Contenido Temático del Tema II: Ingeniería del Software y Gestión de Proyectos

Ingeniería del Software. Características del software. Proceso, modelos, costos y métodos de la Ingeniería del Software. Características y atributos del software. La complejidad inherente al software.
Gestión de Proyectos. Métricas de proyectos. Métricas orientadas al tamaño. Métricas orientadas a la función. Eficiencia y calidad. Estimación y planificación de proyectos. Técnicas de estimación, de descomposición y empíricas. Estimación y planificación ágil. Costos y recursos. Técnicas de evaluación y revisión de actividades. Calendarización del proyecto.
Análisis y gestión de riesgos. Gestión de calidad. Herramientas CASE-I de planificación y de estimación de proyectos.

Introducción

La ingeniería del software es una disciplina de la ingeniería cuyo objetivo es el desarrollo de sistemas de software de bajo coste.

Características del producto que lo hacen complejo

Este es abstracto e intangible. No está restringido por materiales, o gobernado por leyes físicas o por procesos de manufactura. De alguna forma, esto simplifica la ingeniería del software ya que no existen limitaciones físicas del potencial del software. Sin embargo, esta falta de restricciones naturales significa que el software puede llegar a ser extremadamente complejo y, por lo tanto, muy difícil de entender.

Crisis de la Ingeniería del Software

La noción de *ingeniería del software* fue propuesta inicialmente en 1968 en una conferencia sobre la «crisis del software».
Esta crisis del software fue el resultado de la introducción de las nuevas computadoras hardware basadas en circuitos integrados. Su poder hizo que las aplicaciones hasta ese entonces irrealizables fueran una propuesta factible. El software resultante fue de órdenes de magnitud más grande y más complejo que los sistemas de software previos.
El enfoque informal para el desarrollo del software no era muy bueno. Los grandes proyectos a menudo tenían años de retraso. Costaban mucho más de lo presupuestado, eran irrealizables, difíciles de mantener y con un desempeño pobre. El desarrollo de software estaba en crisis. Los costos del hardware se tambaleaban mientras que los del software se incrementaban con rapidez. Se necesitaban nuevas técnicas y métodos para controlar la complejidad inherente a los sistemas grandes.
Estas técnicas han llegado a ser parte de la ingeniería del software y son ampliamente utilizadas. Sin embargo, cuanto más crezca nuestra capacidad para producir software, también lo hará la complejidad de los sistemas de software solicitados. Las nuevas tecnologías resultantes de la convergencia de las computadoras y de los sistemas de

comunicación y complejas interfaces gráficas de usuario impusieron nuevas demandas a los ingenieros de software.
Debido a que muchas compañías no aplican de forma efectiva las técnicas de la ingeniería del software, demasiados proyectos todavía producen software que es irrealizable, entregado tarde y sobre presupuestado.
Se puede afirmar que hemos hecho enormes progresos desde 1968 y que el desarrollo de esta ingeniería ha mejorado considerablemente nuestro software. Comprendemos mucho mejor de las actividades involucradas en el desarrollo de software. Hemos desarrollado métodos efectivos de especificación, diseño e implementación del software. Las nuevas notaciones y herramientas reducen el esfuerzo requerido para producir sistemas grandes y complejos.
Ahora sabemos que no hay una enfoque «ideal» a la ingeniería del software. La amplia diversidad de diferentes tipos de sistemas y organizaciones que usan estos sistemas significa que necesitamos una diversidad de enfoques al desarrollo de software. Sin embargo, las nociones fundamentales de procesos y la organización del sistema son la base de todas estas técnicas, y éstas son la esencia de la ingeniería del software.

Preguntas frecuentes sobre la ingeniería del software

Que es el software?

Programas de ordenador y la documentación asociada. Los productos de software se pueden desarrollar para algún diente en particular o para un mercado general.
Los ingenieros de software se concentran en el desarrollo de productos de software, es decir, software que se vende a un cliente. Existen dos tipos de productos de software:

1. *Productos genéricos*. Son sistemas aislados producidos por una organización de desarrollo y que se venden al mercado abierto a cualquier cliente que le sea posible comprarlos.
Ejemplos de este tipo de producto son el software para PCs tales como bases de datos, procesadores de texto, paquetes de dibujo y herramientas de gestión de proyectos.

2. *Productos personalizados (o hechos a medida)*. Son sistemas requeridos por un cliente en particular. Un contratista de software desarrolla el software especialmente para ese cliente. Ejemplos de este tipo de software son los sistemas de control para instrumentos electrónicos, sistemas desarrollados para llevar a cabo procesos de negocios específicos y sistemas de control del tráfico aéreo.

Una diferencia importante entre estos diferentes tipos de software es que, en los productos genéricos, la organización que desarrolla el software controla su especificación. La especificación de los productos personalizados, por lo general, es desarrollada y controlada por la organización que compra el software. Los desarrolladores de software deben trabajar con esa especificación.
No obstante, la línea de separación entre estos tipos de productos se está haciendo cada vez más borrosa. Cada vez más compañías de software empiezan con un sistema genérico y lo adaptan a las necesidades de un cliente en particular.

¿Qué es la ingeniería del software?

La ingeniería del software es una disciplina de ingeniería que comprende todos los aspectos de la producción de software.

¿Cuál es la diferencia entre ingeniería del software y ciencia de la computación?

La ciencia de la computación comprende la teoría y los fundamentos; la ingeniería del software comprende tas formas prácticas para desarrollar y entregar un software útil.

¿Cuál es ta diferencia entre ingeniería del software e ingeniería de sistemas?

La ingeniería de sistemas se refiere a todos los aspectos del desarrollo de sistemas informáticos, incluyendo hardware, software e ingeniería de procesos. La ingeniería del software es parte de este proceso.

¿Qué es un proceso del software

Un conjunto (4) de actividades cuya meta es el desarrollo o evolución del software.
Existen cuatro actividades fundamentales de procesos (incluidas más adelante en este libro) que son comunes para todos los procesos del software. Estas actividades son:

1. *Especificación del software* donde los clientes e ingenieros definen el software a producir y las restricciones sobre su operación.

2. *Desarrollo del softM'are* donde el software se diseña y programa.

3. *Validación del software* donde el software se valida para asegurar que es lo que el cliente requiere.

4. *Evolución del software* donde el software se modifica para adaptarlo a los cambios requeridos por el cliente y el mercado.

¿Qué es un modelo de procesos del software?

Una representación simplificada de un proceso del software, presentada desde una perspectiva especifica.
Algunos ejemplos de estos tipos de modelos que se pueden producir son:

1. *Un modelo de flujo de trabajo.* Muestra la secuencia de actividades en el proceso junto con sus entradas, salidas y dependencias. Las actividades en este modelo representan acciones humanas.

2. *Un modelo de flujo de datos o de actividad.* Representa el proceso como un conjunto de actividades, cada una de las cuales realiza alguna transformación en los datos. Muestra cómo la entrada en el proceso, tal como una especificación, se transforma en una salida, tal como un diseño. Pueden representar transformaciones llevadas a cabo por las personas o por las computadoras.

3. *Un modelo de rollacción.* Representa los roles de las personas involucrada en el proceso del software y las actividades de las que son responsables.

La mayor parte de los modelos de procesos del software se basan en uno de los tres modelos generales o paradigmas de desarrollo de software:

1. *El enfoque en cascada.* Considera las actividades anteriores y las representa como fases de procesos separados, tales como la especificación de requerimientos, el diseño del software, la implementación, las pruebas, etcétera. Después de que cada etapa queda definida «se firma» y el desarrollo continúa con la siguiente etapa.

2. *Desarrollo iterativo.* Este enfoque entrelaza las actividades de especificación, desarrollo y validación. Un sistema inicial se desarrolla rápidamente a partir de especificaciones muy abstractas. Éste se refina basándose en las peticiones del cliente para producir un sistema que satisfaga las necesidades de dicho cliente. El sistema puede entonces ser entregado. De forma alternativa, se puede reimplementar utilizando un enfoque más estructurado para producir un sistema más sólido y mantenible.

3. *Ingeniería del software basada en componentes (CBSE}.* Esta técnica supone que las partes del sistema existen. El proceso de desarrollo del sistema se enfoca en la integración de estas partes más que desarrollarlas desde el principio.

¿Cuáles son los costos de la ingeniería del software?

A grandes rasgos, el 60 % de los costos son de desarrollo, el 40 % restante son de pruebas. En el caso del software personalizado, los costos de evolución a menudo exceden los de desarrollo.
No existe una respuesta sencilla a esta pregunta ya que la distribución de costos a través de las diferentes actividades en el proceso del software depende del proceso utilizado y del tipo de software que se vaya a desarrollar.
Si se considera que el costo total del desarrollo de un sistema de software complejo es de 100 unidades de costo, la Figura 1.2 muestra cómo se gastan éstas en las diferentes actividades del proceso.

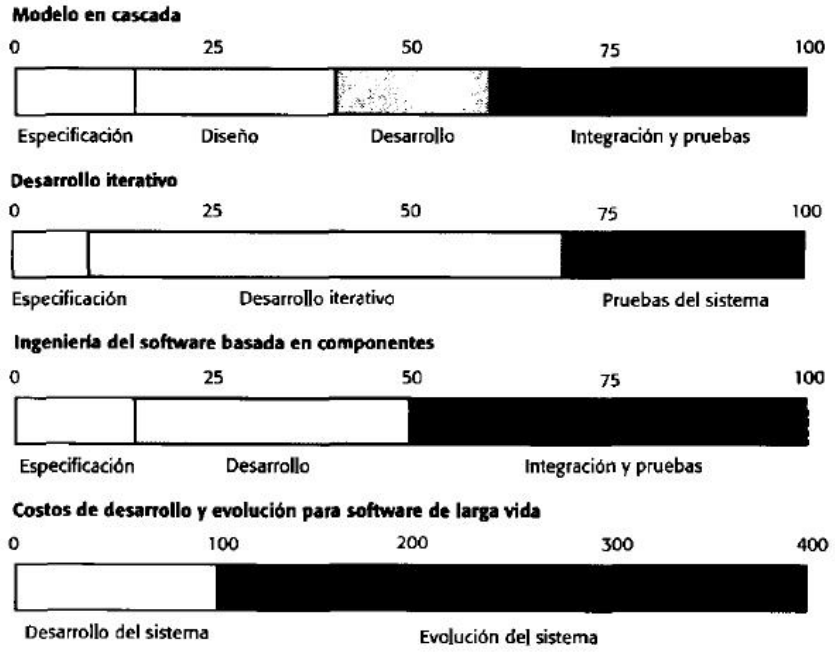


Figura 1.2 Distribución de costos de las actividades de la ingeniería del software.

En el enfoque en cascada, los costos de especificación, diseño, implementación e integración se miden de forma separada.
Si el software se desarrolla utilizando un enfoque iterativo, no existe división entre la especificación, el diseño y el desarrollo. En este enfoque, los costos de la especificación se reducen debido a que sólo se produce la especificación de alto nivel antes que el desarrollo. La especificación, el diseño, la implementación, la integración y las pruebas se llevan a cabo en paralelo dentro de una actividad de desarrollo. Sin embargo, aún se necesita una actividad independiente de pruebas del sistema una vez que la implementación inicial esté completa.
En el desarrollo basado en componentes los costos de desarrollo se reducen en relación a los costos de integración y pruebas. Los costos de integración y pruebas se incrementan porque tenemos que asegurarnos de que los componentes
que utilizamos cumplen realmente su especificación y funcionan como se espera con otros componentes.
Además de los costos de desarrollo, existen costos asociados a cambios que se le hacen al software una vez que está en uso. Los costos de evolución varían drásticamente dependiendo del tipo de sistema. Para sistemas software de larga vida, tales como sistemas de orden y de control que pueden ser usados durante 10 años o más, estos costos exceden a los de desarrollo por un factor de 3 o 4, como se muestra en la barra inferior en la Figura 1.3.

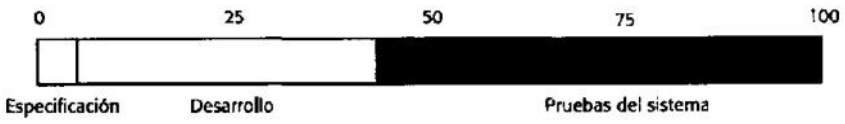


Figura 1.3 Costos del desarrollo del producto.

Los costos de evolución para productos de software genéricos son difíciles de estimar. En muchos casos, existe poca evolución de un producto. Una vez que una versión de producto se entrega, se inicia el trabajo para entregar la siguiente y, por razones de mercadotecnia, ésta se presenta como un producto nuevo (pero compatible) más que como una versión modificada de un producto que el usuario ya adquirió. Por lo tanto, los costos de evolución no se consideran de forma separada como en el caso del software personalizado, sino que son sencillamente los costos del desarrollo para la siguiente versión del sistema.

¿Qué son los métodos de la ingeniería del software?

Enfoques estructurados para el desarrollo de software que incluyen modelos de sistemas, notaciones, reglas, sugerencias de diseño y gulas de procesos.
No existe un método ideal, y métodos diferentes tienen distintas áreas donde son aplicables. Por ejemplo, los métodos orientados a objetos a menudo son apropiados para sistemas interactivos, pero no para sistemas con requerimientos rigurosos de tiempo real.
Todos los métodos se basan en la idea de modelos gráficos de desarrollo de un sistema y en el uso de estos modelos como un sistema de especificación o diseño. Los métodos incluyen varios componentes diferentes (Figura 1.4).

Descripciones del modelo del sistema	Descripciones de los modelos del sistema que desarrollará y la notación utilizada para definir estos modelos.	Modelos de objetos, de flujo de datos, de máquina de estado, etcétera.
Reglas	Restricciones que siempre aplican a los modelos de sistemas.	Cada entidad de un modelo de sistema debe tener un nombre único.
Recomendaciones	Heurística que caracteriza una buena práctica de diseño en este método. Seguir estas recomendaciones debe dar como resultado un modelo del sistema bien organizado.	Ningún objeto debe tener más de siete subobjetos asociados a él.
Gulas en el proceso	Descripciones de las actividades que deben seguirse para desarrollar los modelos del sistema y la organización de estas actividades.	Los atributos de los objetos deben documentarse antes de definir las operaciones asociadas a un objeto.

Figura 1.4 Componentes del método.

¿Qué es CASE (Ingeniería del Software Asistida por Ordenador)?

Sistemas de software que intentan proporcionar ayuda automatizada a las actividades del proceso del software. Los sistemas CASE a menudo se utilizan como apoyo al método.

¿Cuáles son los atributos de un buen software?

El software debe tener la funcionalidad y el rendimiento requeridos por el usuario, además de ser mantenible, confiable y fácil de utilizar. (Fig. 1.5).

Mantenibilidad	El software debe escribirse de tal forma que pueda evolucionar para cumplir las necesidades de cambio de los clientes. Éste es un atributo crítico debido a que el cambio en el software es una consecuencia inevitable de un cambio en el entorno de negocios.
Confiabilidad	La confiabilidad del software tiene un gran número de características, incluyendo la fiabilidad, protección y seguridad. El software confiable no debe causar daños físicos o económicos en el caso de una falla del sistema.
Eficiencia	El software no debe hacer que se malgasten los recursos del sistema, como la memoria y los ciclos de procesamiento. Por lo tanto, la eficiencia incluye tiempos de respuesta y de procesamiento, utilización de la memoria, etcétera.
Usabilidad	El software debe ser fácil de utilizar, sin esfuerzo adicional, por el usuario para quien está diseñado. Esto significa que debe tener una interfaz de usuario apropiada y una documentación adecuada.

Figura 1.5 Atributos esenciales de un buen software.

¿Cuáles con los retos fundamentales a los que se enfrenta la ingeniería de software?

Enfrentarse con la creciente diversidad, las demandas para reducir los tiempos de entrega y el desarrollo de software fiable. Fundamentalmente son tres retos a enfrentar:

1. *El reto de la heterogeneidad*. Cada vez más, se requiere que los sistemas operen como sistemas distribuidos en redes que incluyen diferentes tipos de computadoras y con diferentes clases de sistemas de soporte. A menudo es necesario integrar software nuevo con sistemas heredados más viejos escritos en diferentes lenguajes de programación. El reto de la heterogeneidad es desarrollar técnicas para construir software confiable que sea lo suficientemente flexible para adecuarse a esta heterogeneidad.

2. *El reto de la entrega*. Muchas técnicas tradicionales de ingeniería del software consumen tiempo. El tiempo que éstas consumen es para producir un software de calidad. Sin embargo, los negocios de hoy en día deben tener una gran capacidad de respuesta y cambiar con mucha rapidez. Su software de soporte también debe cambiar con la misma rapidez. El reto de la entrega es reducir los tiempos de entrega para sistemas grandes y complejos sin comprometer la calidad del sistema.

3. *El reto de la confianza*. Puesto que el software tiene relación con todos los aspectos de nuestra vida, es esencial que podamos confiar en él. Esto es especialmente importante en sistemas remotos de software a los que se accede a través de páginas web o de interfaces de servicios web. El reto de la confianza es desarrollar técnicas que demuestren que los usuarios pueden confiar en el software.

Responsabilidad profesional y ética

Los ingenieros de software deben aceptar que su trabajo comprende responsabilidades más amplias que simplemente la aplicación de habilidades técnicas. Deben comportarse de una forma ética y moral responsable si es que desean ser respetados como profesionales, algunas de éstas responsabilidades son:

- *Confidencialidad*. Usted normalmente debe respetar la confidencialidad de sus empleadores o clientes independientemente de que se haya firmado un acuerdo formal de confidencialidad.
- *Competencia*. No debe falsificar su nivel de competencia, ni aceptar conscientemente trabajos que están fuera de su capacidad.
- *Derechos de propiedad intelectual*. Debe ser consciente de las leyes locales que gobiernan el uso de la propiedad intelectual, como las patentes y el copyright. Debe asegurarse de que la propiedad intelectual de los empleadores y clientes está protegida.
- *Uso inapropiado de las computadoras*. No debe emplear sus habilidades técnicas para utilizar de forma inapropiada las computadoras de otras personas. El uso inapropiado de las computadoras va desde los relativamente triviales (juegos) hasta los extremadamente serios (virus).

Las sociedades e instituciones profesionales tiene que desempeñar un papel importante en el establecimiento de estándares éticos. Organizaciones como la ACM, el IEEE (Instituto de Ingenieros Eléctricos y Electrónicos) y la British Computer Society publican un código de conducta profesional o de ética. Los miembros de estas organizaciones se comprometen a cumplir ese código cuando se inscriben en ellas. Estos códigos de conducta generalmente se refieren al comportamiento ético fundamental.
La ACM y el IEEE han cooperado para crear un código de ética y práctica profesional.

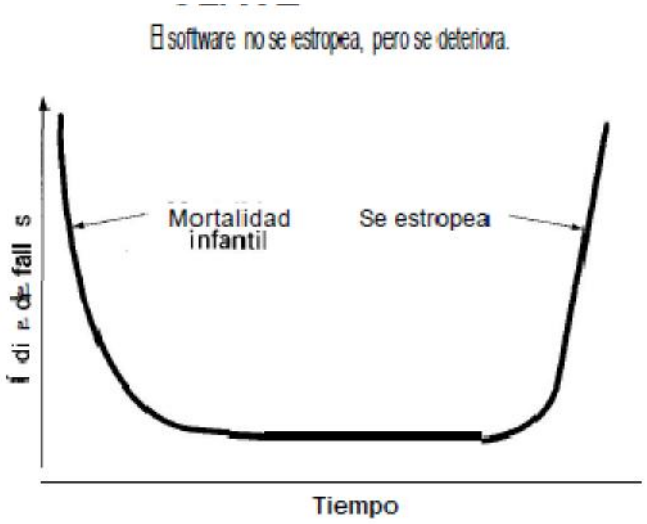
Características del software

Para poder comprender lo que es el software (y consecuentemente la ingeniería del software), es importante examinar las características del software que lo diferencian de otras cosas que los hombres pueden construir. Cuando se construye hardware, el proceso creativo humano (análisis, diseño, construcción, prueba) se traduce finalmente en una forma física. Si construimos una nueva computadora, nuestro boceto inicial, diagramas formales de diseño y prototipo de prueba, evolucionan hacia un producto físico (chips, tarjetas de circuitos impresos, fuentes de potencia, etc.).
El software es un elemento del sistema que es lógico, en lugar de físico. Por tanto el software tiene unas características considerablemente distintas a las del hardware:

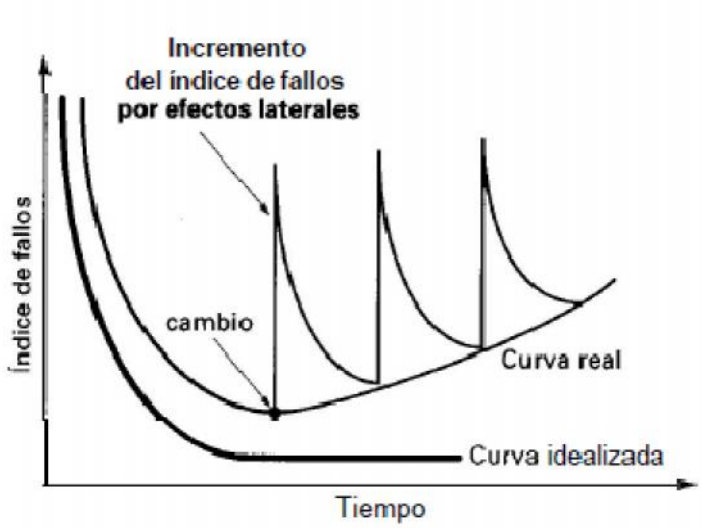
1. El software se desarrolla. Aunque existen similitudes entre el desarrollo del software **y** la construcción del hardware, ambas actividades son fundamentalmente diferentes. En ambas actividades la buena calidad se adquiere mediante un buen diseño, pero la fase de construcción del hardware puede introducir problemas de calidad que no existen (o son fácilmente corregibles) en el software. Ambas actividades dependen de las personas, pero la relación entre las personas dedicadas y el trabajo realizado es completamente diferente para el software (véase el Capítulo 7). Ambas actividades requieren la construcción de un «producto» pero los enfoques son diferentes.
Los costes del software se encuentran en la ingeniería. Esto significa que los proyectos de software no se pueden gestionar como si fueran proyectos de fabricación.

2. ***El*** *software no se «estropea»*. La Figura describe, para el hardware, la proporción de fallos como una función del tiempo. Esa relación, denominada frecuentemente «curva de

bañera», indica que el hardware exhibe relativamente muchos fallos al pnncipio de su vida (estos fallos son atribuibles normalmente a defectos del diseño o de la fabricación); una vez corregidos los defectos, la tasa de fallos cae hasta un nivel estacionario (bastante bajo, con un poco de optimismo) donde permanece durante un cierto periodo de tiempo. Sin embargo, conforme pasa el tiempo, el hardware empieza a desgastarse y la tasa de fallos se incrementa.



El software no es susceptible a los males del entorno que hacen que el hardware se estropee. Por tanto, en teoría, la curva de fallos para el software tendría la forma que muestra la Figura siguiente Los defectos no detectados harán que falle el programa durante las primeras etapas de su vida. Sin embargo, una vez que se corrigen (suponiendo que no se introducen nuevos errores) la curva se aplana, como se muestra.



La curva idealizada es una gran simplificación de los modelos reales de fallos del software. Sin embargo la implicación es clara, el software no se estropea. ¡Pero se deteriora!.

Esto que parece una contradicción, puede comprenderse mejor considerando «la curva actual» mostrada en
la Figura. Durante su vida, el software sufre cambios (mantenimiento). Conforme se hacen los cambios, es bastante probable que se introduzcan nuevos defectos, haciendo que la curva de fallos tenga picos.
Antes de que la curva pueda volver al estado estacionario original, se solicita otro cambio, haciendo que de nuevo se cree otro pico. Lentamente, el nivel mínimo de fallos comienza a crecer -e1 software se va deteriorando debido a los cambios.
Otro aspecto de ese deterioro ilustra la diferencia entre el hardware y el software. Cuando un componente de hardware se estropea se sustituye por una pieza de repuesto.
No hay piezas de repuesto para el software. Cada fallo en el software indica un error en el diseño o en el proce***so*** mediante el que se tradujo el diseño a código máquina ejecutable. Por tanto, el mantenimiento del software tiene una complejidad considerablemente mayor que la del mantenimiento del hardware.

3. *Aunque la industria tiende a ensamblar componentes, la mayoría del software se construye a medida.*
Consideremos la forma en la que se diseña y se construye el hardware de control para un producto basado en computadora. El ingeniero de diseño construye un sencillo esquema de la circuitería digital, hace algún análisis fundamental para asegurar que se consigue la función adecuada y va al armario donde se encuentran los catálogos de componentes digitales.
Después de seleccionar cada componente, puede solicitarse la compra.

A medida que la disciplina del software evoluciona, se crea un grupo de componentes de diseño estándar. Tornillos estándar y circuitos integrados preparados para la venta son solamente los dos mil componentes estándar que utilizan ingenieros mecánicos y eléctricos cuando diseñan nuevos sistemas. Los componentes reutilizables se han creado para que el ingeniero pueda concentrarse en elementos verdaderamente innovadores de un diseño, por ejemplo, las partes del diseño que representan algo nuevo.
En el mundo del hardware, la reutilización de componentes es una parte natural del proceso de ingeniería.
En el mundo del software es algo que sólo ha comenzado a lograrse en una escala amplia.
El componente de software debería diseñarse e implementarse para que pueda volver a ser reutilizado en muchos programas diferentes. En los años 60, se construyeron bibliotecas de subrutinas científicas reutilizables en una amplia serie de aplicaciones científicas y de ingeniería. Esas bibliotecas de subrutinas reutilizaban de forma efectiva algoritmos bien definidos, pero tenían un dominio de aplicación limitado.
Hoy en día, hemos extendido nuestra visión de reutilización para abarcar no sólo los algorítmos, sino también estructuras de datos. Los componentes reutilizables modernos encapsulan tanto datos como procesos que se aplican a los datos, permitiendo al ingeniero del software crear nuevas aplicaciones a partir de las partes reutilizables. Por ejemplo, las interfaces gráficas de usuario de hoy en día se construyen frecuentemente a partir de componentes reutilizables que permiten la creación de ventanas gráficas, de menús desplegables y de una amplia variedad de mecanismos de interacción.

Aplicaciones del software

El software puede aplicarse en cualquier situación en la que se haya definido previamente un conjunto específico de pasos procedimentales (es decir, un algoritmo) (excepciones notables a esta regla son el software de los sistemas expertos y de redes neuronales). El contenido y el determinismo de la información son factores importantes a considerar para determinar la naturaleza de una aplicación de software. El contenido se refiere al significado y a la forma de la información de entrada y salida.

El determinismo de la información se refiere a la predecibilidad del orden y del tiempo de llegada de los datos. Un programa de análisis de ingeniería acepta datos que están en un orden predefinido, ejecuta el algoritmo de análisis sin interrupción y produce los datos resultantes en un informe o formato gráfico. Se dice que tales aplicaciones son determinadas. Un sistema operativo multiusuario, por otra parte, acepta entradas que tienen un contenido variado y que se producen en instantes arbitrarios, ejecuta algoritmos que pueden ser interrumpidos por condiciones externas y produce una salida que depende de una función del entorno y del tiempo. Las aplicaciones con estas características se dice que son indeterminadas.
Algunas veces es difícil establecer categorías genéricas para las aplicaciones del software que sean significativas.
Conforme aumenta la complejidad del software, es más difícil establecer compartimentos nítidamente separados. Las siguientes áreas del software indican la amplitud de las aplicaciones potenciales:

Software de sistemas. El software de sistemas es un conjunto de programas que han sido escritos para servir a otros programas. Algunos programas de sistemas (por ejemplo: compiladores, editores y utilidades de gestión de archivos) procesan estructuras de información complejas pero determinadas. Otras aplicaciones de sistemas (por ejemplo: ciertos componentes del sistema operativo, utilidades de manejo de periféricos, procesadores de telecomunicaciones) procesan datos en gran medida indeterminados. En cualquier caso, el área del software de sistemas se caracteriza por una fuerte interacción con el hardware de la computadora; una gran utilización por múltiples usuarios; una operación concurrente que requiere una planificación, una compartición de recursos y una sofisticada gestión de procesos; unas estructuras de datos complejas y múltiples interfaces externas.

Software de tiempo real. El software que coordina/ analiza/controla sucesos del mundo real conforme ocurren, se denomina de tiempo real. Entre los elementos del software de tiempo real se incluyen: un componente de adquisición de datos que recolecta y da formato a la información recibida del entorno externo, un componente de análisis que transforma la información según lo requiera la aplicación, un componente de control/salida que responda al entorno externo, y un componente de monitorización que coordina todos los demás componentes, de forma que pueda mantenerse la repuesta en tiempo real (típicamente en el rango de un milisegundo a un segundo).

Software de gestión. El proceso de la información comercial constituye la mayor de las áreas de aplicación del software. Los «sistemas» discretos (por ejemplo: nóminas, cuentas de haberes-débi.tos, inventarios, etc.) han evolucionado hacia el software de

sistemas de información de gestión **(SIG)** que accede a una o más bases de datos que contienen información comercial.
Las aplicaciones en esta área reestructuran los datos existentes para facilitar las operaciones comerciales o gestionar la toma de decisiones. Además de las tareas convencionales de procesamientos de datos, las aplicaciones de software de gestión también realizan cálculo interactivo (por ejemplo: el procesamiento de transacciones en puntos de ventas).

Software de ingeniería y científíco. El software de ingeniería y científico está caracterizado por los algoritmos de «manejo de números». Las aplicaciones van desde la astronomía a la vulcanología, desde el análisis de la presión de los automotores a la dinámica orbital de las lanzaderas espaciales y desde la biología molecular a la fabricación automática. Sin embargo, las nuevas aplicaciones del área de ingeniería/ ciencia se han alejado de los algoritmos convencionales numéricos. El diseño asistido por computadora, la simulación de sistemas y otras aplicaciones interactivas, han comenzado a tomar características del software de tiempo real e incluso del software de sistemas.

Software empotrado. Los productos inteligentes se han convertido en algo común en casi todos los mercados de consumo e industriales. El software empotrado reside en memoria de sólo lectura y se utiliza para controlar productos y sistemas de los mercados industriales y de consumo. El software empotrado puede ejecutar funciones muy limitadas y curiosas (por ejemplo: el control de las teclas de un horno de microondas) o suministrar una función significativa y con capacidad de control (por ejemplo: funciones digitales en un automóvil, tales como control de la gasolina, indicadores en el salpicadero, sistemas de frenado, etc.).

Software de computadoras personales. El mercado del software de computadoras personales ha germinado en las últimas décadas. El procesamiento de textos, las hojas de cálculo, los gráficos por computadora, multimedia, entretenimientos, gestión de bases de datos, aplicaciones financieras, de negocios y personales y redes o acceso a bases de datos externas son algunas de los cientos de aplicaciones.

Software basado en Web. Las páginas Web buscadas por un explorador son software que incorpora instrucciones ejecutables (por ejemplo, CGI, HTML, Perl, o Java), y datos (por ejemplo, hipertexto y una variedad de formatos de audio y visuales). En esencia, la red viene a ser una gran computadora que proporciona un recurso software casi ilimitado que puede ser accedido por cualquiera con un modem.

Software de inteligencia artificial. El software de inteligencia artificial (IA) hace uso de algoritmos no numéricos para resolver problemas complejos para los que no son adecuados el cálculo o el análisis directo. Los sistemas expertos, también llamados sistemas basados en el conocimiento, reconocimiento de patrones (imágenes y voz), redes neuronales artificiales, prueba de teoremas, y los juegos son representativos de las aplicaciones de esta categoría.

La evolución del Software

Hoy en día el software tiene un doble papel. Es un producto y, al mismo tiempo, el vehículo para entregarlo.
Como producto, hace entrega de la potencia informática que incorpora el hardware informático o, más ampliamente, una red de computadoras que es accesible por hardware local. Si reside dentro de un teléfono celular u opera dentro de una computadora central, el software es un transformador de información, produciendo, gestionando, adquiriendo, modificando, mostrando o transmitiendo información que puede ser tan simple como un solo bit, o tan complejo como una presentación en multimedia. Como vehículo utilizado para hacer entrega del producto, el software actúa como la base de control de la computadora (sistemas operativos), la comunicación de información (redes) y la creación y control de otros programas (herramientas de software y entornos).
El papel del software informático ha sufrido un cambio significativo durante un periodo de tiempo superior
a 50 años. Enormes mejoras en rendimiento del hardware, profundos cambios de arquitecturas informáticas, grandes aumentos de memoria y capacidad de almacenamiento y una gran variedad de opciones de entrada y salida han conducido a sistemas más sofisticados y más complejos basados en computadora. La sofisticación y la complejidad pueden producir resultados deslumbrantes cuando un sistema tiene éxito, pero también pueden suponer grandes problemas para aquellos que deben construir sistemas complejos.
Libros populares publicados durante los años 70 y 80 proporcionan una visión histórica útil dentro de la percepción cambiante de las computadoras y del software, y de su impacto en nuestra cultura. Osborne hablaba de una «nueva revolución industriah. Toffler llamó a la llegada de componentes microelectrónicos la «tercera ola del cambio» en la historia de la humanidad, y Naisbitt predijo la transformación de la sociedad industrial a una «sociedad de información». Feigenbaum y McCorduck sugirieron que la información **y** el conocimiento (controlados por computadora) serían el foco de poder del siglo veintiuno, y Sto11 argumentó que la «comunidad electrónica» creada mediante redes y software es la clave para el intercambio de conocimiento alrededor del mundo.
Al comienzo de los años 90, Toffler describió un «cambio de poder» en el que las viejas estructuras de poder (gubernamentales, educativas, industriales, económicas y militares) se desintegrarían a medida que las computadoras y el software nos llevaran a la democratización del conocimiento.

A Yourdon [YOU92] le preocupaba que las compañías en Estados Unidos pudieran perder **su** competitividad en empresas relativas al software y predijo «el declive y la caída del programador americano». Hammer y Champy [HAM93] argumentaron que las tecnologías de información iban a desempeñar el papel principal en la areingeniería de la compañía».
A mediados de los años 90, la persistencia de las computadoras y del softwaregeneró una erupción de libros por «neo-Luddites» (por ejemplo: *Resisting the **Vi**rtual Life*, editado por James Brook y Ian Boal, y *The Future Does not Compute* de Stephen Talbot). Estos autores critican enormemente la computadora, haciendo énfasis en preocupaciones legítimas pero ignorando los profundos beneficios que se han llevado a cabo [LEV95].

Al final de los años 90, Yourdon [YOU96] volvió a evaluar las perspectivas del software profesional y sugirió la «resurrección y elevación» del programador americano.

A medida que internet creció en importancia, **su** cambio de pensamiento demostró ser correcto. Al final del siglo veinte, el enfoque cambió una vez más. Aquí tuvo lugar el impacto de la «bomba de relojería» Y2K (por ejemplo: [YOU98b], [DEJ98], [KAR99]). Aunque muchos vieron las predicciones de los críticos del Y2K como reacciones, sus populares lecturas devolvieron la difusión del software a sus vidas. Hoy en día, «la computación omnipresente» [NOR98] ha producido una generación de aplicaciones de información que tienen conexión en banda ancha a la Web para proporcionar «una capa de conexión sobre nuestras casas, oficinas, y autopistas» [LEV99]. El papel del software continúa su expansión.
El programador solitario de antaño ha sido reemplazado por un equipo de especialistas del software, cada uno centrado en una parte de la tecnología requerida para entregar una aplicación concreta. Y de este modo, las cuestiones que se preguntaba el programador solitario son las mismas cuestiones que nos preguntamos cuando construimos sistemas modernos basados en computadoras:

Gestión de Proyectos Software

La gestión de proyectos de software es una parte esencial de la ingeniería del software. La buena gestión no puede garantizar el éxito del proyecto. Sin embargo, la mala gestión usualmente lleva al fracaso del proyecto. El software es entregado tarde, los costes son mayores que los estimados y los requerimientos no se cumplen.
Los gestores de software son responsables de la planificación y temporalización del desarrollo de los proyectos. Supervisan el trabajo para asegurar que se lleva a cabo conforme a los estándares requeridos y supervisan el progreso para comprobar que el desarrollo se ajusta al tiempo previsto y al presupuesto. La administración de proyectos de software es necesaria debido a que la ingeniería de software profesional siempre está sujeta a restricciones organizacionales de tiempo y presupuesto. El trabajo del gestor de proyectos de software es asegurar que éstos cumplan dichas restricciones y entregar software que contribuya a las metas de la compañía de desarrollo software.
Los gestores de software hacen el mismo tipo de trabajo que otros gestores. Sin embargo, la ingeniería del software es diferente en varios aspectos a otros tipos, lo que hace a la gestión de software particularmente difícil.
Algunas de estas diferencias son las siguientes:

1. *El producto es intangible.* No se puede ver ni tocar. Los gestores de proyectos de software no pueden ver el progreso. Confían en otros para elaborar la documentación necesaria para revisar el progreso.

2. *No existen procesos del software estándar:* En las disciplinas de ingeniería con larga historia, el proceso se prueba y verifica. Para tipos particulares de sistemas, como puentes o edificios, el proceso de ingeniería se comprende bien. Sin embargo, los procesos de software varían notablemente de una organización a otra. A pesar de que la comprensión del proceso del software se ha desarrollado de forma significativa en los últimos años, aún no se puede predecir con certeza cuándo un proceso particular tiende a desarrollar problemas. Esto es especialmente cierto cuando el proyecto software es parte un proyecto de ingeniería de un sistema grande.

3. *A menudo los proyectos grandes son únicos.* Por lo general, los proyectos grandes de software son diferentes de proyectos previos. En consecuencia, los gestores, aun cuando cuenten con una amplia experiencia, ésta no es suficiente para anticipar los problemas.
Más aún, los rápidos cambios tecnológicos en tas computadoras y las comunicaciones hacen parecer obsoleta la experiencia previa. Las lecciones aprendidas en esas experiencias pueden no ser transferibles a los nuevos proyectos.

Debido a estos problemas, no es sorprendente que algunos proyectos de software se retrasen, sobrepasen el presupuesto y se entreguen fuera de tiempo. A menudo, los sistemas de software son nuevos y tecnológicamente innovadores. Frecuentemente los proyectos de ingeniería innovadores (como los nuevos sistemas de transporte) también tienen problemas de temporalización. Dadas las mezclas de dificultades, es notable que muchos proyectos de software sean entregados a tiempo y según lo presupuestado.

Actividades de gestión

Los gestores de proyectos son responsables de algunas o de la totalidad de las siguientes actividades:

- Redacción de la propuesta
- Planificación y calendarización del proyecto
- Estimación de costes del proyecto
- Supervisión y revisión del proyecto
- Selección y evaluación del personal
- Redacción y presentación de informes

La primera etapa de un proyecto de software implica redactar una propuesta para realizar ese proyecto. La propuesta describe los objetivos del proyecto y cómo se llevaría a cabo. Por lo general, incluye estimaciones de coste y tiempo y justifica por qué el contrato del proyecto se le debe dar a una organización o a un equipo en particular.
La redacción de la propuesta es una tarea crítica, ya que la existencia de muchas organizaciones de software depende de las propuestas aceptadas y los contratos asignados. No existen guías para esta tarea; la redacción de propuestas es una habilidad que se adquiere con la práctica y la experiencia.
La planificación de proyectos se refiere a la identificación de actividades, hitos y entregas de un proyecto. Por lo tanto, se debe bosquejar un plan para guiar el desarrollo hacia las metas del proyecto.
La estimación del coste es una actividad relacionada con la estimación de los recursos requeridos para llevar a cabo el plan del proyecto.
La supervisión de proyecto es una actividad continua. El gestor debe tener conocimiento del progreso del proyecto y comparar el progreso con los costes actuales y los planificados.
Aunque muchas organizaciones tienen mecanismos formales para supervisar, un gestor hábil podría formarse una imagen clara de lo que pasa llevando a cabo una entrevista informal con el personal del proyecto.

La supervisión informal predice problemas importantes del proyecto, y revela dificultades que pueden aparecer. Por ejemplo, las entrevistas diarias con el personal del proyecto pueden exteriorizar un problema en un fallo del software. Más que esperar un informe de atraso del proyecto, el gestor de software podría asignar algún experto para resolver el problema o podría decidir si se vuelve a programar.
Durante un proyecto, es normal tener varias revisiones formales de su gestión. Se hace la revisión completa del progreso y de los desarrollos técnicos del proyecto, y se tiene en cuenta el estado del proyecto junto con los propósitos de la organización que ha encargado el software.
El resultado de una revisión puede dar lugar a la cancelación del proyecto. El tiempo de desarrollo para un proyecto grande de software puede ser de varios años. Durante ese tiempo los objetivos organizacionales tienden obviamente a cambiar. Estos cambios pueden significar que el software ya no se necesita o que los requerimientos originales del proyecto son inapropiados.
La gestión puede decidir parar el desarrollo del software o cambiar el proyecto para adecuarlo a los cambios de los objetivos de la organización.
Por lo general, los gestores de proyectos tienen que seleccionar a las personas para trabajar en su proyecto. De forma ideal, habrá personal disponible con habilidades y experiencia apropiada para trabajar en el proyecto. Sin embargo, en muchos casos, los gestores tienen que establecer un equipo ideal mínimo para el proyecto. Las razones que explican esto son:

1. El presupuesto del proyecto no cubre la contratación de personal con sueldos altos. Se tiene que contratar personal con menos experiencia y menor sueldo.

2. El personal con experiencia apropiada no está disponible dentro o fuera de la organización. Es imposible reclutar nuevo personal para el proyecto. Dentro de la organización, los mejores trabajadores ya se han asignado a otros proyectos.

3. La organización desea desarrollar las habilidades de sus empleados. El personal inexperto puede ser asignado al proyecto para aprender y adquirir experiencia.

El gestor de software tiene que trabajar con estas restricciones al seleccionar al personal del proyecto. Sin embargo, todos estos problemas son probables a menos que exista un miembro del proyecto que cuente con algo de experiencia en el tipo de sistema a desarrollar. Sin esta experiencia, probablemente se cometerán muchos errores pequeños. En el Capítulo 25 se abordará la formación del equipo de desarrollo y la selección del personal.
Los gestores del proyecto son responsables de informar a los clientes y contratistas sobre el proyecto. Tienen que redactar documentos concisos y coherentes que resuman la información crítica de los informes detallados del proyecto. Les debe ser posible presentar esta información durante las revisiones de progreso. En consecuencia, comunicarse efectivamente de forma oral y escrita es una habilidad esencial que un gestor de proyectos debe tener.

Planificación del proyecto

La gestión efectiva de un proyecto de software depende de planificar completamente el progreso del proyecto. El gestor del proyecto debe anticiparse a los problemas que puedan surgir, así como preparar soluciones a esos problemas. Un plan, preparado al inicio de un proyecto, debe utilizarse como un conductor para el proyecto. Este plan inicial debe ser el mejor posible de acuerdo con la información disponible. Éste evolucionará conforme el proyecto progrese y la información sea mejor.
La figura siguiente refiere a los distintos planes propios del Ingeniero de Software.

Plan de calidad	**Describe los procedimientos y los estándares de calidad que se utilizarán en un proyecto. Véase el Capítulo 24.**
Plan de validación	**Describe el enfoque, los recursos y la programación utilizados para la validación del sistema.**
Plan de gestión de configuraciones	**Describe los procedimientos para la gestión de configuraciones y las estructuras a utilizar. Véase el Capítulo 29.**
Plan de mantenimiento	**Predice los requerimientos del mantenimiento del sistema, los costes del mantenimiento y el esfuerzo requerido. Véase el Capítulo 27.**
Plan de desarrollo del personal	**Describe cómo se desarrollan las habilidades y experiencia de los miembros del equipo del proyecto.**

El pseudocódigo que se muestra en la Figura *5.2* describe el proceso de planificación del proyecto para el desarrollo de software.

```
Establecer las restricciones del proyecto
Hacer la valoración inicial de los parámetros del proyecto
Definir los hitos del proyecto y productos a entregar
Mientras el proyecto no se haya completado o cancelado repetir
        Bosquejar la programación en el tiempo del proyecto
        Iniciar actividades acordes con la programación
        Esperar (por un momento)
        Revisar el progreso del proyecto
        Revisar las estimaciones de los parámetros del proyecto
        Actualizar la programación del proyecto
        Renegociar las restricciones del proyecto y los productos a entregar
        Si (surgen problemas) entonces
            Iniciar la revisión técnica y la posible revisión
        fin de si
fin de repetir
```

El plan del proyecto

El plan del proyecto fija los recursos disponibles, divide el trabajo y crea un calendario de trabajo. En algunas organizaciones, el plan del proyecto es un único documento que incluye todos los diferentes tipos de planes. En otros casos, este plan sólo se refiere al proceso de desarrollo. Otros pueden estar referenciados, pero son proporcionados por separado.

El plan que se describe aquí tiene que ver con el último tipo de plan mencionado. Los detalles de este plan varían dependiendo del tipo de proyecto y de la organización. Sin embargo, muchos planes incluyen las siguientes secciones:

1. *Introducción*. Describe brevemente los objetivos del proyecto y expone las restricciones (por ejemplo, presupuesto, tiempo, etcétera) que afectan a la gestión del proyecto.

2. *Organización del proyecto*. Describe la forma en que el equipo de desarrollo está organizado, la gente involucrada y sus roles en el equipo.

3. *Análisis de riesgo*. Describe los posibles riesgos del proyecto, la probabilidad de que surjan estos riesgos y las estrategias de reducción de riesgos propuestas. La gestión de riesgos se estudia en la Sección 5.4.

4. *Requerimientos de recursos de hardware y software*. Describe el hardware y el software de ayuda requeridos para llevar a cabo el desarrollo. Si es necesario comprar hardware, se deben incluir las estimaciones de los precios y las fechas de entrega.

5. *División del trabajo*. Describe la división del proyecto en actividades e identifica los hitos y productos a entregar asociados con cada actividad.

6. *Programa del proyecto*. Describe las dependencias entre actividades, el tiempo estimado requerido para alcanzar cada hito y la asignación de la gente a las actividades.

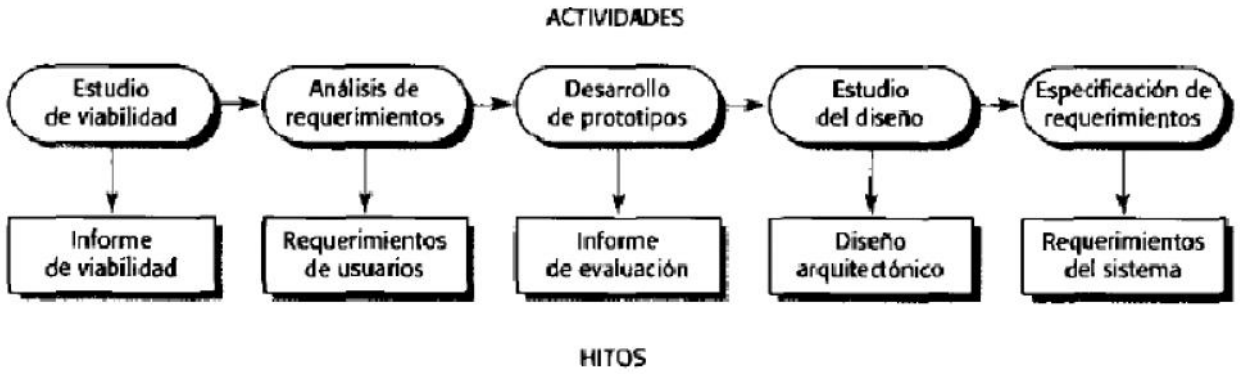
7. *Mecanismos de supervisión e informe*. Describe la gestión de informes y cuándo deben producirse, así como los mecanismos de supervisión del proyecto a utilizar.

El plan del proyecto debe revisarse regularmente durante el proyecto. Algunas partes, como el calendario del proyecto, cambiarán frecuentemente; otras serán más estables. Para simplificar las revisiones, se debe organizar el documento en secciones separadas que permitan su reemplazo de forma individual conforme evoluciona el plan.

Hitos y entregas

Los gestores necesitan información para hacer su trabajo. Como el software es intangible, esta información sólo se puede proveer como documentos que describan el estado del software que se está desarrollando. Sin esta información, es imposible juzgar el progreso y no se pueden actualizar los costes y calendarios.
Cuando se planifica un proyecto, se debe establecer una serie de *hitos* —puntos finales de una actividad del proceso del software—. En cada uno, debe existir una salida formal, como un informe, que se debe presentar al gestor. Los informes de hitos no deben ser documentos amplios. Deben ser informes cortos de los logros en una actividad del proyecto. Los hitos deben representar el fin de una etapa lógica en el proyecto. Los hitos

indefinidos como «80 % del código completo» son imposibles de validar y carecen de utilidad para la gestión del proyecto.
No podemos validar si se ha llegado a esta etapa debido a que la cantidad de código que se tiene que desarrollar no es precisa.
Una entrega es el resultado del proyecto que se entrega al cliente. De forma general, se entrega al final de una fase principal del proyecto como la especificación, el diseño, etcétera.
Como regla general, las entregas son hitos, pero éstos no son necesariamente entregas. Dichos hitos pueden ser resultados internos del proyecto que son utilizados por el gestor del proyecto para verificar el progreso del mismo pero que no se entregan al cliente.
Para establecer los hitos, el proceso del software debe dividirse en actividades básicas con sus salidas asociadas. Por ejemplo, la siguiente figura muestra las actividades involucradas en la especificación de requerimientos cuando se utiliza la construcción de prototipos para ayudar a validar dichos requerimientos. También se muestran las salidas principales de cada actividad (los hitos del proyecto). Las entregas del proyecto, las cuales son entregadas al cliente, son la definición y especificación de requerimientos.



Calendarización del proyecto

Ésta es una de las tareas más difíciles para los gestores de proyectos. Los gestores estiman el tiempo y los recursos requeridos para completar las actividades y organizarías en una sucesión coherente. A menos que el proyecto a calendarizar sea similar a otro anterior, las estimaciones previas son una base incierta para la calendarización del nuevo proyecto. La estimación del calendario se complica más por el hecho de que proyectos diferentes pueden utilizar métodos de diseño y lenguajes de implementación diferentes.
Si el proyecto es técnicamente complejo, las estimaciones iniciales casi siempre son optimistas aun cuando los gestores traten de considerar las eventualidades. A este respecto, la calendarización del tiempo para la creación del software no es diferente a la de cualquier otro tipo de proyecto grande y complejo.
Los calendarios se deben actualizar continuamente en la medida que se disponga de mejor información acerca del progreso.
La calendarización del proyecto (véase la figura siguiente) implica separar todo el trabajo de un proyecto en actividades complementarias y considerar el tiempo requerido para completar dichas actividades. Por lo general, algunas de éstas se llevan a cabo en paralelo. Debemos coordinar estas actividades paralelas y organizar el trabajo para que

la mano de obra se utilice de forma óptima. Deben evitarse situaciones en que el proyecto entero se retrase debido a que no se ha terminado una actividad crítica.

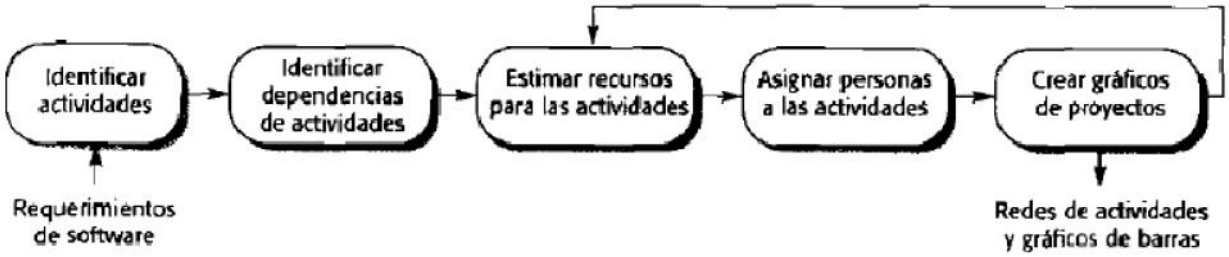


Figura 5.4 El proceso de calendarización del proyecto.

Normalmente, las actividades del proyecto deben durar por lo menos una semana. Hacer subdivisiones más finas significa invertir una cantidad desproporcionada de tiempo en la estimación y revisión de tablas. También es útil asignar una cantidad de tiempo máxima de 8 a 10 semanas para realizar cualquier actividad. Si lleva más tiempo, se deben hacer subdivisiones.
Al estimar la calendarización, los gestores no deben suponer que cada etapa del proyecto estará libre de problemas. Las personas que trabajan en él pueden enfermarse o renunciar, el hardware puede fallar y el software o hardware de soporte puede llegar tarde. Si el proyecto es nuevo y técnicamente complejo, ciertas partes podrían ser más complicadas y llevarían más tiempo del que se estimó originalmente.
Como en los calendarios, los gestores deben estimar los recursos necesarios para completar cada tarea. El recurso principal es el esfuerzo humano que se requiere. Otros recursos pueden ser el espacio en disco requerido en un servidor, el tiempo requerido de hardware especializado, un simulador o el presupuesto para viajes del personal del proyecto.
Una buena regla práctica es estimar como si nada fuera a salir mal, y entonces incrementar la estimación para abarcar los problemas previstos. Con este mismo fin, a la estimación se le debe agregar un factor de contingencia adicional. Este factor extra de contingencia depende del tipo de proyecto, de los parámetros del proceso (fecha de entrega, estándares, etcétera) y de la calidad y experiencia de los ingenieros de software que trabajen en el proyecto.
Como regla, para los problemas previstos siempre debe agregarse un 30 % a la estimación original y otro 20 % para cubrir algunas cosas no previstas. Por lo general, el calendario del proyecto se representa como un conjunto de gráficos que muestran la división del trabajo, las dependencias de las actividades y la asignación del personal. Esto se aborda en la siguiente sección.
Por lo general, las herramientas de gestión de software, se utilizan para automatizar la producción de diagramas.

Gráficos de barras y redes de actividades

Los gráficos de barras (Gantt) y las redes de actividades (PERT) son notaciones gráficas que se utilizan para ilustrar la calendarización del proyecto. Los gráficos de barras muestran quién es responsable de cada actividad y cuándo debe comenzar y finalizar ésta. Las redes de actividades muestran las dependencias entre las diferentes actividades

que conforman un proyecto. Los gráficos de barras y las redes de actividades se generan automáticamente a partir de una base de datos de la información del proyecto utilizando una herramienta de gestión de proyectos.

Gestión de riesgos

Una tarea importante del gestor de proyectos 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. Los resultados de este análisis de riesgos se deben documentar a lo largo del plan del proyecto junto con el análisis de consecuencias cuando el riesgo ocurra. Identificar éstos y crear planes para minimizar sus efectos en el proyecto se llama gestión de riesgos.
De forma simple, se puede concebir un riesgo como una probabilidad de que una circunstancia adversa ocurra. Los riesgos son una amenaza para el proyecto, para el software que se está desarrollando y para la organización.
Estas categorías de riesgos se definen como se muestra a continuación:

1. *Riesgos del proyecto.* Estos afectan la calendarización o los recursos del proyecto. Un ejemplo podría ser la pérdida de un diseñador experimentado.

2. *Riesgos del producto.* Éstos afectan a la calidad o al rendimiento del software que se está desarrollando. Un ejemplo podría ser que el rendimiento en un componente que hemos comprado sea menor que el esperado.

3. *Riesgos del negocio.* Estos afectan a la organización que desarrolla o suministra el software. Por ejemplo, que un competidor introduzca un nuevo producto es un riesgo de negocio.

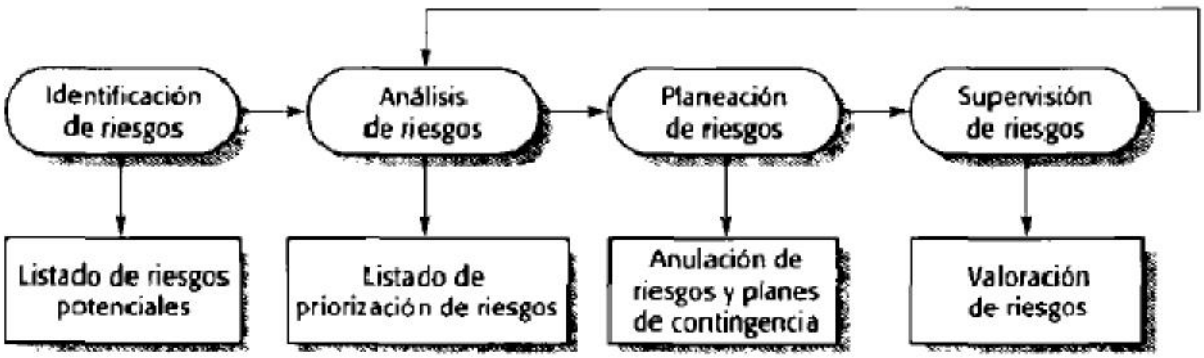
Estos tipos no son exclusivos entre sí. Si un programador experto abandona el proyecto, esto es un riesgo para el proyecto (debido a que la entrega del sistema se puede retrasar), para el producto (debido a que un sustituto puede no ser tan experto y cometer muchos errores) y para el negocio (debido a que esa experiencia puede no contribuir a negocios futuros).
Los riesgos que pueden afectar a un proyecto dependen del propio proyecto y del entorno organizacional donde se desarrolla. Sin embargo, muchos riesgos son universales. La figura siguiente muestra algunos de estos riesgos.

Rotación de personal	Proyecto	Personal con experiencia abandona el proyecto antes de que finalice.
Cambio de gestión	Proyecto	Habrá un cambio de gestión organizacional con diferentes prioridades.
No disponibilidad del hardware	Proyecto	El hardware esencial para el proyecto no será entregado a tiempo.
Cambio de requerimientos	Proyecto y producto	Habrá más cambios en los requerimientos de lo esperado.
Retrasos en la especificación	Proyecto y producto	Las especificaciones de las interfaces esenciales no estarán a tiempo.
Subestimación del tamaño	Proyecto y producto	El tamaño del sistema se ha subestimado.
Bajo rendimiento de la herramienta CASE	Producto	Las herramientas CASE que ayudan al proyecto no tienen el rendimiento esperado.
Cambio de tecnología	Negocio	Un producto competitivo se pone en venta antes de que el sistema se complete.
Competencia del producto	Negocio	La tecnología fundamental sobre la que se construirá el sistema se sustituye por nueva tecnología.

Figura 5.9 Posibles riesgos del software.

La gestión de riesgos es importante particularmente para los proyectos de software debido a las incertidumbres inherentes con las que se enfrentan muchos proyectos. Estas incertidumbres son el resultado de los requerimientos ambiguamente definidos, las dificultades en la estimación de tiempos y los recursos para el desarrollo del software, la dependencia en las habilidades individuales, y los cambios en los requerimientos debidos a los cambios en las necesidades del cliente. Es preciso anticiparse a los riesgos: comprender el impacto de éstos en el proyecto, en el producto y en el negocio, y considerar los pasos para evitarlos. En el caso de que ocurran, se deben crear planes de contingencia para que sea posible aplicar acciones de recuperación.
La Figura muestra el proceso de gestión de riesgos. Este comprende varias etapas:

1. *Identificación de riesgos.* Identificar los posibles riesgos para el proyecto, el producto y los negocios.
2. *Análisis de riesgos.* Valorar las probabilidades y consecuencias de estos riesgos.
3. *Planificación de riesgos.* Crear planes para abordar los riesgos, ya sea para evitarlos o minimizar sus efectos en el proyecto.
4. *Supervisión de riesgos.* Valorar los riesgos de forma constante y revisar los planes para la mitigación de riesgos tan pronto como la información de los riesgos esté disponible.



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 de gestión de riesgos se deben documentar en un plan de gestión de riesgos. Este debe incluir un estudio de los riesgos a los que se enfrenta el proyecto, un análisis de éstos y los planes requeridos para su gestión. Si es necesario, puede incluir algunos resultados de la gestión de riesgos; por ejemplo, planes específicos de contingencia que se activan si aparecen dichos riesgos.

Identificación de riesgos

Ésta es la primera etapa de la gestión de riesgos. Comprende el descubrimiento de los posibles riesgos del proyecto. En principio, no hay que valorarlos o darles prioridad en esta etapa aunque, en la práctica, por lo general no se consideran los riesgos con consecuencias menores o con baja probabilidad.
Esta identificación se puede llevar a cabo a través de un proceso de grupo utilizando un enfoque de tormenta de ideas o simplemente puede basarse en la experiencia. Para ayudar al proceso, se utiliza una lista de posibles tipos de riesgos. Hay al menos seis tipos de riesgos que pueden aparecer:

1. *Riesgos de tecnología.* Se derivan de las tecnologías de software o de hardware utilizadas en el sistema que se está desarrollando.

2. *Riesgos de personal.* Riesgos asociados con las personas del equipo de desarrollo.

3. *Riesgos organizacionales.* Se derivan del entorno organizacional donde el software se está desarrollando.

4. *Riesgos de herramientas.* Se derivan de herramientas CASE y de otro software de apoyo utilizado para desarrollar el sistema.

5. *Riesgos de requerimientos*. Se derivan de los cambios de los requerimientos del cliente y el proceso de gestionar dicho cambio.

6. *Riesgos de estimación*. Se derivan de los estimados administrativos de las características del sistema y los recursos requeridos para construir dicho sistema.

La figura siguiente proporciona algunos ejemplos de riesgos posibles en cada una de estas categorías. El resultado de este proceso debe ser una larga lista de riesgos que podrían presentarse y afectar al producto, al proceso o al negocio.

Tecnología	La base de datos que se utiliza en el sistema no puede procesar muchas transacciones por segundo como se esperaba. Los componentes de software que deben reutilizarse contienen defectos que limitan su funcionalidad.
Personal	Es imposible reclutar personal con las habilidades requeridas para el proyecto. El personal clave está enfermo y no disponible en momentos críticos. La capacitación solicitada para el personal no está disponible.
Organizacional	La organización se reestructura de tal forma que una gestión diferente se responsabiliza del proyecto. Los problemas financieros de la organización fuerzan a reducciones en el presupuesto del proyecto.
Herramientas	Es ineficiente el código generado por las herramientas CASE. Las herramientas CASE no se pueden integrar.
Requerimientos	Se proponen cambios en los requerimientos que requieren rehacer el diseño. Los clientes no comprenden el impacto de los cambios en los requerimientos.
Estimación	El tiempo requerido para desarrollar el software está subestimado. La tasa de reparación de defectos está subestimada. El tamaño del software está subestimado.

Análisis de riesgos

Durante este proceso, se considera por separado cada riesgo identificado y se decide acerca de la probabilidad y la seriedad del mismo. No existe una forma fácil de hacer esto —recae en la opinión y experiencia del gestor del proyecto—. No se hace una valoración con números precisos sino en intervalos:

• La probabilidad del riesgo se puede valorar como muy bajo (< 10%), bajo (10-25%), moderado (25-50%), alto (50-75%) o muy alto (>75%).
• Los efectos del riesgo pueden ser valorados como catastrófico, serio, tolerable o insignificante.

El resultado de este proceso de análisis se debe colocar en una tabla, la cual debe estar ordenada según la seriedad del riesgo.
La figura siguiente ilustra este proceso de análisis.
Obviamente, aquí es arbitraria la valoración de la probabilidad y seriedad. En la práctica, para hacer esta valoración se necesita información detallada del proyecto, el proceso, el equipo de desarrollo y la organización.
Por supuesto, tanto la probabilidad como la valoración de los efectos de un riesgo cambian conforme se disponga de mayor información acerca del riesgo y los planes de

gestión del mismo se implementen. Por lo tanto, esta tabla se debe actualizar durante cada iteración del proceso de riesgos.
Una vez que los riesgos se hayan analizado y clasificado, se debe discernir cuáles son los más importantes que se deben considerar durante el proyecto. Este discernimiento debe depender de una combinación de la probabilidad de aparición del riesgo y de los efectos del mismo.

Los problemas financieros de la organización fuerzan a reducir el presupuesto del proyecto.	Baja	Catastrófico
Es imposible reclutar personal con las habilidades requeridas para el proyecto.	Alta	Catastrófico
El personal clave está enfermo y no disponible en momentos críticos.	Moderada	Serio
Los componentes de software que deben reutilizarse contienen defectos que limitan su funcionalidad.	Moderada	Serio
Se proponen cambios en los requerimientos que requieren rehacer el diseño.	Moderada	Serio
La organización se reestructura de tal forma que cambia el grupo de gestión.	Alta	Serio
La base de datos que se utiliza en el sistema no puede procesar muchas transacciones por segundo como se esperaba.	Moderada	Serio
El tiempo requerido para desarrollar el software está subestimado.	Alta	Serio
Las herramientas CASE no se pueden integrar.	Alta	Tolerable
Los clientes no comprenden el impacto de los cambios en los requerimientos.	Moderada	Tolerable
La capacitación solicitada para el personal no está disponible.	Moderada	Tolerable
La tasa de reparación de defectos está subestimada.	Moderada	Tolerable
El tamaño del software está subestimado.	Alta	Tolerable
Es ineficiente el código generado por las herramientas CASE.	Moderada	Insignificante

Figura 5.12 Análisis de riesgos.

En general, siempre se deben tener en cuenta todos los riesgos catastróficos, así como todos los riesgos serios que tienen más que una moderada probabilidad de ocurrir. Boehm (Boehm, 1988) recomienda identificar y supervisar los «10 riesgos más altos», pero este número parece demasiado arbitrario. El número exacto de riesgos a supervisar debe depender del proyecto. Pueden ser cinco o 15. No obstante, el número apropiado debe ser manejable. Un número muy grande de riesgos requiere obtener mucha información.

Planificación de riesgos

El proceso de planificación de riesgos considera cada uno de los riesgos clave que han sido identificados, así como las estrategias para gestionarlos. Otra vez, no existe un proceso sencillo que nos permita establecer los planes de gestión de riesgos. Depende del juicio y de la experiencia del gestor del proyecto. La Figura siguiente muestra posibles estrategias para los riesgos que han sido identificados en la figura anterior.

Problemas financieros de la organización	Preparar un documento breve para el gestor principal que muestre que el proyecto hace contribuciones muy importantes a las metas del negocio.
Problemas de reclutamiento	Alertar al cliente de las dificultades potenciales y los posibles retrasos, investigar la compra de componentes.
Enfermedad de personal	Reorganizar el equipo de tal forma que haya solapamiento en el trabajo y las personas comprendan el de los demás.
Componentes defectuosos	Reemplazar los componentes defectuosos con los comprados de fiabilidad conocida.
Cambios de los requerimientos	Rastrear la información para valorar el impacto de los requerimientos, maximizar la información oculta en ellos.
Reestructuración organizacional	Preparar un documento breve para el gestor principal que muestre que el proyecto hace contribuciones muy importantes a las metas del negocio.
Rendimiento de la base de datos	Investigar la posibilidad de comprar una base de datos de alto rendimiento.
Tiempo de desarrollo subestimado	Investigar los componentes comprados y la utilización de un generador de programas.

Figura 5.13 Estrategia de gestión de riesgos.

Estas estrategias seguidas pueden dividirse en tres categorías.

1. *Estrategias de prevención.* Siguiendo estas estrategias, la probabilidad de que el riesgo aparezca se reduce. Un ejemplo de este tipo de estrategias es la estrategia de evitación de defectos en componentes de la Figura.

2. *Estrategias de minimización.* Siguiendo estas estrategias se reducirá el impacto del riesgo. Un ejemplo de esto es la estrategia frente a enfermedad del personal de la Figura.

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 el mostrado en la Figura con la estrategia para problemas financieros.

Puede verse aquí una analogía con las estrategias utilizadas en sistemas críticos para asegurar fiabilidad, protección y seguridad. Básicamente, es mejor usar una estrategia para evitar el riesgo. Si esto no es posible, utilizar una para reducir los efectos serios de los riesgos.
Finalmente, tener estrategias para reducir el impacto del riesgo en el proyecto y en el producto.

Supervisión de riesgos

La supervisión de riesgos normalmente valora cada uno de los riesgos identificados para decidir si éste es más o menos probable y si han cambiado sus efectos. Por supuesto, esto no se puede observar de forma directa, por lo que se tienen que buscar otros factores para dar indicios de la probabilidad del riesgo y sus efectos. Obviamente, estos factores

dependen de los tipos de riesgo. La Figura siguiente da algunos ejemplos de los factores que ayudan en la valoración de estos tipos de riesgos.

Tecnología	Entrega retrasada del hardware o de la ayuda al software, muchos problemas tecnológicos reportados.
Personal	Baja moral del personal, malas relaciones entre los miembros del equipo, disponibilidad de empleo.
Organizacional	Chismorreo organizacional, falta de acciones por el gestor principal.
Herramientas	Rechazo de los miembros del equipo para utilizar herramientas, quejas acerca de las herramientas CASE, peticiones de estaciones de trabajo más potentes.
Requerimientos	Peticiones de muchos cambios en los requerimientos, quejas del cliente.
Estimación	Fracaso en el cumplimiento de los tiempos acordados, y en la eliminación de defectos reportados.

Figura 5.14 Factores de riesgo.

La supervisión de riesgos debe ser un proceso continuo y, en cada revisión del progreso de gestión, cada uno de los riesgos clave debe ser considerado y analizado por separado.