CV08: Incremento de los riesgos durante el proceso de desarrollo y ninguna gestión de los mismos.	CP-CV01: No se ha seleccionado un ciclo de vida y/o metodología adecuados a las características y restricciones del proyecto (por ejemplo, entrega del producto por etapas, liberación de versiones, signos visibles de desarrollo, etc.). CP-CV02: El desarrollo del proyecto no se ha estructurado de forma tal que se pueda apreciar claramente su progreso. CP-CV03: La metodología elegida no aporta los elementos de control de riesgos necesarios (Por ejemplo cascada pura, entrega por etapas, etc.). CP-CV04: El ciclo de vida y/o la metodología usada es desconocida por el equipo de desarrollo, lo que está derivando en una mala implementación de la misma.
Comunicación con la directiva y con el cliente	SS-CD01: Percepción de un desarrollo lento por parte del cliente. SS-CD02: Percepción de un desarrollo lento por parte de la directiva. SS-CD03: No existen los medios (artefactos que soporten y	CP-CD01: Cliente no involucrado en el desarrollo del proyecto. CP-CD02: Directiva no involucrada en el desarrollo del proyecto. CP-CD03: No existen adecuados canales de comunicación/colaboración con el cliente y con la

	justifiquen el alcance) por los cuales la dirección del equipo de desarrollo y la del cliente puedan llegar a un acuerdo en cuanto a los tiempos de entrega y el contenido de los mismas. Esta situación genera un ambiente de desconfianza y malestar.	directiva. CP-CD04: La metodología no incluye artefactos en los cuales se soporten la toma de decisiones y puedan servir como base para la definición del alcance del proyecto. CP-CD05: Los artefactos dedicados a la comunicación, alcance y soporte de calidad, no están siendo usados debidamente (Project charter, Visión, Especificación complementaria, etc.).
Equipo de desarrollo	SS-ED01: El equipo de desarrollo ha perdido la moral, lo que conlleva a entregas plagadas de errores. SS-ED02: El personal a cargo del equipo de desarrollo no tiene los medios ni la capacidad para exigir entregables con calidad. SS-ED03: El equipo de desarrollo está llevando jornadas de trabajo de más de 50 horas a la semana. SS-ED04:Las relaciones personales entre los miembros del equipo se han deteriorado. SS-ED05: Se ha perdido la confianza en el proyecto que se está desarrollando y el ambiente se ha vuelto de frustración y conformismo.	CP-ED01: La planeación del proyecto ha sido defectuosa y no existe motivación en el equipo de desarrollo. CP-ED02: Los hitos y asignaciones no están siendo claras para el grupo. CP-ED03: No se tiene un buen nivel técnico en el equipo. CP-ED04: La responsabilidad de alcanzar los objetivos está recayendo sólo en unos cuantos, se debe revisar la tabla de responsabilidades CP-ED05: No ha habido un correcto liderazgo CP-ED06: No se está reconociendo el esfuerzo del equipo. CP-ED07: Los recursos humanos son insuficientes. CP-ED08: La imagen del líder o del personal a cargo del proyecto es negativa y afecta al grupo.
Arquitectura y componentes del sistema	SS-AC01: La arquitectura no ha resultado lo suficientemente robusta. SS-AC02: La arquitectura no cumple con los requerimientos del cliente, lo que ha forzado a estarla modificando en etapas bastante avanzadas del proyecto. SS-AC03: La serie de cambios efectuados ha creado una arquitectura completamente	CP-AC01: Falta de entendimiento de los requerimientos de alto nivel. CP-AC02: No se identificaron los atributos de calidad. CP-AC03: Pobre diseño de alto nivel. CP-AC04: La arquitectura escogida no es la correcta. CP-AC05: No hay un correcto control de las versiones. CP-AC06: Las vistas de diseño no

	distinta a la que se tenía diseñada en un inicio. SS-AC04: Las vistas de la arquitectura no se pueden interpolar. SS-AC05: El equipo de desarrollo y en general los involucrados desconocen la arquitectura del sistema. SS-AC06: Los componentes que se han construido (llámense subsistemas o módulos) presentan problemas de integración. SS-AC07: La arquitectura no se comprende.	están aportando la información suficiente. CP-AC07: Falta de experiencia del arquitecto. CP-AC08: La documentación no está siendo actualizada, lo que genera que lo que se ve en una vista en particular, no se refleja en otra. CP-AC09: Se ha errado en el diseño detallado. CP-AC10: Se omitió integrar en los artefactos las vistas de interacción. CP-AC11: El equipo encargado de diseñar no tiene la suficiente experiencia para poder plasmar un diseño de calidad.
Uso de tecnologías, herramientas y lenguajes	SS-TH01: El lenguaje elegido para el desarrollo se ha hecho muy difícil de comprender para el equipo de trabajo. SS-TH02: Las pruebas unitarias están denotando un gran margen de error. SS-TH03: La tecnología usada no permite llevar a cabo la implementación de los requerimientos del usuario. SS-TH04: Las herramientas elegidas están creando más conflictos que los problemas que resuelven.	CP-TH01: Desconocimiento de la tecnología con la que se está trabajando. CP-TH02: No se tiene el nivel técnico necesario en el equipo de desarrollo, y la curva de aprendizaje no se está atacando con miras a hacerla lo más corta posible. CP-TH03: Mala elección del lenguaje y en general de la tecnología a usar. CP-TH04: Tecnología obsoleta. CP-TH05: Problemas de licenciamiento
Control de la calidad	SS-CC01: Inexistente. SS-CC02: Los artefactos y demás entregables no son consistentes de versión a versión (se repiten errores y/o no hay formatos bases). SS-CC03: La especificación de la calidad sólo es seguida por una de las partes. SS-CC04: Resulta imposible exigir al equipo de desarrollo codificar de forma estándar, siguiendo un esquema de nombrado, documentación, etc. SS-CC05: Expresar ante el cliente los estándares que aseguran la calidad del proyecto	CP-CC01: No existe la cultura en el equipo de desarrollo que pueda llevar a desarrollar documentación orientada a la calidad. CP-CC02: No se negoció ningún punto que asegure la calidad. CP-CC03: El equipo de desarrollo y el cliente no llegan a un consenso mediante el cual la calidad es cuantificada y calificada. CP-CC04: El personal que está a cargo de la calidad del proyecto es inexperto y no ha sido correctamente inducido en el tema. CP-CC05: No se han destinado

	desarrollado se vuelve una tarea difusa y complicada.	esfuerzos en busca de proporcionar la calidad al proyecto (por ejemplo, Talleres QAW, etc.).

VIII.4.5 Diagnóstico diferencial del proyecto de desarrollo de software

Al igual que en el proceso de diagnóstico médico, la robustez diagnóstica de este plan de recuperación está directamente relacionada con la capacidad del mismo para identificar todas las posibles causas de abandono que pudieran estar asociadas a un cierto grupo de signos y síntomas previamente identificados y su capacidad para diferenciar, sobre la base de toda la actual información recopilada, así como de la experiencia codificada en su base de conocimientos, cual de todas estas causas es la que más impacta en el incumplimiento o abandono del proyecto. La meta de esta fase del plan de recuperación es alcanzar un diagnóstico específico, sobre la base de la exclusión deducida de otras causas o problemas que comparten gran parte de los signos y síntomas identificados.

El diagnóstico diferencial, como parte del plan de recuperación, realmente inicia con la individualización de las causas probables que originaron la manifestación de los grupos de signos y síntomas previamente identificados (ver *figura 8.5*). Como se puede apreciar en la *figura 8.7*, el segundo paso de este proceso de diagnóstico diferencial se nutre precisamente de esta valiosa información recabada en la fase previa del plan y la pericia codificada en forma de reglas de producción en una base de conocimientos que prevé el plan de recuperación. En la *figura 8.8* se puede apreciar dos ejemplos de estas reglas de producción escritas en el lenguaje declarativo Prolog.

Figura 8.7: Diagrama de flujo de datos correspondiente al proceso de identificación de las causas de abandono.

```
diagnóstico("Problemas con la Especificación del Alcance", CERT) :-
            (signo(SS-EA01);signo(SS-EA02);
             signo(SS-EA03);signo(SS-EA04);
             signo(SS-EA05)),
            (causa(CP-EA01);causa(CP-EA02);
             causa(CP-EA03);causa(CP-EA04);
             causa(CP-EA05);causa(CP-EA06)).

diagnóstico("Problemas con la Estimación y Planeación", CERT) :-
            (signo(SS-EP01);signo(SS-EP02);signo(SS-EP03);
             signo(SS-EP04);signo(SS-EP05);signo(SS-EP06);
             signo(SS-EP07);signo(SS-EP07);signo(SS-EP08);
             signo(SS-EP09);signo(SS-EP10);
             signo(SS-EP11);signo(SS-EP12)),
            (causa(CP-EP01);causa(CP-EP02);causa(CP-EP03);
             causa(CP-EP04);causa(CP-EP05);causa(CP-EP06);
              causa(CP-EP07);causa(CP-EP08);causa(CP-EP09);
              causa(CP-EP10);causa(CP-EP11);causa(CP-EP12);
              causa(CP-EP13);causa(CP-EP14)).
```

Figura 8.8. Reglas de producción contenidas en la base de conocimiento que prevé el plan de recuperación. Las reglas son escritas en el lenguaje declarativo Prolog e invocadas durante la fase de Diagnóstico diferencial.

VIII.4.6 Estudio de factibilidad para la recuperación

Una vez efectuado el diagnóstico diferencial del proyecto y por lo tanto, identificado el problema o problemas que afectan el mismo, la próxima fase del plan de recuperación aborda el estudio de factibilidad para la recuperación del proyecto. La esencia de esta fase consiste en hacer del conocimiento de ambas directivas, la del cliente y la del equipo de desarrollo, el problema o problemas diagnosticados y decidir conjuntamente si emprender o no el plan de recuperación. La *figura 8.9* muestra el diagrama de flujo de datos correspondiente a esta fase del plan de recuperación.

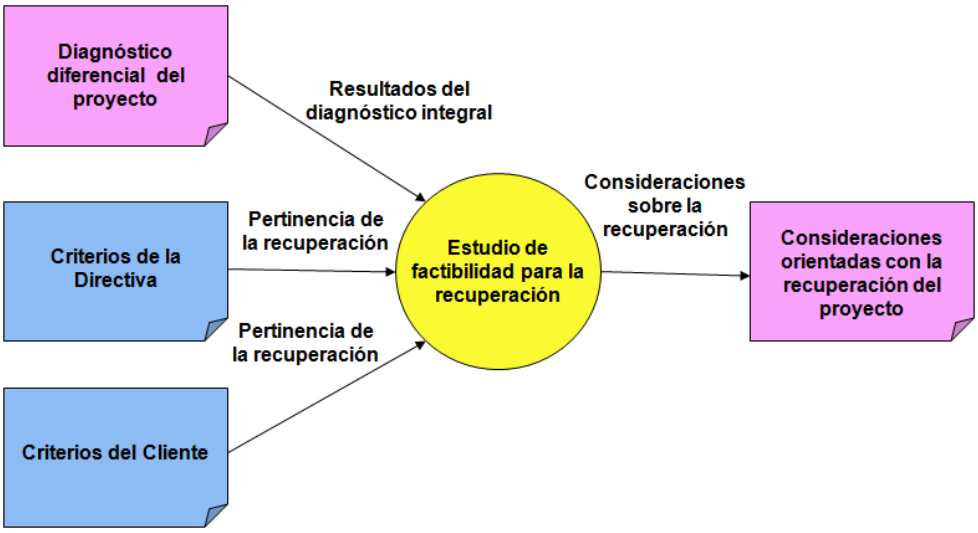


Figura 8.9: Diagrama de flujo de datos correspondiente al proceso de estudio de factibilidad para la recuperación.

VIII.4.7 Escenarios de Proyectos con Problemas y un Caso de Estudio

A continuación se expondrán de manera breve dos escenarios y un caso de estudio de proyectos de desarrollo de software, que en menor o mayor medida, son considerados como proyectos fallidos. Recordemos que para que un proyecto sea no exitoso, basta con exceder los tiempos estimados para su realización y/o incrementar los recursos que se tenían destinados para la construcción del mismo.

Escenarios

a) A mediados del 2003, una de las firmas más importantes dedicadas al autotransporte de pasajeros en el país, comenzó el desarrollo de lo que sería un ERP (del inglés *Enterprise Resource Planning*) a la medida, el cual buscaba cubrir e integrar las diferentes áreas que conforman la empresa, logrando con ello una mejor explotación de los recursos.
   En el desarrollo del proyecto se integraron diferentes consultorías tanto nacionales como internacionales. Aunque dentro de los recursos humanos se tenía un arquitecto en jefe, cuya responsabilidad sería mantener homogéneo el desarrollo del sistema y apegado a estándares, la realidad fue que en determinado momento se dejó de dar seguimiento, y los estándares terminaron por pasar a un segundo plano. Esta situación derivó en pobres esquemas de control y por último en una desviación bastante elevada de lo que debería ser el proyecto en sí.

Actualmente el ERP ya se encuentra en producción, sin embargo un hecho alarmante, es que aún a la fecha actual, los subsistemas que lo conforman no han logrado la interoperabilidad que se tenía planeada y que era uno de los principales requerimientos no funcionales del sistema. Sin embargo, en la planeación inicial se consideró tenerlo en operación a finales del 2006.

Podría apreciarse este escenario como un proyecto que tuvo problemas, pero que al final se pudo retomar. Sin embargo, los tiempos a los que se tuvo que recurrir y el esfuerzo invertido pudieron ser sustancialmente menores a los reales. Cabe señalar que al no contar con un plan de recuperación y más aún, al no reconocer el estatus de proyecto con problemas no se puso en marcha una iniciativa orientada a la recuperación de proyectos.

b) En 2006 una reconocida institución financiera del país, decidió ceder a un grupo externo el desarrollo de uno de los módulos que conformarían su sistema integral de información (SII). El sistema que se debería desarrollar tendría como objetivo llevar la gestión de la parte fiduciaria, y estaría fuertemente ligada a la casa de bolsa de la institución.

   Dado que la institución ya tenía bastante adelantado el SII, impuso una serie de controles que tendrían que ser llevados a cabo por la consultoría externa y mediante la cual se valoraría el avance en el desarrollo del mismo.

   Sin embargo, los controles y demás estándares tardaron en entrar en el proyecto, más aún, no fueron considerados en las etapas de inicio y negociación. De aquí que, al tratar de irlos agregando en el transcurso de la construcción, crearon una losa muy pesada de llevar que derivó en problemas con las entregas y el desenvolvimiento del proyecto.

   Otro de los factores a considerar fue el nivel del equipo de trabajo. Al buscar ganar el proyecto, la consultoría externa ofreció precios y tiempos que no correspondían a las dimensiones del sistema (no existió el análisis de viabilidad), por lo que incurrió en formar un equipo de experiencia mínima, con lo cual parecería cubrir la parte económica, pero que dejó sin contemplar el factor tiempo y calidad.

   Tan dispareja combinación dio como resultado la desesperación y pérdida de la confianza del cliente, que a la larga decidió cancelar el proyecto, dejando una considerable pérdida económica para ambas partes (institución y consultoría externa).

   Esta situación, nos pone en perspectiva de dos grupos. Por una parte, uno con idea de lo que debe ser un desarrollo estandarizado, y al parecer controlado, y por otra parte, un grupo que aunque con buenas intenciones no supo llevar la negociación del proyecto. De esta forma, la postura de ambos grupos hizo imposible que se concretara la comunicación necesaria.

   Imaginemos el proyecto llegando a sus últimas consecuencias, donde la junta directiva del cliente decidió cancelar. Recordemos que esta decisión arrojó pérdidas a ambas partes, pues nada de lo desarrollado terminó por usarse.

¿Qué hubiera pasado si antes de cancelar el proyecto se hubiera invertido en un análisis sobre la realidad del mismo? Seguramente el resultado de éste hubiera arrojado un elemento de decisión a considerar y la pérdida hubiera sido considerablemente menor, o inclusive transformada en ganancia.

Caso de estudio

Los involucrados (Actores):

- Clínica para la Rehabilitación y Prevención de Adicciones (CRPA): Solicita un ERP con el objetivo de mejorar el negocio.
- Adrián Ramos: Encargado del área administrativa y representante por parte de CRPA ante ITe Consultores.
- ITe Consultores: Empresa dedicada al desarrollo de software a la medida.
- Leonel Hinojosa: Empleado de ITe Consultores y primer consultor a cargo del proyecto.
- Mauro Escalante: Empleado de ITe Consultores, quien tiene la responsabilidad de darle continuidad al proyecto CRPA.

Los antecedentes

CRPA se constituye en 1980, con la misión de satisfacer la demanda en atención a personas con problemas de alcoholismo y drogadicción. Desde entonces, CRPA ha contribuido en la rehabilitación de personas que sufren la enfermedad de la adicción, así como en la atención requerida a sus familiares.

Debido al crecimiento que la clínica ha ido experimentado, la directiva se encuentra interesada en una solución de software que permita la integración de sus diferentes áreas (admisión, unidad médica, unidad terapéutica, compras, contabilidad y recursos humanos), logrando con ello una mejor administración y seguimiento del negocio en general. Se ha resuelto solicitar a ITe Consultores, el desarrollo del sistema integral CRPA (SI-CRPA), que buscará cubrir todas las áreas, y que deberá ser pensado como un ERP a la medida.

Contexto del desarrollo del proyecto

Cuando a Mauro lo asignaron como líder del proyecto SI-CRPA, ya habían pasado más de tres meses desde su comienzo. La tarea prioritaria fue ponerse al corriente en cuanto al avance logrado hasta ese momento. Al estar revisando la planeación se percató, que si bien no existía un cronograma de actividades detallado, se tenía un documento con algunos entregables esbozados de manera informal y con las fechas aproximadas de lo que sería la entrega. Esta situación daba la impresión de tener un proyecto dentro de los tiempos, y que presumía gozar de buena salud. Pero, en la revisión detallada de los entregables, se notaba que lo único que el cliente había visto, eran elementos de diseño visual como *look and feel*

y algunas interfaces gráficas. Pese a todo esto, la fecha negociada para la entrega del primer módulo del sistema (módulo de Admisión) estaba muy cerca.

La consultoría había pensado que la salida de Leonel del proyecto no sería bien recibida por el cliente, y que incluso por la cercanía de las entregas, sería delicado que esto se supiera. A consecuencia de lo anterior a Mauro se le indicó que toda la retroalimentación que pudiera necesitar debería obtenerla del trabajo desarrollado por Leonel. Ahora, el problema al que se enfrentaba Mauro era que la documentación de la que se hablaba era insuficiente, y en la mayoría de los casos no existía.

Dadas las restricciones impuestas por la consultoría y que la documentación proporcionada por Leonel era insuficiente, se comenzó una etapa de entrevistas con éste, quién, como se mencionó, fue el único involucrado en el proyecto. Desafortunadamente, la transferencia de conocimiento se vio limitada a unas cuantas interfaces gráficas, derivando en una pobre comprensión de los requerimientos y una deficiente idea de lo era el negocio. Mauro externó a la dirección de la consultoría la imposibilidad de llevar a cabo la entrega en tiempo, dado que no se comprendían los requerimientos con claridad y que no se contaba con algún soporte documental donde el cliente y la consultoría pudieran plasmar los acuerdos adquiridos. Finalmente se llegó al acuerdo de retomar el análisis con el cliente, pero con la premisa de no hacerle saber el descontrol en el que se encontraba el proyecto, y para este fin, se le planteó la actividad como una etapa necesaria para corroborar lo ya analizado. Había comenzado una nueva etapa en el proyecto.

Una vez encaminados en el análisis con los involucrados en el negocio de CRPA, se llegó a la conclusión de que el alcance previsto para el proyecto, estaba muy alejado de la realidad y que por lo tanto los costos superarían con creces lo estimado. Cuestión esta que derivó en una nueva etapa de negociación donde la consultoría buscaba incrementar el precio del proyecto y el cliente esperaba al menos se le entregara lo ya pagado. Al final la negociación fracasó y no se le dio más soporte al proyecto, lo que conllevó a un abandono por ambas partes.

Hacia la recuperación del proyecto

Una vez planteado el caso de estudio, podemos identificar casi de inmediato las características propias de un proyecto en conflicto. Sin embargo, llevaremos estas deducciones al plano formal, y para ello aplicaremos el plan de recuperación aquí propuesto. Recordemos que el plan consiste en seis fases, pero en el actual caso de estudio aplicaremos sólo las cuatro primeras (Identificación de los signos y síntomas de abandono, determinación de las causas de abandono, diagnóstico diferencial y estudio de factibilidad para la recuperación).

a) Identificación de los signos y síntomas de abandono del proyecto

Basándonos en la tabla 1, y utilizando ésta a modo de lista de chequeo, identificamos la siguiente sintomatología del proyecto:

Categoría: Especificación del alcance del proyecto

SS-EA01: No se comprende claramente el alcance del sistema.
SS-EA02: No se logra identificar con claridad los productos resultantes del proyecto.
SS-EA05: Documentación carente de un análisis de riesgos.

Categoría: Estimación y planeación del proyecto

SS-EP01: No se cumplen las entregas en las fechas acordadas.
SS-EP02: Retraso significativo en el desarrollo.
SS-EP03: Percepción de un desarrollo lento.
SS-EP05: El desarrollo del proyecto no se adapta a la planeación y recursos disponibles.
SS-EP06: Se trabaja bajo la presión de la fecha límite.
SS-EP07: Deterioro de las relaciones entre desarrolladores, directivos, cliente, marketing y otros participantes.
SS-EP08: No se están generando fechas de entrega reales.
SS-EP09: El cliente ha perdido la confianza en lo que puede esperar del sistema y de quién lo provee.
SS-EP10: Se percibe un ambiente de incertidumbre en el equipo y el seguimiento que se le puede dar al mismo es cada vez más complicado.
SS-EP11: Se ha perdido la noción de avance, y es difícil o imposible identificar el estatus real del proyecto.
SS-EP12: Los riesgos identificados han sido minimizados y no han sido considerados en la planeación, o peor aún no se les ha identificado.

Categoría: Ciclo de vida y metodología de desarrollo

SS-CV01: Imposibilidad de mostrarle al cliente signos evidentes de progreso.
SS-CV02: Imposibilidad de mostrar a la directiva signos visibles de progreso.
SS-CV04: Imposibilidad de liberar al cliente versiones del producto para refinar éste a partir de sus comentarios.
SS-CV06: Imposibilidad de liberar el sistema por etapas sucesivas a lo largo del proyecto.
SS-CV07: Impresión de un desarrollo lento.
SS-CV08: Incremento de los riesgos durante el proceso de desarrollo y ninguna gestión de los mismos.

Categoría: Comunicación con la directiva y con el cliente

SS-CD01: Percepción de un desarrollo lento por parte del cliente.
SS-CD02: Percepción de un desarrollo lento por parte de la directiva.
SS-CD03: No existen los medios (artefactos que soporten y justifiquen el alcance) por los cuales la dirección del equipo de desarrollo y la del cliente puedan llegar a un acuerdo en cuanto a los tiempos de entrega y el contenido de los mismas. Esta situación genera un ambiente de desconfianza y malestar.

Categoría: Equipo de desarrollo

SS-ED02: El personal a cargo del equipo de desarrollo no tiene los medios ni la capacidad para exigir entregables con calidad.
SS-ED03: El equipo de desarrollo está llevando jornadas de trabajo de más de 50 horas a la semana.
SS-ED04: Las relaciones personales entre los miembros del equipo se han deteriorado.

Categoría: Arquitectura y componentes del sistema

SS-AC05: El equipo de desarrollo y en general los involucrados desconocen la arquitectura del sistema.

Categoría: Uso de tecnologías, herramientas y lenguajes

La información del caso de estudio no nos permite identificar alguna causa o signo en esta categoría.

Categoría: Control de la calidad

SS-CC01: Inexistente.
SS-CC04: Resulta imposible exigir al equipo de desarrollo codificar de forma estándar, siguiendo un esquema de nombrado, documentación, etc.
SS-CC05: Expresar ante el cliente los estándares que aseguran la calidad del proyecto desarrollado se vuelve una tarea difusa y complicada.

a) Determinación de las causas de abandono del proyecto

Categoría: Especificación del alcance del proyecto
CP-EA01: Pobre especificación de los requerimientos.
CP-EA02: No se ha definido con claridad el alcance del sistema.
CP-EA06: La metodología escogida no es la correcta, o se está omitiendo el/los artefacto/s que contemplan el análisis de riesgos. El líder de proyecto no tiene la capacidad para incluir un análisis del tipo.

Categoría: Estimación y planeación del proyecto

CP-EP05: Se ha subestimado la complejidad y alcance del proyecto.
CP-EP07: La comunicación entre los diferentes involucrados en el proyecto es deficiente.
CP-EP08: Los artefactos que se están usando en el control del proyecto no están aportando lo necesario, o se está fallando en su uso.
CP-EP10: Los hitos no son alcanzables o se están moviendo continuamente.
CP-EP11: No existe un plan de calidad o no es lo suficientemente robusto.
CP-EP12: El control de riesgos no está siendo atendido a cabalidad.
CP-EP13: No existe documentación que soporte los riesgos del proyecto.
CP-EP14: Estimación y planeación inexistentes.

Categoría: Ciclo de vida y metodología de desarrollo

CP-CV01: No se ha seleccionado un ciclo de vida y/o metodología adecuados a las características y restricciones del proyecto (por ejemplo, entrega del producto por etapas, liberación de versiones, signos visibles de desarrollo, etc.).
CP-CV02: El desarrollo del proyecto no se ha estructurado de forma tal que se pueda apreciar claramente su progreso.
CP-CV03: La metodología elegida no aporta los elementos de control de riesgos necesarios (Por ejemplo cascada pura, entrega por etapas, etc.).

Categoría: Comunicación con la directiva y con el cliente

CP-CD05: Los artefactos dedicados a la comunicación, alcance y soporte de calidad, no están siendo usados debidamente (Project charter, Visión, Especificación complementaria, etc.).

Categoría: Equipo de desarrollo

CP-ED01: La planeación del proyecto ha sido defectuosa y no existe motivación en el equipo de desarrollo.
CP-ED02: Los hitos y asignaciones no están siendo claras para el grupo.
CP-ED03: No se tiene un buen nivel técnico en el equipo.
CP-ED05: No ha habido un correcto liderazgo
CP-ED07: Los recursos humanos son insuficientes.

Categoría: Arquitectura y componentes del sistema

CP-AC02: No se identificaron los atributos de calidad.
CP-AC06: Las vistas de diseño no están aportando la información suficiente.
CP-AC07: Falta de experiencia del arquitecto.
CP-AC08: La documentación no está siendo actualizada, lo que genera que lo que se ve en una vista en particular, no se refleja en otra.
CP-AC11: El equipo encargado de diseñar no tiene la suficiente experiencia para poder plasmar un diseño de calidad.

Categoría: Uso de tecnologías, herramientas y lenguajes

La información del caso de estudio no nos permite identificar alguna causa o signo en esta categoría.

Categoría: Control de la calidad

CP-CC01: No existe la cultura en el equipo de desarrollo que pueda llevar a desarrollar documentación orientada a la calidad.
CP-CC05: No se han destinado esfuerzos en busca de proporcionar la calidad al proyecto (por ejemplo, Talleres QAW, etc.).

b) Diagnóstico diferencial del proyecto de desarrollo de software

Una vez llevado a cabo el proceso de identificación de los signos y síntomas, así como de sus causas probables, la próxima fase a ejecutar será inferir mediante éstos, el diagnóstico del proyecto SI-CRPA. Como se puede apreciar en la *figura 8.7*, el proceso de diagnóstico diferencial aplica a la información recabada de signos, síntomas y causas probables de abandono del proyecto las reglas de producción contenidas en la base de conocimientos. Como ejemplo ilustrativo, en la *figura 8.10* se puede apreciar dos de las reglas de producción satisfechas y el problema diagnosticado por cada una de éstas con su correspondiente valor de certeza.

```
diagnóstico("Problemas con la Especificación del Alcance", 0.95) :-
        (signo(SS-EA01);signo(SS-EA02);signo(SS-EA05)),
        (causa(CP-EA01);causa(CP-EA02);
         causa(CP-EA06)).

diagnóstico("Problemas con la Estimación y Planeación", 1.0) :-
        (signo(SS-EP01);signo(SS-EP02);signo(SS-EP03);
         signo(SS-EP05);signo(SS-EP06);signo(SS-EP07);
         signo(SS-EP07);signo(SS-EP08);signo(SS-EP09);
         signo(SS-EP10);signo(SS-EP11);signo(SS-EP12)),
        (causa(CP-EP05);causa(CP-EP07);
         causa(CP-EP08);causa(CP-EP10);
         causa(CP-EP11);causa(CP-EP12);
         causa(CP-EP13);causa(CP-EP14)).
```

Figura 8.10: Reglas de producción aplicadas durante el Diagnóstico diferencial al caso de estudio SI-CRPA.

Del diagnóstico diferencial ejecutado podemos concluir lo siguiente:

- En primera instancia, tenemos un proyecto cuya conducción ha sido artesanal. Es decir, no se le ha aplicado uno solo de los principios fundamentales que la ingeniería de software nos provee. Usando el Proceso Unificado de Desarrollo (UP, de la sigla en inglés) como referencia, podemos concluir que la fase de inicio no ha sido bien ejecutada, pues es en ésta donde tendría que ser contemplado un análisis de viabilidad que nos permita realizar las primeras estimaciones del proyecto. Sin lo anterior, resulta imposible llevar a cabo la planeación.

- La mayoría de los problemas identificados en el proyecto se deben a una deficiente planeación, lo que a su vez llevó a que se le restara importancia a la conformación del equipo de trabajo, así como a la correcta elección del ciclo de vida, métodos y herramientas a utilizar.

En resumen, estamos ante un caso donde la etapa inicial fue subvalorada y que por consiguiente no aportó los elementos de calidad suficientes para pensar en un proyecto exitoso. Dado que la mayoría de los errores se cometieron en la etapa temprana del proyecto y que no se logró un avance considerable, se propone trabajar sobre los siguientes aspectos:

- Revisión de los requerimientos del sistema (documentación existente, entrevistas registradas, etc.).

- Efectuar una correcta estimación del tamaño del producto de software, esfuerzo personas-mes y planeación en meses, siendo conscientes que las fechas de entrega comprometidas muy probablemente tendrán que desfasarse.

- Integrar personal al equipo de trabajo. Dado que el proyecto está en las primeras fases de desarrollo, no contraviene el principio del mito hombre/mes propuesto por [Brooks, 1987].

La recomendación incluye un trabajo extra en la comunicación entre la directiva de la consultoría y la del cliente, pues las etapas en las que se tienen problemas, incluyen a ambas.

c) Estudio de factibilidad para la recuperación

Considerando que: i) del caso de estudio no es posible conocer el tiempo estimado para el desarrollo del proyecto, ii) se sabe que al menos existen tres meses de retraso, iii) el sistema tiene que ser atacado desde sus primeras etapas, y iv) ya se ha invertido tiempo en un segundo análisis de requerimientos; se propone lo siguiente:

i) Recuperar la etapa de negociación con miras a continuar el desarrollo del proyecto, sin perder de vista el retraso de tres meses, lo que conllevará a una repercusión en costos.

ii) Cerrar el proyecto, recuperando el trabajo llevado a cabo en el segundo análisis.

Independientemente de la decisión final tomada, el plan de recuperación debería ser tomado en cuenta como un referente para el desarrollo de posteriores proyectos.

VIII.5 Conclusiones de la propuesta metodológica

En el punto VIII.4 hemos presentado los aspectos teórico-metodológicos de un plan para la recuperación de proyectos de software, el cual ha tomado como marco de referencia los elementos y procesos de inferencia que caracterizan el diagnóstico médico. Siguiendo esta filosofía, el enfoque propuesto centra la recuperación de proyectos en tres fases claves: i) la identificación de signos y síntomas de problemas y/o abandono, ii) la detección de las causas que originaron la manifestación de dicha sintomatología, y iii) el diagnóstico

diferencial para determinar la problemática que caracteriza el proyecto. A través de dos escenarios y un caso de estudio, relativos a proyectos en problemas, se intentó mostrar al lector la robustez, facilidad y alta aplicabilidad del enfoque de recuperación propuesto, el cual pretende constituir una valiosa herramienta para el desarrollador de software en la compleja tarea de recuperación de proyectos.

Cabe señalar que no fueron incluidas en su totalidad las imprescindibles tablas de referencia (las cuales funcionan como listas de chequeo) para la identificación de signos y síntomas de problemas y/o abandono y las causas que los originan, optándose por presentar solo, y a modo ilustrativo, un pequeño fragmento de las mismas. De igual forma, el trabajo se centra en la presentación y descripción detallada de solo cuatro de las seis fases que componen el plan de recuperación, siendo definitivamente éstas las más importantes.

La próxima etapa en el desarrollo del presente trabajo, la cual no es parte del contenido del presente material, será el diseño e implementación del plan de recuperación propuesto, de forma tal que el mismo se convierta en una herramienta computacional fuertemente visual e interactiva, con las características propias de un ambiente de trabajo integrado (IDE, de las siglas en inglés), proporcionando al usuario el soporte necesario para la recuperación de proyectos de software [González y Soto, 2010]. En este sentido ya se han logrado avances significativos como se puede apreciar en *las figuras 8.11* y *8.12*.

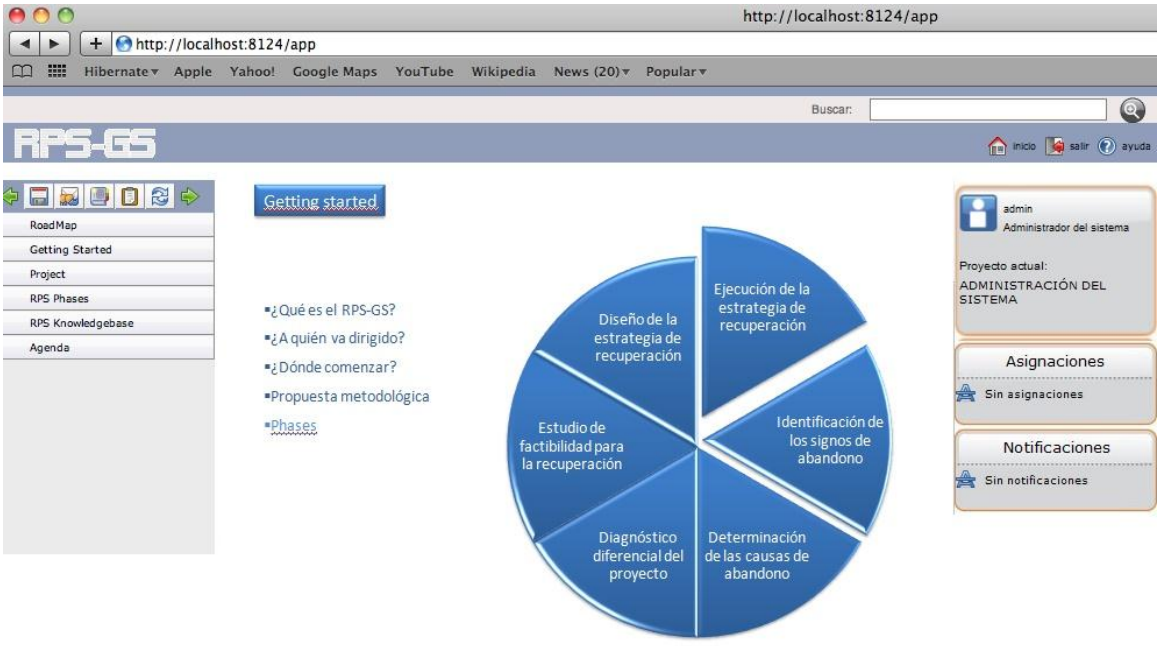


Figura 8.11: RPS-GS *framework*: la herramienta computacional de soporte al Plan de Recuperación.

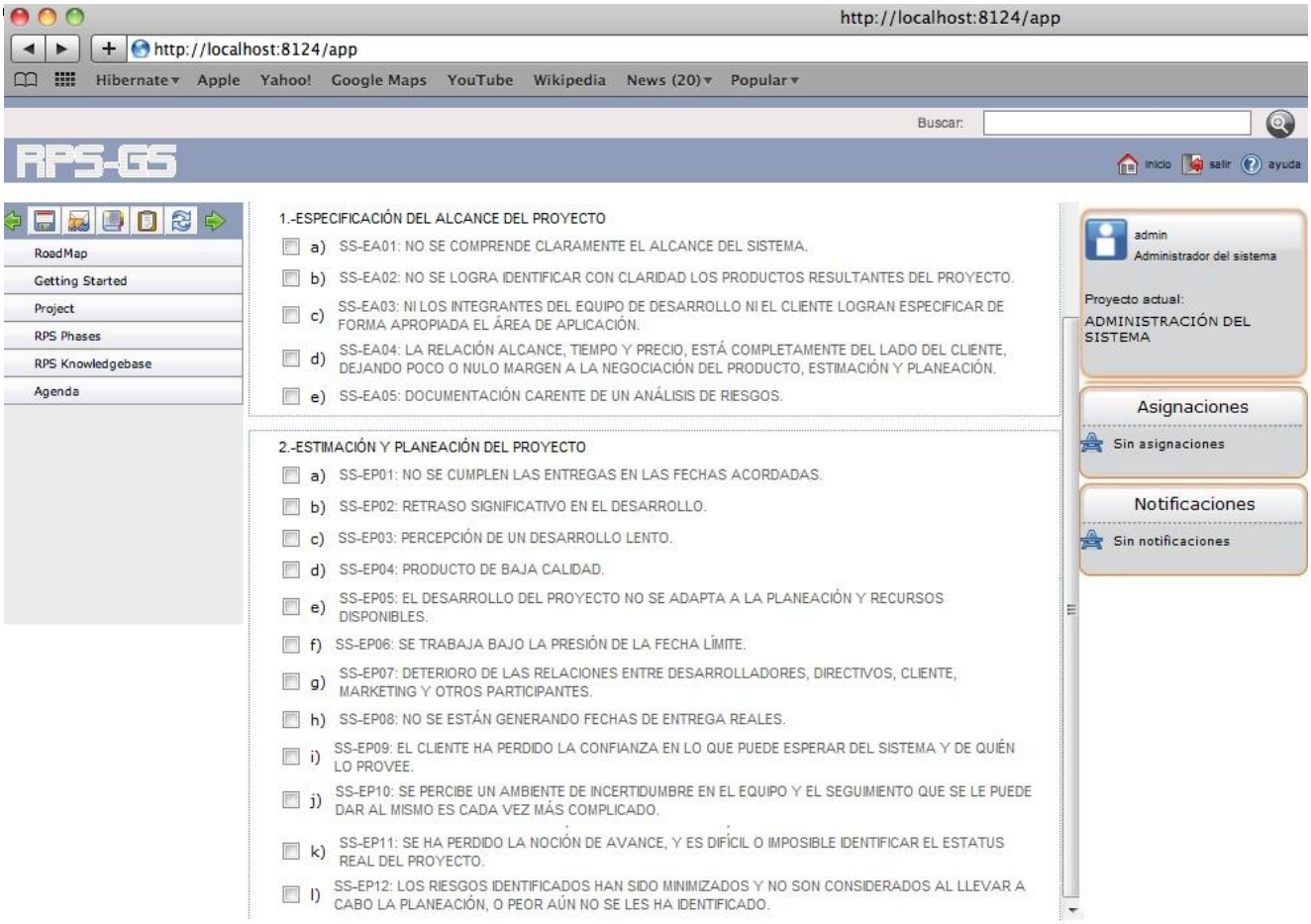


Figura 8.12: Identificación de signos y síntomas de abandono a través del RPS-GS *framework*.

Bibliografía

[1] Ambler, S., A Manager's Introduction to The Rational Unified Process (RUP), 2005, *http://www.ambysoft.com/downloads/managersIntroToRUP.pdf*

[2] Ambler, Scott, The Object Primer 3rd Edition: Agile Model Driven Development with UML 2, U.S.A.: Cambridge University Press, 2008.

[3] Anderson, David, Feature-Driven Development: towards a TOC, Lean and Six Sigma solution for software engineering, Theory of Constraints, International Certification Organization, Microsoft, 2004.

[4] Aguilar Sierra Alejandro, introducción a la programación extrema, en Revista Digital Universitaria, Vol (3) No. 4, México: UNAM, 2002.

[5] Araújo Alejandro, Test Driven Development: Fortalezas y Debilidades. Montevideo-Uruguay: Instituto de Computación, Fac. de Ingeniería. UDELAR, 2007.

[6] Barzanallana R., Herramientas Case. Universidad de Murcia, 2010-2011. http://www.um.es/docencia/barzana/IAGP/Enlaces/CASE_principales.html

[7] Boehm, B., *Software Engineering Economics*, Englewood Cliffs, Prentice Hall, 1981.

[8] Bohem, B., & Papaccio P., *Understanding and controlling Software Costs*, IEEE Transactions on Software Engineering SE-10 (may): 290-303.

[9] Booch Grady, Rumbaugh Jim, & Jacobson Ivar. The Unified Modeling Language User Guide. Addison-Wesley, 1999.

[10] Briseño, A.M., *Administración de Proyectos*, 2003, http://www.tidap.gob.mx/Presentaciones/talleres/t_AnaBrise%F1o.pdf

[11] Brooks, F.P. *No Silver Bullet Essence and Accidents of Software Engineering*, IEEE Computer. 20 (4), 10-19, 1987.

[12] Center of Systems and Software Engineering http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html

[13] Charbonneau S., Software Project Management. A Mapping between RUP and the PMBOK, 2004, http://www.ibm.com/developerworks/rational/library/4721.html

[14] González D. H., *Las métricas de Software y su uso en la región*, Tesis de Licenciatura en Ingeniería en Sistemas Computacionales, Universidad de las Américas de Puebla, 2001.

[15] González, P.P., Soto-Galindo, I. *RPS-GS: Un Plan para la Recuperación de Proyectos de Software*. En González Fraga, J.A., Ascencio López, J.I., Martínez Martínez, E., Meza Kubo, M.V., Osorio Cayetano, O.R. (Eds.) Libro Electrónico Memorias 3er Congreso Internacional en Ciencias Computacionales CICOMP'2010, Editorial Universidad Autónoma de Baja California, ISBN: 978-607-7753-81-0, 2010.

[16] Fenton E. N. *Software Metrics A rigorous approach* .Chapman & Hall, 1991.
[17] Fowler, Martin. The new methodology. *http://www.martinfowler.com/articles/newMethodology.html*. Traducción al español. http://www.programacionextrema.org/articulos/newMethodology. 2001.
[18] IEEE-1058, publicado en 1987, revisado en 1993.
[19] IPFUG: International Users Group Function Point http://www.ifpug.org/publications/guidelines.htm
[20] Jackson Michael, Software Requirements & Specifications, a lexicon of practice, principles and prejudices. Essex: ACM press/Addison-Wesley, 1998.
[21] Khalifa Mohamed, Verner June M., Drivers for Software Development Method Usage, IEEE Transactions on Engineering Management, Vol. 47, No. 3, pp. 360-369, 2000.
[22] López, B., *Administración de proyectos*, http://www.itnuevolaredo.edu.mx/takeyas.
[23] Luckey T., Phillips J., *Software Project Management for Dummies*, Wiley, U.S.A., 2006.
[24] Mangione Carmine, Software Project Failure: The Reasons, The Costs, http://www.cioupdate.com/reports/article.php/1563701/Software-Project-Failure-The-Reasons-The-Costs.htm, 2003.
[25] Manifesto for Agile Software Development, http://www.agilemanifesto.org., 2001.
[26] McConnell, S., *Desarrollo y gestión de proyectos informáticos*, McGraw Hill y Microsoft Press, España, 1997.
[27] McManus John & Wood-Harper Trevor, A study in project failure. The chartered institute for IT, http://www.bcs.org/server.php?show=ConWebDoc.19584.
[28] Metodología MÉTRICA versión 3. Gestión de Proyectos. Ministerio de Administraciones Públicas. Gobierno de España. http://administracionelectronica.gob.es/archivos/pae_000001038.pdf
[29] Norris & Rigby, Ingeniería de Software explicada, México: Megabyte-Noriega editores, 1994.
[30] Pazderski P., Agile through SCRUM, Software Process Consultant Inc. 26 May 2010 CQAA Lunch & Learn, 2010.
[31] Périssé M.C., *Proyecto Informático una metodología simplificada*. Argentina. http://www.cyta.com.ar/biblioteca/bddoc/bdlibros/proyectoinformatico/libro/, 2001.
[32] Pfleeger, S., Atlee J., Ingeniería de software, Teoría y práctica, Buenos Aires: Pearson Education, Buenos Aires, 2002.
[33] Pfleeger, Shari and Atlee Joanne (2006). Software Engineering, Theory and practice. Third edition, New Jersey U.S.A.: Pearson Prentice Hall.
[34] Piorun, D. ¿Por qué fracasan los proyectos?, 2003. http://www.degerencia.com/articulo/por_que_fracasan_los_proyectos
[35] PMI Guía de los fundamentos para la Dirección de Proyectos (Guía del PMBOK) Project Management Institute, 2008.
[36] Pressman, Roger S. *Ingeniería del Software: Un Enfoque Práctico*, Cuarta edición, España, McGraw-Hill, 1998.

[37] Pressman, Roger S. *Ingeniería del Software: Un enfoque práctico*, 6a edición. Editorial McGraw Hill, México, 2006.
[38] Pucha, L., Ingeniería del Software, Gestión de proyectos http://eqaula.org/eva/mod/resource/view.php?id=1581
[39] Robbins S.P. & De Censo D.A., *Fundamentos de Administración*. Conceptos y aplicaciones, Prentice Hall, México, 1996.
[40] Senn J. A., *Análisis y diseño de sistemas de información*, Editorial McGraw-Hill, 2ª ed., México, 1992.
[41] Sommerville, *Software Engineering*, 8º ed., Addison Wesley, 2006.
[42] Standish Group, The Standish Group Report: Chaos Report, 1994.
[43] Soto-Galindo, I., González, P.P. *Un Plan para la Recuperación de Proyectos de Software inspirado en el Proceso de Diagnóstico Médico*. En García Gaona, A.R. y Sánchez Guerrero L. (Eds.) Desarrollo Tecnológico, Libro Electrónico "XXIII Congreso Nacional y IX Congreso Internacional de Informática y Computación 2010", Editorial Alfa Omega, ISBN: 978-607-707-097-9, 2010.
[44] Subsecretaría de la función pública *Taller de proyectos de procesos http://*www.funcionpublica.gob.mx/ssfp/dgeabg/doctos/da/CONTROL.ppt
[45] Toledo, R., *La administración de proyectos como estrategia de crecimiento*, http://direccionestrategica.itam.mx/Administrador/Uploader/material/LaAdministracindeProyectoscomoEstrategiadeCrecimiento.pdf
[46] Symons, C., *Software Sizing and Estimating: Mk II FPA (Function Point Analysis)*. John Wiley & Sons, England, 1991.
[47] Turban E., McClean E., Wetherbe J., *Tecnologías de información para la administración*; Editorial CECSA, México, 2001.
[48] Weitzenfeld, Alfredo. "Ingeniería de Software Orientada a Objetos con UML, Java e Internet". Editorial Thomson, 2004.
[49] West, Dave and Grant Tom, Agile Development:ent: Mainstream Adoption Has Changed Agility. Trends in Real-World Adoption Of Agile Methods. Application Development & Program Management Professional, January, Forrester Research Inc., http://www.mendeley.com/research/agile-development-mainstream-adoption-changed-agility/, 2010.
[50] Wiegers, K., "More about software requirements". Microsoft Press, 2006.

Notas del Curso: Administración de Proyectos
Se terminó de imprimir el 30 de noviembre de 2012 en
Publidisa Mexicana S. A. de C.V.
Calz. Chabacano No. 69, Planta Alta
Col. Asturias C.P. 06850.