5 EL CICLO DE VIDA DEL PROYECTO

Todo error humano es impaciencia, una renunciación prematura al método, una engañosa sujeción a un engaño.

Franz Kafka, Cartas

En este capítulo se aprenderá:

1. El concepto del ciclo de vida de un proyecto.
2. Las características del ciclo de vida de un proyecto clásico.
3. Las diferencias entre proyectos clásicos y semiestructurados.
4. Los componentes del ciclo de vida estructurado.
5. Las diferencias entre ciclos de vida radicales y conservadores.

Para ser un buen analista de sistemas se requiere más que simples herramientas de modelado; necesitamos *métodos*. En la profesión de desarrollo de sistemas, los términos "método", "metodología", "ciclo de vida del proyecto" y "ciclo de vida del desarrollo de sistemas" se usan de manera casi indistinta. En la parte III veremos métodos detallados de cómo efectuar el análisis de sistemas, en el contexto más amplio de un método –conocido como ciclo de vida del proyecto estructurado–, para llevar a cabo el desarrollo global de un sistema nuevo.

Antes de presentar el ciclo de vida del proyecto estructurado, es importante examinar el ciclo de vida del proyecto clásico que se trata en muchos textos y se utiliza en muchas organizaciones para el desarrollo de sistemas hoy en día, sobre todo

para identificar sus limitaciones y puntos débiles. Después de este examen haremos una breve exposición acerca del ciclo de vida del proyecto *semiestructurado:* un ciclo de vida de proyecto que incluye algunos, pero no todos los elementos del desarrollo moderno de sistemas. En seguida se presentará el ciclo de vida del proyecto *estructurado,* mostrando una visión global de las principales actividades y de cómo se interrelacionan. Por último, se verán brevemente el ciclo de vida *formador de prototipos* que popularizaron Bernard Boar, James Martin, y varios vendedores de lenguajes de programación de cuarta generación.

También exploraremos el concepto de *desarrollo iterativo* o *descendente.* En particular, se presentarán los conceptos de desarrollo iterativo *radical* y desarrollo iterativo *conservador.* Dependiendo de la naturaleza de un proyecto de desarrollo de sistemas, puede haber razones válidas para adoptar un método en lugar de otro, e incluso algunos proyectos pudieran requerir una combinacion de ambos.

5.1 EL CONCEPTO DE CICLO DE VIDA DE UN PROYECTO

Como pudiera esperarse, las organizaciones pequeñas de proceso de datos tienden a ser relativamente informales: los proyectos de desarrollo de sistemas nacen de conversaciones entre el usuario y el administrador del proyecto (que puede ser a la vez el analista, el programador, el operario y el conserje), y el proyecto procede desde el análisis hasta el diseño e implantación sin mayor alboroto.

Sin embargo, en las organizaciones más grandes, las cosas se llevan a cabo de manera mucho más formal. La comunicación entre los usuarios, la administración y el equipo del proyecto suele ser por escrito, y todo mundo entiende que el proyecto pasará por diversas fases antes de completarse. Aun así, es sorprendente ver la gran diferencia entre las maneras en que dos administradores afrontan sus respectivos proyectos. De hecho, a menudo se deja a discreción del administrador determinar las fases y actividades de su proyecto, y cómo se llevarán a cabo.[1]

Recientemente, sin embargo, ha empezado a cambiar el enfoque que se le da al desarrollo de sistemas. Cada vez son más las organizaciones grandes y pequeñas que están adoptando un ciclo de vida uniforme y único para sus proyectos. Esto a veces se conoce como el plan del proyecto, la metodología del desarrollo del sistema o, simplemente, "la forma en la que hacemos las cosas aquí". El manual del ciclo de vida del proyecto suele ser un libro tan voluminoso como el compendio de normas, que yace (usualmente sin ser leído) sobre el escritorio de todo analista y

1 Esto suena como si la anarquía prevaleciera en la mayoría de las organizaciones de proceso electrónico de datos. Sin embargo, hay dos situaciones comunes que llevan a este enfoque individualista aun en la organización más ejemplar: 1) La organización altamente descentralizada, donde cada departamento tiene su grupo de proceso electrónico de datos con sus propias normas locales y 2) el periodo de varios años tras de que el último "ciclo de vida oficial del proyecto" se juzgara inútil y se descartara.

programador. Ese manual ofrece un procedimiento común a seguir para desarrollar un sistema computacional que puede orientar a cualquier miembro de la organización de desarrollo de sistemas.

El enfoque puede ser casero o, como alternativa, pudiera ser que la organización para el desarrollo de sistemas decida comprar un paquete de administración de proyectos y ajustarlo a las necesidades de la compañía.[2] Parece obvio que, aparte de darle empleo a las personas que crean los manuales de ciclo de vida de los proyectos (y a aquellos que escriben libros de texto al respecto), es conveniente la metodología del proyecto. ¿De qué sirve entonces tener un ciclo de vida de un proyecto? Existen tres objetivos principales:

1. Definir las actividades a llevarse a cabo en un proyecto de desarrollo de sistemas.
2. Lograr congruencia entre la multitud de proyectos de desarrollo de sistemas en una misma organización.
3. Proporcionar puntos de control y revisión administrativos de las decisiones sobre continuar o no con un proyecto.

El primer objetivo es de particular importancia en una organización grande donde constantemente está ingresando personal nuevo a los puestos de administración de proyectos. El administrador novato pudiera no tomar en cuenta o subestimar la importancia de fases clave del proyecto si se basa sólo en su intuición. De hecho, pudiera suceder que los programadores y analistas de bajo rango no entiendan dónde y cómo encajan sus esfuerzos individuales en el proyecto global, a menos que se les dé una descripción adecuada de *todas* las fases del proyecto.

El segundo objetivo también es importante en una organización grande. Para los niveles más altos de la administración pudiera ser bastante confuso seguir la pista de cientos de proyectos diferentes, cada uno de los cuales se lleva a cabo de distinta manera. Por ejemplo, si el proyecto A define la actividad de análisis de sistemas de diferente forma que el proyecto B y el B no incluye una fase de diseño, ¿cómo puede darse cuenta el administrador de segundo o tercer nivel de cuál proyecto se está rezagando y cuál continúa según lo previsto?[3]

2 Existen varios de estos paquetes en el mercado, que cuestan entre $10,000 y $100,000 dólares estadounidenses (o su equivalente en moneda nacional), o más. Algunos de los ejemplos más conocidos son Spectrum (de Spectrum International Corp.), SDM-70 (de AGS Software), y Method/1 (de Arthur Andersen). No comentaré acerca de ningún paquete de administración de proyectos en particular; sólo le sugiero que tenga en mente los conceptos presentados en este libro si su organización utiliza un paquete obtenido en el mercado.

3 Miller en [Miller, 1978], señala que éste es un fenómeno comúnmente observado; de hecho, lo presenta como una "hipótesis" general aplicable a todos los sistemas en activo:

> HIPOTESIS 2-1: Los componentes de un sistema incapaces de asociarse, o que carecen de la experiencia que haya formado tales asociaciones, deben funcionar de acuerdo con una programación rígida o con reglas de operación altamente estandarizadas. Se sigue que si la rotación de los componentes rebasa el ritmo con que se están desarrollando las asociaciones necesarias para su operación, aumenta la rigidez en la programación.

El tercer objetivo de un ciclo de vida de proyecto normal se refiere a la necesidad de la administración de controlar un proyecto. En los proyectos triviales, el único punto de revisión probablemente esté al final del proyecto: ¿se concluyó a tiempo y centro de los márgenes del presupuesto acordado? (o, más simple aún, ¿se concluyó siquiera?) ¿Y cumplió con los requisitos del usuario? Pero, para proyectos más grandes, debería contarse con varios puntos intermedios de revisión, que permitieran determinar si el proyecto se estuviera retrasando o si fueran necesarios recursos adicionales. Además, el usuario inteligente también necesitará puntos de revisión en diversas etapas del proyecto para que pueda determinar si quiere continuar financiándolo.[4]

Dicho todo esto, no queda más que subrayar que el ciclo de vida del proyecto definitivamente no está a cargo del proyecto; no le evitará al administrador del proyecto la difícil labor de tomar decisiones, sopesar alternativas, librar batallas políticas, negociar con usuarios recalcitrantes, animar a programadores deprimidos, ni ninguna de las demás tribulaciones relacionadas con los proyectos. El administrador del proyecto todavía tiene que *administrar,* en todo el sentido de la palabra. La única ayuda que puede proporcionar el ciclo de vida del proyecto es que pueda *organizar* las actividades del administrador, aumentando la probabilidad de que se aborden los problemas pertinentes en el momento adecuado.

5.2 EL CICLO DE VIDA DEL PROYECTO CLASICO

El tipo de ciclo de vida de proyecto que se usa actualmente en la mayoría de las organizaciones difiere de aquel al que estaremos dedicando la mayor parte de nuestra atención en la parte III. En la figura 5.1 se muestra el ciclo de vida clásico o convencional. Cada proyecto atraviesa por algún tipo de análisis, diseño e implantación, aunque no se haga exactamente como se muestra en el diagrama. El ciclo de vida de proyecto utilizado, por ejemplo, en la organización de usted, pudiera diferir del que se muestra en la figura 5.1 en una o en todas las formas siguientes:

4 De hecho, los procedimientos de la mayor parte de los proyectos de proceso de datos son tales que existe sólo un punto de control desde el cual el usuario tiene una manera obvia y limpia de arrepentirse: al final de la fase de encuesta o del estudio de factibilidad. En teoría, sin embargo, el usuario debería tener la oportunidad de cancelar un proyecto de proceso de datos al final de cualquier fase si piensa que está desperdiciando su dinero.

Figura 5.1(a): **El ciclo de vida del proyecto clásico**

- Las fases de exploración y análisis pudieran juntarse en una sola (sobre todo en organizaciones en las cuales se considera factible desde el inicio cualquier cosa que quiera el usuario).
- Puede no haber fase de estudio de hardware si se cree que cualquier sistema nuevo pudiera instalarse con las computadoras existentes sin causar mayor problema operacional.
- Las fases de diseño preliminar y de diseño de detalles pudieran juntarse en una sola llamada simplemente de diseño.
- Diversas fases de prueba pueden juntarse en una sola; de hecho, podrían incluirse con la codificación.

De aquí que el ciclo de vida del proyecto en una organización sola puede tener cinco fases o siete o doce, pero seguir siendo todavía de tipo clásico.

¿Qué es lo que realmente caracteriza el ciclo de vida de un proyecto como clásico? Se distinguen dos aspectos: una fuerte tendencia a la implantación ascendente del sistema y la insistencia en la progresión lineal y secuencial de una fase a la siguiente.

5.2.1 Implantación ascendente

El uso de la implantación ascendente es una de las grandes debilidades del ciclo de vida de los proyectos clásicos. Como se podrá ver en la figura 5.1(a), se espera que los programadores lleven a cabo primero sus pruebas modulares, luego las pruebas del subsistema, y finalmente las pruebas del sistema mismo. Este enfoque también se conoce como el ciclo de vida de cascada, y está basado en el diagrama presentado originalmente en [Royce, 1970], y popularizado posteriormente por Barry Boehm [Boehm, 1981]. Se muestra en la figura 5.1(b).

No está claro de dónde surgió originalmente este enfoque, pero pudiera haberse tomado de las líneas de montaje de las industrias manufactureras. La implantación ascendente es buena para el montaje de automóviles en línea, *¡pero sólo después de que el prototipo esté completamente libre de fallas!*[5] Desafortunadamente, muchas organizaciones que desarrollan sistemas todavía producen sistemas únicos, para lo cual el enfoque ascendente presenta un gran número de dificultades serias:

---

5 Muchos creen que el enfoque ascendente pudiera provenir de la incustria computacional del *hardware* porque muchos de los programadores y administradores de programación de los años 50 y 60 eran ingenieros eléctrónicos que habían tenido que ver previamente con el desarrollo de hardware.

- Nada está hecho hasta que *todo* esté terminado. Por eso, si el proyecto se atrasa y la fecha límite cae precisamente en medio del proceso de prueba del sistema, no habrá nada que mostrarle al usuario más que una enorme pila de listados de programas, los cuales, vistos en su totalidad, no le ofrecen nada de valor.

- Las fallas más triviales se encuentran al comienzo del periodo de prueba y las más graves al final. Esto es casi una tautología: las pruebas modulares dejan al descubierto fallas relativamente simples dentro de los módulos individuales. Las pruebas del sistema, por otra parte, descubren errores grandes de interfaz entre subsistemas. La cuestión es que los errores de interfaz no son lo que el programador desea descubrir al final de un proyecto de desarrollo; tales fallas pueden obligar a la recodificación de un gran número de módulos, y pueden tener un impacto devastador sobre el calendario, justo en un momento en el cual es probable que todo el mundo esté algo cansado y molesto tras haber trabajado duro durante tantos meses.

- La eliminación de fallas suele ser extremadamente difícil durante las últimas etapas de prueba del sistema. Nótese que se puede distinguir entre *pruebas* y *eliminación de fallas*. La eliminación de fallas es el arte de descubrir dónde está la falla (y subsecuentemente cómo arreglarla) después de que el proceso de prueba ha determinado que la falla de hecho existe. Cuando la falla se descubre durante la fase de prueba del sistema en un proyecto ascendente, a menudo suele ser extremadamente difícil determinar cuál módulo la contiene; pudiera tratarse de cualquiera de los cientos (o miles) de módulos que se han combinado por primera vez. Localizar una falla a menudo es como hallar una aguja en un pajar.

- La necesidad de prueba con la computadora aumenta exponencialmente durante las etapas finales de prueba. Para ser más específicos, el administrador del proyecto a menudo descubre que necesita una gran cantidad de horas-máquina para probar el sistema; tal vez 12 horas de labor ininterrumpida diaria. Dado que suele ser difícil obtener tanto tiempo de uso de la computadora,[6] el proyecto suele retrasarse mucho.

5.2.2 Progresión Secuencial

La segunda debilidad más importante del ciclo de vida de un proyecto clásico es su insistencia en que las fases se sucedan secuencialmente. Querer esto es una tendencia natural humana: deseamos decir que hemos *terminado* la fase de análisis

6 Estoy convencido de que aquí se aplica otra más de las leyes de Murphy: Entre más grande y más crítico sea el proyecto, más probable es que la fecha límite coincida con el proceso de fin de año o alguna otra crisis organizacional que monopoliza todo el tiempo de computadora disponible.

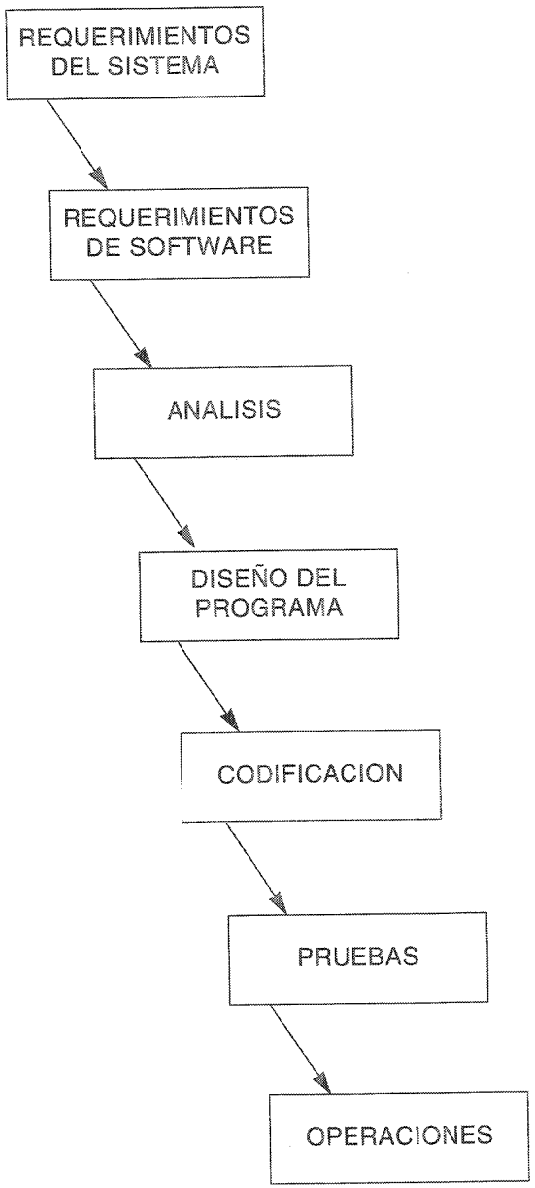

Figura 5.1(b): **El modelo de cascada del desarrollo de sistemas**

del sistema y que nunca tendremos que volver a preocuparnos por ella. De hecho, muchas organizaciones formalizan esto con un ritual conocido como "congelar" la especificación o congelar el documento de diseño.

El único problema que trae consigo este deseo de progreso ordenado es que no es nada realista. En lo particular, el enfoque secuencial no permite el tratamiento de fenómenos reales como los relacionados con el personal, la política de la compañía, o la economía. Por ejemplo, la persona que hizo el trabajo, el analista o el diseñador, pudieron haber cometido un error y haber elaborado un producto con fallas. De hecho, como humanos, rara vez atinamos a hacer bien un trabajo al primer intento, pero se suelen hacer repetidas mejoras del trabajo imperfecto. También pudiera ser que la persona que revisa el trabajo o, como caso particular, el usuario que revisa el trabajo del analista del sistema pudiera haber cometido un error. O tal vez el encargado de llevar a cabo la labor asociada con cada fase no haya tenido tiempo suficiente para terminar, pero no quiera admitirlo. Esta es una manera amable de decir que, en la mayoría de los proyectos complejos, la labor de análisis, de diseño y de prueba concluye cuando alguien decide que se ha agotado el tiempo, no cuando se quisiera.

Comúnmente surgen otros problemas asociados con el ciclo de vida del proyecto clásico o secuencial: durante los meses (o años) que toma desarrollar el sistema, el usuario pudiera cambiar de parecer respecto a lo que debe hacer el sistema. Durante el período que transcurre para desarrollar el sistema, pueden cambiar ciertos aspectos del ambiente del usuario (por ejemplo, la economía, la competencia, los reglamentos gubernamentales que afectan a las actividades del usuario).

Una característica adicional del ciclo de vida del proyecto clásico es que se apoya en técnicas anticuadas. Es decir, tiende a ignorar el uso del análisis estructurado la programación estructurada, o cualquier otra técnica moderna de desarrollo de sistemas.[7] Pero el hecho de que el ciclo de vida clásico *ignore* estas técnicas no significa que el administrador del proyecto no pueda utilizarlas. Desafortunadamente, muchos programadores, analistas y jefes de proyecto sienten que el ciclo de vida del proyecto es un mandato de la administración de alto nivel; y si la administración no dice nada al respecto del uso de la programación estructurada, entonces creen que no están obligados a utilizar métodos no clásicos.

5.3 EL CICLO DE VIDA SEMIESTRUCTURADO

Los comentarios de la sección anterior pueden hacer que parezca que la mayoría de las organizaciones de proceso de datos todavía viven en la Edad Media. De hecho, esto es injusto: no todas las organizaciones utilizan el ciclo de vida clásico. Desde fines de los años 70 y principios de los 80, ha crecido la tendencia a reconocer al diseño estructurado, la programación estructurada y la implantación descendente como parte del ciclo de vida del proyecto. Este reconocimiento ha llevado al ciclo de vida del proyecto semiestructurado que se muestra en la figura 5.2. Se muestran dos detalles obvios no presentes en el enfoque clásico:

1. La secuencia ascendente de codificación, la prueba de módulos y prueba del sistema se reemplazan por una implantación de arriba hacia abajo, que es un enfoque en el cual los módulos de alto nivel se codifican y prueban primero, seguidos por los de bajo nivel, más detallados. También hay fuertes indicios de que la programación estructurada debe usarse como método para codificar el sistema.

2. El diseño clásico se reemplaza por el diseño estructurado, que es un enfoque de diseño formal de sistemas tratado en textos tales como [Yourdon y Constantine, 1989] y [Page-Jones, 1988].

Aparte de estas diferencias obvias, hay algunos detalles sutiles acerca del ciclo de vida modificado. Por ejemplo, considere que la implantación descendente significa que se pondrán en ejecución paralelamente parte de la codificación y de las pruebas. Esto difiere mucho de las fases secuenciales que vimos en el ciclo de vida clásico. En lo particular, puede darse una *retroalimentación* entre la codificación, la prueba y la eliminación de las fallas. Cuando el programador prueba la versión de alto nivel del sistema, a veces se le puede llegar a oír exclamar: "¡Vaya, no tenía idea de que la instrucción FRAMMIS de doble precisión funcionara de esa manera!". Desde luego, se puede tener la seguridad de que en el futuro usará de manera muy diferente esta instrucción.

Tal vez sea aún más importante el hecho de que la implantación descendente pone en tentación a los ejecutores del sistema (y a los analistas si aún no han abandonado el proyecto) de no hablar con los usuarios sino hasta *después* de haberse congelado las especificaciones. Por eso, es posible que el usuario señale errores o malentendidos en la especificación, o incluso pudiera expresar el deseo de *cambiarla* y, si la conversación se da directamente entre el usuario y el que implanta, la modificación pudiera hacerse antes de que el administrador del proyecto se dé cuenta siquiera. En resumen, a menudo la implantación descendente ofrece retroalimentación entre el proceso de implantación y el de análisis, aunque esto no se muestre específicamente en la Figura 5.2, y aunque el usuario y el administrador del proyecto de proceso de datos pudieran negar que esté sucediendo.

Como último punto a tratar acerca del ciclo de vida semiestructurado, tenemos que una gran parte del trabajo que se realiza bajo el nombre de "diseño estructurado" es en realidad un esfuerzo manual para enmendar especificaciones erróneas. Esto se puede apreciar en la figura 5.3, que muestra los detalles del diseño estructurado. (Nótese que esta Figura consiste en los detalles del proceso 3 de la figura 5.2)

En la figura 5.3, la actividad 3.1 (con el título de CODIFICAR LA ESPECIFICACION FUNCIONAL) representa la labor que han tenido que desempeñar desde hace

7 Resumiremos estas técnicas modernas de desarrollo en el capítulo 7.

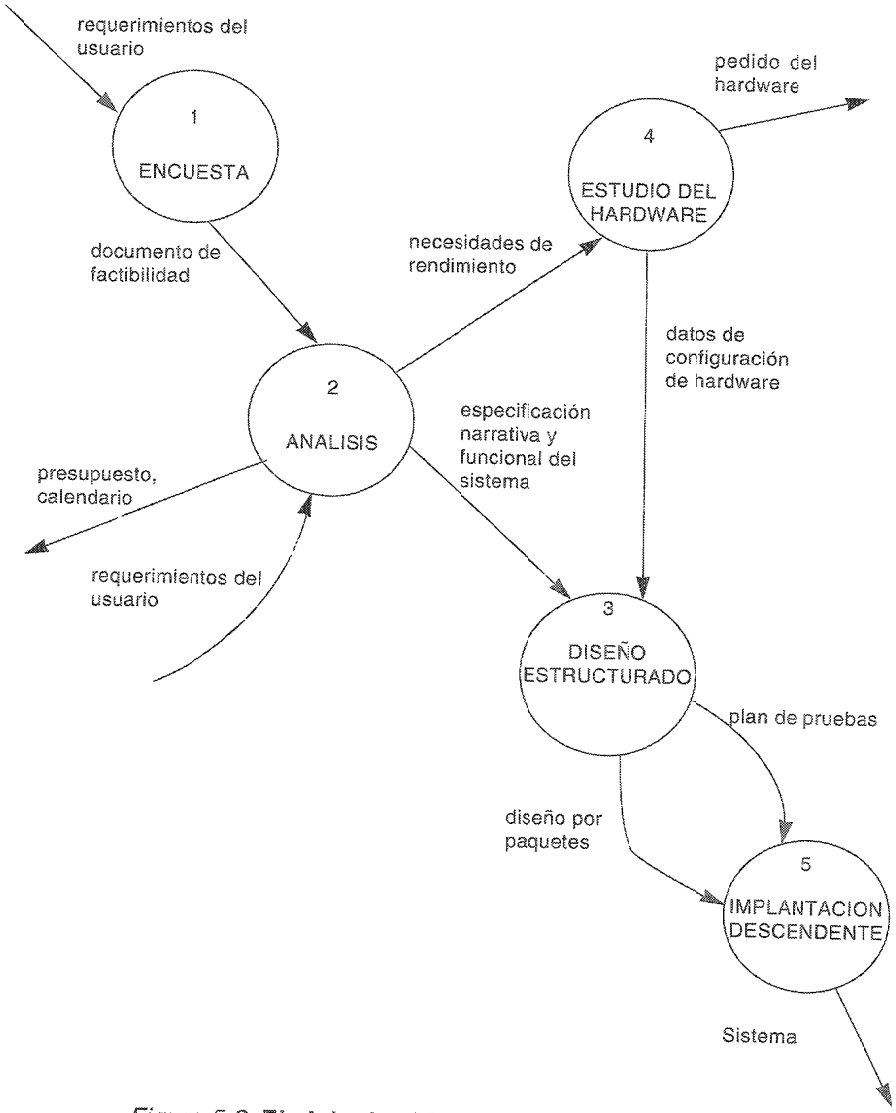

Figura 5.2: **El ciclo de vida del proyecto semiestructurado**

mucho los diseñadores: traducir un documento narrativo, ambiguo, monolítico y redundante a un modelo útil y no de procedimientos, para que sirva de base para derivar la jerarquía de módulos que ejecutarán los requisitos del usuario. En otras palabras, los que llevan a cabo el diseño estructurado han supuesto tradicionalmente que se les daría una especificación clásica; en consecuencia, su primera tarea, desde su punto de vista, es transformar la especificación en un paquete de diagramas de flujo de datos, de diccionarios de datos, de diagramas de entidad relación y de especificaciones de procesos.

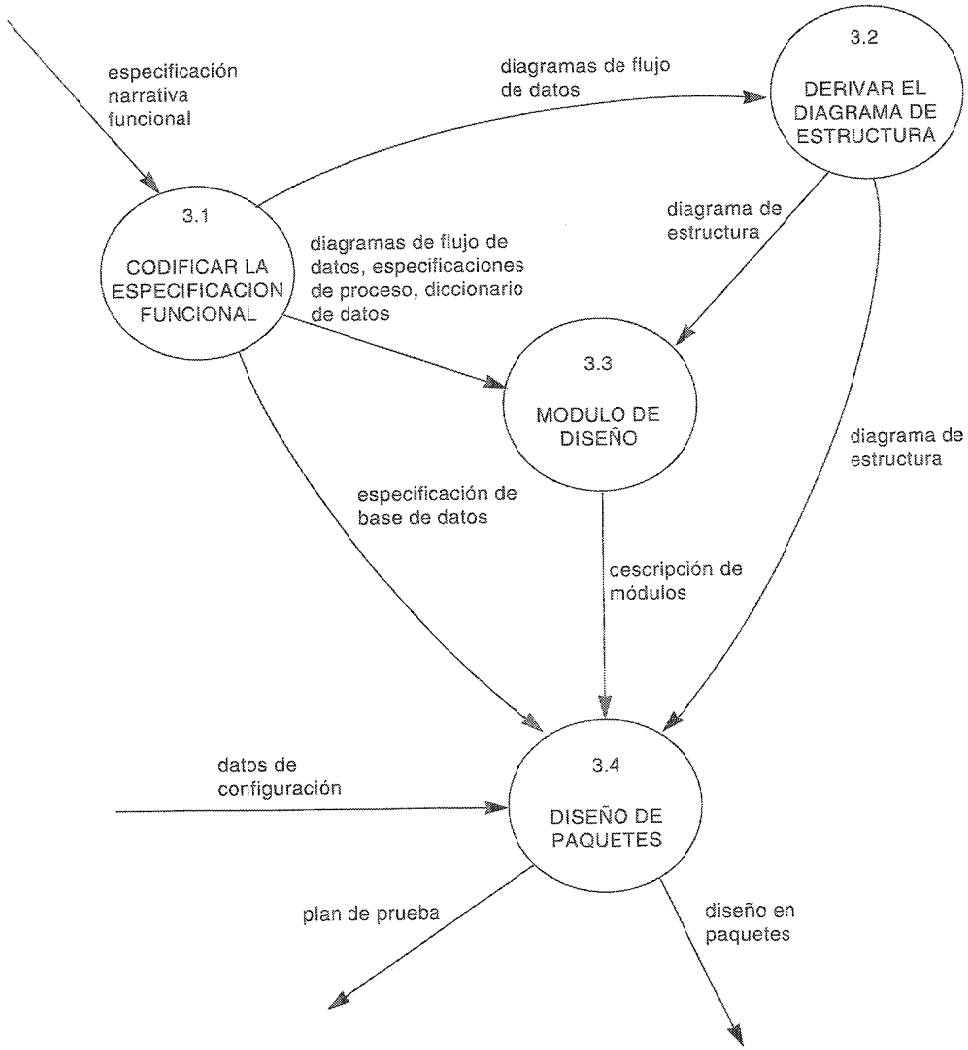

Figura 5.3: **Detalles de la actividad de diseño**

Esta labor es más difícil de lo que se pudiera imaginar: históricamente se ha llevado a cabo en el vacío. En general, los diseñadores tenían poco contacto con el analista que escribía la especificación y definitivamente *¡no* tenían contacto con el usuario!

Es obvio que esta situación amerita un cambio. Al presentar el análisis estructurado, que es el enfoque moderno de análisis de sistemas que se maneja en este libro, además de extenderse con la idea de la retroalimentación entre una parte del proyecto y otra, se crea un tipo totalmente distinto de ciclo de vida del proyecto. Este es el ciclo de vida estructurado del proyecto que discutiremos a continuación.

5.4 EL CICLO DE VIDA ESTRUCTURADO DEL PROYECTO

Ahora que ya hemos visto los ciclos de vida del proyecto clásico y semiestructurado, estamos listos para examinar el ciclo de vida estructurado, que se muestra en la figura 5.4.

Examinaremos brevemente las nueve actividades y los tres terminadores del ciclo de vida del proyecto, como se muestra en la figura 5.4. Los terminadores son los usuarios, los administradores y el personal de operaciones; como se recordará, discutimos sus papeles en el capítulo 3. Se trata de individuos o grupos que proporcionan las entradas al equipo del proyecto, y son los beneficiados finales del sistema. Ellos interactúan con las nueve actividades que se muestran en la figura 5.4 . En las siguientes secciones se resume cada una de las actividades.

5.4.1 Actividad 1: La encuesta

Esta actividad también se conoce como el estudio de factibilidad o como el estudio inicial de negocios. Por lo común, empieza cuando el usuario solicita que una o más partes de su sistema se automaticen. Los principales objetivos de la encuesta son los siguientes:

- *Identificar a los usuarios responsables y crear un "campo de actividad" inicial del sistema.* Esto puede comprender la conducción de una serie de entrevistas para determinar qué usuarios estarán comprendidos en (o serán afectados por) el proyecto propuesto.[8] Pudiera también implicar el desarrollo de un diagrama inicial de contexto, que es un diagrama de flujo de datos sencillo del tipo que se muestra en la figura 4.2, en el cual se representa el sistema completo con un solo proceso.[9]

8 Las técnicas de encuesta se discuten en el Apéndice E.

9 El diagrama de contexto es parte del modelo ambiental que se discutirá con mayor detalle en el capítulo 18. Su principal propósito, como se indica aquí, es definir cuánto abarca el sistema, así como los diversos terminadores (personas, unidades organizacionales, otros sistemas de cómputo, etc.) con los que el sistema interactuará.

- *Identificar las deficiencias actuales en el ambiente del usuario.* Esto en general comprenderá la lista de funciones que hacen falta o que se están llevando a cabo insatisfactoriamente en el sistema actual. Por ejemplo, esto pudiera incluir declaraciones como las siguientes:

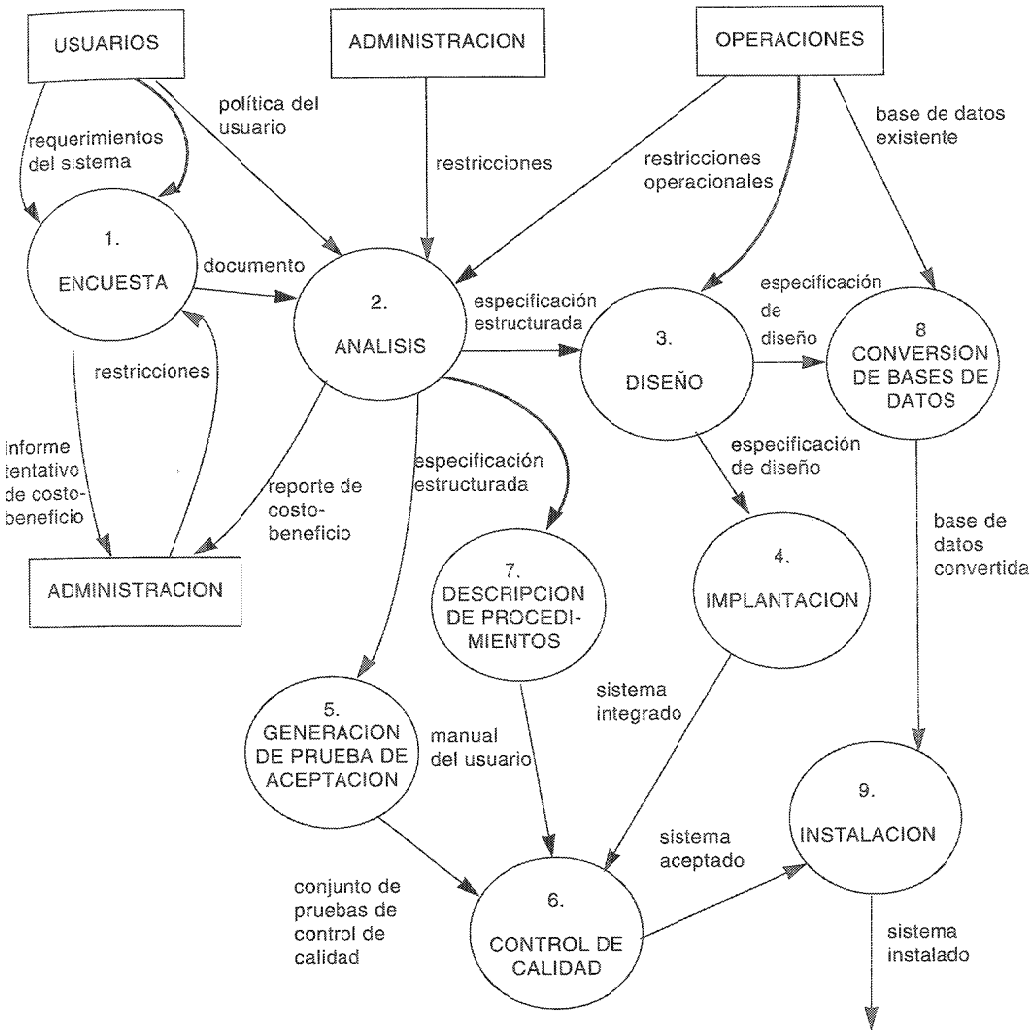

Figura 5.4: **El ciclo de vida del proyecto estructurado**

  - El hardware del sistema actual no es confiable y el vendedor se acaba de declarar en quiebra.
  - El software del sistema actual no se puede mantener, y no podemos ya contratar programadores de mantenimiento dispuestos a darle mantenimiento en el lenguaje que originalmente se utilizó para desarrollarlo.
  - El tiempo de respuesta del sistema telefónico de pedidos actual es tan malo que muchos clientes cuelgan frustrados antes de hacer su pedido.
  - El sistema actual no es capaz de producir los informes requeridos por la modificación a los impuestos decretada el año anterior.
  - El sistema actual no es capaz de recibir los informes sobre límites de crédito del departamento de contabilidad, y no puede producir los informes de promedio de volumen de pedidos que requiere el departamento de mercadotecnia.

- *Establecer metas y objetivos para un sistema nuevo.* Esto puede ser también una simple lista narrativa que contenga las funciones existentes que deben reimplantarse, las nuevas que necesitan añadirse y los criterios de desempeño del nuevo sistema.
- *Determinar si es factible automatizar el sistema y de ser así, sugerir escenarios aceptables.* Esto implicará algunas estimaciones bastante rudimentarias y aproximadas del costo y el tiempo necesarios para construir un sistema nuevo y los beneficios que se derivarán de ello;[10] también involucrará dos o más escenarios (por ejemplo, el escenario con una computadora grande, el de procesamiento distribuido, etc.). Aunque a estas alturas la administración y los usuarios usualmente querrán una estimación precisa y detallada, el analista tendrá mucha suerte si logra determinar el tiempo, los recursos y los costos con un error menor del 50% en esta etapa tan temprana del proyecto.
- *Preparar el esquema que se usará para guiar el resto del proyecto.* Este esquema incluirá toda la información que se lista anteriormente, además de identificar al administrador responsable del proyecto. También pudiera describir los detalles del ciclo de vida que seguirá el resto del proyecto.

En general, la encuesta ocupa sólo del 5 al 10 por ciento del tiempo y los recursos de todo el proyecto, y para los proyectos pequeños y sencillos pudiera ni siquiera ser una actividad formal. Sin embargo, aun cuando no consuma mucho del

10 Los cálculos de costo-beneficio se discutirán en el apéndice C.

tiempo y de los recursos del proyecto, es una actividad verdaderamente importante: al final de la encuesta, la administración pudiera decidir cancelar el proyecto si no parece atractivo desde el punto de vista de costo-beneficio.

Como analista, usted podrá o no estar involucrado en la encuesta; pudiera ser que antes de que siquiera se haya enterado del proyecto, el usuario y los niveles apropiados de la administración ya la hayan hecho. Sin embargo, en proyectos grandes y complejos, la encuesta requiere trabajo tan detallado que a menudo el usuario solicitará la colaboración del analista lo más pronto posible.

No discutiremos la encuesta con mayor detalle en este libro. Si llega a tener que ver con esta actividad, encontrará de utilidad los apéndices E y C. Para detalles adicionales, consulte [Dickinson, 1981], [Gore y Stubbe, 1983] y [Yourdon, 1988].

5.4.2 Actividad 2: El análisis de sistemas

El propósito principal de la actividad de análisis es transformar sus dos entradas –o insumos o factores– principales, las políticas del usuario y el esquema del proyecto, en una especificación estructurada. Esto implica modelar el ambiente del usuario con diagramas de flujo de datos, diagramas de entidad- relación, diagramas de transición de estado y demás herramientas que se presentaron en el capítulo 4. Estas herramientas se tratan con detalle en la parte II.

El *proceso* paso a paso del análisis de sistemas (es decir, las subactividades de la actividad de análisis de la figura 5.4) se discute en la parte III. Como veremos, implica el desarrollo de un *modelo ambiental* (que se trata en el capítulo 18) y el desarrollo de un *modelo de comportamiento* (que se discute en los capítulos 19 y 20). Estos dos modelos se combinan para formar el *modelo esencial* (que se explica en el capítulo 17), que representa una descripción formal de lo que el nuevo sistema debe hacer, independientemente de la naturaleza de la tecnología que se use para cubrir los requerimientos.

Además del modelo del sistema que describe los requerimientos del usuario, generalmente se prepara un conjunto de presupuestos y cálculos de costos y beneficios más precisos y detallados al final de la actividad de análisis. Esto se discute con más detalle en el apéndice C.

Obviamente, como analista del sistema, en esto pasará la mayor parte de su tiempo. No hay nada más que se necesite decir acerca de la actividad de análisis en este momento, ya que ese es el tema que trata todo el resto del libro.

5.4.3 Actividad 3: el diseño

La actividad de diseño se dedica a asignar porciones de la especificación (también conocida como modelo esencial) a procesadores adecuados (sean máquinas o humanos) y a labores apropiadas (o tareas, particiones, etc.) dentro de cada procesador. Dentro de cada labor, la actividad de diseño se dedica a la creación de

una jerarquía apropiada de módulos de programas y de interfases entre ellos para implantar la especificación creada en la actividad 2. Además, la actividad de diseño se ocupa de la transformación de modelos de datos de entidad-relación en un diseño de base de datos; véase [Inmon, 1988] para más detalles.

Parte de esta actividad le interesará como analista: el desarrollo de algo conocido como el *modelo de implantación del usuario*. Este modelo describe los asuntos relacionados con la implantación que le importan al usuario al grado de que no se los quiere confiar a los diseñadores y programadores. Los asuntos principales que suelen preocupar al usuario son aquellos relacionados con la especificación de la frontera humano-máquina y la especificación de la interfaz hombre-máquina. Esa frontera separa las partes del modelo esencial que llevará a cabo una persona (como actividad manual), de las partes que se implantarán en una o más computadoras. De manera similar, la interfaz hombre-máquina es una descripción del formato y de la secuencia de entradas que los usuarios proporcionan a la computadora (por ejemplo, el diseño de pantallas y el diálogo en línea entre el usuario y la computadora), además del formato y la secuencia de salidas –o productos– que la computadora proporciona al usuario. El modelo de implantación del usuario se describe en el capítulo 21.

En el capítulo 22 se puede encontrar una introducción al proceso de diseño de sistemas. Se puede encontrar material adicional en [Yourdon y Constantine, 1989], [Page-Jones, 1988], [Jackson, 1975], y otros.

5.4.4 Actividad 4: implantación

Esta actividad incluye la codificación y la integración de módulos en un esqueleto progresivamente más completo del sistema final. Por eso, la actividad 4 incluye tanto programación estructurada como implantación descendente.

Como podrá imaginar, el analista típicamente no se verá involucrado en esta actividad, aunque hay algunos proyectos (y organizaciones) donde el análisis, el diseño y la implantación de sistemas los hace la misma persona. Este tema se discute más a fondo en el capítulo 23.

5.4.5 Actividad 5: generación de pruebas de aceptación

La especificación estructurada debe contener toda la información necesaria para definir un sistema que sea aceptable desde el punto de vista del usuario. Por eso, una vez generada la especificación, puede comenzar la actividad de producir un conjunto de casos de prueba de aceptación desde la especificación estructurada.

Dado que el desarrollo de las pruebas de aceptación puede suceder al mismo tiempo que las actividades de diseño e implantación, pudiera ser que al analista le sea asignada esta labor al término del desarrollo del modelo esencial en la actividad 2. En el capítulo 23 se discute con más detalle el proceso de prueba.

5.4.6 Actividad 6: garantía de calidad

La garantía de calidad también se conoce como la prueba final o la prueba de aceptación. Esta actividad requiere como entradas los datos de la prueba de aceptación generada en la actividad 5 y el sistema integrado producido en la actividad 4.

El analista pudiera estar involucrado con la actividad de garantía de calidad, pero por lo regular no lo está. Pueden tomar la responsabilidad uno o más miembros de la organización usuaria, o pudiera llevarla a cabo un grupo independiente de prueba o un departamento de control de calidad. Consecuentemente, no se discutirá con más detalle la función de garantía de calidad.

Nótese, por cierto, que algunas personas le llaman a esta actividad "control de calidad" en lugar de "garantía de calidad". Sin importar la terminología, se necesita una actividad que *verifique* que el sistema tenga un nivel apropiado de calidad; le hemos llamado garantía de calidad en este libro. Nótese también que es importante llevar a cabo actividades de garantía de calidad *en cada una* de las actividades anteriores para asegurar que se hayan realizado con un nivel apropiado de calidad. Por eso, se esperaría que esto se haga durante toda la actividad de análisis, diseño y programación para asegurar que el analista esté desarrollando especificaciones de alta calidad, que el diseñador esté produciendo diseños de alta calicad y que el programador este escribiendo códigos de alta calidad. La actividad de garantía de calidad que se menciona aquí es simplemente la prueba *final* de la calidad del sistema.

5.4.7 Actividad 7: descripción del procedimiento

A lo largo de todo este libro nos preocupamos por el desarrollo de un sistema completo: no sólo de la porción automatizada, sino también de la parte que llevarán a cabo las personas. Por ello, una de las actividades importantes a realizar es la generación de una descripción formal de las partes del sistema que se harán en forma manual, lo mismo que la descripción de cómo interactuarán los usuarios con la parte automatizada del nuevo sistema. El resultado de la actividad 7 es un manual para el usuario.

Como podrá imaginar, esta también es una actividad en la que pudiera verse involucrado como analista. Aunque no se discutirá más a fondo en este libro, podría consultar libros acerca de redacción técnica para obtener mayor información sobre la escritura de manuales para el usuario.

5.4.8 Actividad 8: conversión de bases de datos

En algunos proyectos, la conversión de bases de datos involucraba más trabajo (y más planeación estratégica) que el desarrollo de programas de computadora para el nuevo sistema. En otros casos, pudiera no haber existido una base de datos que convertir. En el caso general, esta actividad requiere como entrada la base de datos actual del usuario, al igual que la especificación del diseño producida por medio de la actividad 3.

Según sea de la naturaleza del proyecto, el analista podría tener que ver con la actividad de conversión de la base de datos. Sin embargo no discutiremos esta actividad con mayor detalle en este libro.

5.4.9 Actividad 9: instalación

La actividad final, desde luego, es la instalación; sus entradas son el manual del usuario producido en la actividad 7, la base de datos convertida que se creó con actividad 8 y el sistema aceptado producido por la actividad 6. En algunos casos, sin embargo, la instalación pudiera significar simplemente un cambio de la noche a la mañana al nuevo sistema, sin mayor alboroto; en otros casos, la instalación pudiera ser un proceso gradual, en el que un grupo tras otro de usuarios van recibiendo manuales y entrenamiento y comenzando a usar el nuevo sistema.

5.4.10 Resumen del ciclo de vida del proyecto estructurado

Es importante ver la figura 5.4 como lo que es: un *diagrama de flujo de datos. No* es un diagrama de flujo; nada implica que toda la actividad N debe concluir antes de comenzar la actividad N + 1. Por el contrario, la red de flujos de datos que conectan las actividades hace ver con claridad que pudieran estarse llevando a cabo diversas actividades paralelamente. Debido a este aspecto no secuencial, usamos la palabra *actividad* en el ciclo de vida del proyecto estructurado en lugar de *"fase"*, que es más convencional. El término fase tradicionalmente se refiere a un periodo particular en un proyecto en el cual se estaba desarrollando una, y sólo una, actividad.

Hay otra cosa que debe recalcarse acerca del uso de un diagrama de flujo de datos para describir el ciclo de vida del proyecto: un diagrama de flujo de datos clásico, como el que se muestra en la figura 5.4, no muestra en forma explícita la retroalimentación, ni el control.[11] Prácticamente todas las actividades de la figura 5.4 pueden y suelen producir información que puede llevar a modificaciones adecuadas de una o más de las actividades precedentes. De aquí que la actividad de diseño puede producir información que acaso cambie algunas de las decisiones de costo-beneficio en la actividad de análisis; de hecho, el conocimiento que se obtiene a partir de la actividad de diseño pudiera incluso llevar a revisar decisiones anteriores acerca de la factibilidad básica del proyecto.

Más aún, en casos extremos, ciertos eventos que pudieran darse en cualquier actividad pueden causar que todo el proyecto termine repentinamente. Las entradas de la administración se muestran sólo para la actividad de análisis pues ésta es la única que requiere datos de la administración; sin embargo, se supone, que la administración ejerce *control* sobre todas las actividades.

11 En realidad, hay maneras de mostrar la retroalimentación y el control en los diagramas de flujo de datos, como se verá en el capítulo 9. Las notaciones adicionales (para flujos de control y de procesos de control) normalmente se utilizan para modelar sistemas de tiempo real y hemos evitado su uso en este modelo del "sistema para construir sistemas".

En resumen, la fFigura 5.4 sólo señala la o las entradas requeridas por cada actividad, y la o las salidas o productos que se generan. La secuencia de las actividades sólo puede suponerse en la medida en que la presencia o ausencia de datos haga posible comenzar una determinada actividad.

5.5 IMPLANTACION RADICAL CONTRA IMPLANTACION DESCENDENTE CONSERVADORA

En la sección anterior señalé que el ciclo de vida del proyecto estructurado permite que más de una actividad se lleve a cabo a la vez. Pongámoslo de otra manera: en la situación más extrema, *todas* las actividades del ciclo de vida estructurado pudieran estarse realizando simultáneamente. En el otro extremo, el administrador del proyecto pudiera decidir adoptar el enfoque secuencial, que implica terminar *completamente* una actividad antes de emprender la siguiente.

Es conveniente tener terminología para discutir estos extremos así como los términos medios entre ellos. El enfoque *radical* del ciclo de vida del proyecto estructurado es aquel en el que las actividades 1 a 9 se llevan a cabo paralelamente desde el principio del proyecto: la codificación se inicia el primer día del proyecto, y la encuesta y el análisis continúan hasta el último. En cambio, en el enfoque *conservador* del ciclo de vida del proyecto estructurado, la actividad N completa se termina antes de comenzar con la actividad N + 1.

Obviamente, ningún administrador en sus cabales adoptaría cualquiera de estos dos extremos. La clave para reconocer esto consiste en que los extremos radical y conservador definidos anteriormente son los puntos extremos de una gama de opciones; esto se ilustra en la figura 5.5. Existe un infinito número de opciones entre los extremos radical y conservador. Un administrador de proyecto pudiera decidir terminar el 75% de la actividad de encuesta, seguido por la terminación del 75% del análisis del sistema, y luego del 75% del diseño para poder producir un esqueleto razonable de un sistema cuyos detalles pudieran posteriormente refinarse al pasar por segunda vez por el ciclo de vida entero del proyecto. O bien, el administrador pudiera decidir terminar todas las actividades de encuesta y de análisis, seguido por la terminación del 50% del diseño y el 50% de la implantación. Las posibilidades son interminables.

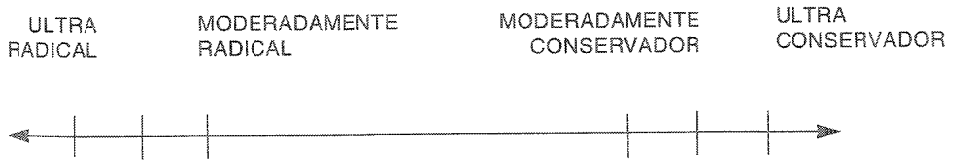

Figura 5.5: **Elecciones de implantación radical y conservadora**

¿Cómo decide un administrador de proyecto si adoptar un enfoque radical o conservador? Básicamente, no hay respuesta; la decisión suele basarse en los siguientes factores:

- ¿Qué tan voluble es el usuario?
- ¿Bajo qué presión labora el equipo del proyecto para producir resultados tangibles e inmediatos?
- ¿Bajo qué presión labora el administrador del proyecto para producir un presupuesto, programa, y estimación de personas y otros recursos?
- ¿Cuáles son los peligros de cometer un error técnico importante?

Como podrá apreciarse, ninguna de estas preguntas puede responderse claramente. Por ejemplo, uno no puede preguntarle al usuario, en una conversación informal, "¿Qué tan voluble andas hoy?". Por otro lado, el administrador del proyecto debiera poder juzgar la situación basándose en la observación, sobre todo si es un veterano que ha lidiado anteriormente con muchos usuarios y administradores de alto nivel.

Si el administrador del proyecto juzga que está tratando con un usuario voluble cuya personalidad es tal que retrasa la toma de decisiones hasta estar seguro de que el sistema va a funcionar, entonces probablemente optaría por un enfoque más radical. Lo mismo si trata con un usuario sin experiencia, a quien le hayan creado pocos sistemas. ¿Por qué pasar años desarrollando un conjunto perfecto de especificaciones tan sólo para descubrir que el usuario no comprendió su significado?

Por otro lado, si el administrador trata con un usuario veterano que está absolutamente seguro de lo que quiere, y si éste último trabaja en un área estable y con poca probabilidad de cambiar radicalmente de un mes a otro, entonces puede darse el lujo de adoptar un enfoque más conservador. Desde luego, hay muchas situaciones intermedias: el usuario puede estar seguro de *algunas* de las funciones de negocios que deberán llevarse a cabo, pero al mismo tiempo no estar seguro del tipo de informes administrativos que desea que el sistema le proporcione. O bien, si el usuario está familiarizado con sistemas computacionales por lotes *(batch)*, podría no estar seguro del impacto que pudiera tener en la empresa un sistema en línea.

Además de la volubilidad, existe un segundo factor que se debe considerar: la presión a la que se está sometido para producir resultados tangibles e inmediatos. Si, debido a las políticas u otras presiones externas, el equipo que realiza el proyecto *debe* concluirlo forzosamente para una fecha determinada, entonces se requiere un enfoque un tanto radical. El administrador del proyecto aún corre el riesgo de que el sistema sólo esté completo en un 90 por ciento para la fecha límite, pero por lo menos será un esqueleto operante completo en un 90 por ciento que puede mostrarse y tal vez incluso ponerse a producir. Eso generalmente es mejor que haber

terminado todo el análisis de sistemas, todo el diseño y toda la codificación, pero nada de las pruebas.

Desde luego, *todos* los proyectos llegan a verse apremiados a llegar a resultados tangibles; la cuestión es el del apremio. Es un asunto que puede ser algo dinámico: un proyecto que comienza holgadamente con un programa cómodo puede de repente volverse de alta prioridad y la fecha límite adelantarse seis meses o un año. Una de las ventajas de hacer el análisis, diseño, codificación e implantación del sistema en forma descendente es que se puede suspender una actividad en cualquier momento y dejar los detalles restantes para consideración posterior; mientras tanto, el análisis de alto nivel que se haya terminado puede usarse para comenzar el diseño de alto nivel, y así para los demás casos.

Otro factor más en la administración de proyectos es el requisito siempre presente en la mayoría de las organizaciones grandes de que se tienen que producir programas, estimaciones, presupuestos, etc. En algunas organizaciones, esto suele hacerse de manera bastante informal, normalmente porque los proyectos son relativamente pequeños y porque la administración siente que cualquier error en la estimación tendrá poco impacto en la organización global. En tales casos se puede adoptar un enfoque radical, aunque cualquier intento de hacer una estimación se tendrá que reducir al nivel de conjeturas viscerales. En cambio, la mayoría de los proyectos requieren estimaciones relativamente detalladas de necesidades de personal, recursos computacionales, etc., y esto sólo se puede realizar tras un sondeo, análisis y diseño bastante detallados. En otras palabras, entre más detalladas y precisas tengan que ser las estimaciones, más probable es que el proyecto siga un enfoque conservador.

Finalmente, el administrador del proyecto debe considerar el peligro de cometer un error técnico importante. Por ejemplo, suponga que toda su experiencia pasada en desarrollo de proyectos ha sido con una pequeña computadora de procesamiento por lotes IBM/36. Ahora, de repente, está a cargo de desarrollar un sistema de multiprocesamiento en línea para administración de bases de datos distribuidas, en tiempo real, que procesará 2 millones de transacciones diarias desde 5000 terminales distribuidas en todo el mundo. En tal situación, uno de los peligros del enfoque radical es descubrir algún error importante en el diseño tras haber realizado una buena parte del esqueleto de alto nivel del sistema.

Pudiera descubrir, por ejemplo, que para que su gran sistema funcione se requiere que un módulo de bajo nivel lleve a cabo su función en 19 microsegundos, pero sus programadores de repente le informan que es imposible codificar un módulo con tanta eficiencia, ni en COBOL, ni en C, ni siquiera (¡uf!) en lenguaje ensamblador. Por lo tanto, debe estar alerta al hecho de que seguir un enfoque radical requiere que sus analistas y diseñadores escojan un "tope máximo" para su sistema en etapa relativamente temprana, y que siempre existe el peligro de descubrir, ya cerca del final, que escogieron un máximo equivocado.

Sin embargo, considere otra situación: el administrador del proyecto ha decidido construir un sistema electrónico de proceso de datos con equipo nuevo, sistema operativo nuevo, sistema de administración de bases de datos nuevo (producido por alguien que no sea el vendedor), y un paquete de telecomunicaciones nuevo (producido por otra empresa más). Todos los proveedores tienen manuales brillantes e impresionantes que describen sus productos, pero nunca han probado la interfaz entre sus respectivos productos de hardware y software. ¿Quién sabe si siquiera funcionarán juntos? ¿Quién sabe si las funciones prometidas por un proveedor queden anuladas por los recursos del sistema que utiliza el otro? Ciertamente, en un caso como éste, el administrador del proyecto pudiera elegir un enfoque radical, para que la versión esqueleto o primaria del sistema pueda utilizarse para explorar los posibles problemas de interacción e interfaz entre los componentes de los diferentes proveedores.

Si el administrador del proyecto está a cargo de un tipo familiar de sistema como, por ejemplo, su nonagésimo noveno sistema de nóminas, probablemente tenga bastante idea de qué tan realistas sean sus metas. Es posible que recuerde, de su proyecto anterior, qué tipo de módulos necesitará a nivel detallado, y probablemente recuerde con claridad cómo se veía la estructura de alto nivel del sistema. En tal caso, pudiera estar dispuesto a correr el riesgo de cometer un error dados los demás beneficios que puede traerle un enfoque racical.

En resumen, el enfoque radical es el más adecuado para intentos apenas disfrazados de investigación y desarrollo, en los que nadie está muy seguro de qué es lo que se supone que debe hacer el sistema final. Y es bueno para los casos en los que para determinada fecha algo *tiene que* estar ya funcionando, y en situaciones en las que la percepción del usuario respecto a lo que desea que el sistema haga esté sujeta a posibles cambios. El enfoque conservador, por otro lado, suele usarse en proyectos más grandes, en los que se invierten cantidades enormes de dinero y para los cuales se requiere un análisis y diseño muy detallados para evitar desastres subsecuentes. Sin embargo, cada proyecto es diferente y requiere de su propia combinación de implantación descendente conservadora y radical. Para tratar la naturaleza individual de cada proyecto, el administrador debe estar dispuesto a modificar su enfoque en mitad del camino si es necesario.

5.6 EL CICLO DE VIDA DE PROTOTIPOS

Se ha vuelto popular en los últimos años una variación del enfoque descendente antes discutido. En general se le conoce como el enfoque de *prototipos* y lo popularizaron Bernard Boar, James Martin y otros. Como lo describe Boar [Boar, 1984]:

> Una alternativa de enfoque para la definición de los requerimientos consiste en capturar un conjunto inicial de necesidades e implantarlas rápidamente con la intención declarada de expandirlas y refinarlas iterativamente al ir aumentando la comprensión que del sistema tienen el usuario y quien lo desarrolla. La definición del sistema se realiza mediante el descubrimiento evolutivo y gradual y no a través de la previsión omnisciente... Este tipo de enfoque se llama "de prototipos". También se le conoce como modelado del sistema o desarrollo heurístico. Ofrece una alternativa atractiva y practicable a los métodos de especificación para tratar mejor la incertidumbre, la ambigüedad y la volubilidad de los proyectos reales.

Por muchas razones, esto suena exactamente como el enfoque descendente radical que se discutió en la sección anterior. La principal diferencia es que el enfoque estructurado que se discute a lo largo de este libro supone que tarde o temprano se construirá *un modelo en papel completo del sistema* (es decir, un juego completo de diagramas de flujo de datos, de diagramas entidad-relación, de diagramas de transición de estados, de especificaciones de procesos, etc.). El modelo se completará más pronto con un enfoque conservador y más tarde con uno radical, pero para el final del proyecto habrá un juego formal de documentos que deberán permanecer siempre con el sistema, a lo largo de su corrección y mantenimiento.

El enfoque de prototipos, por otro lado, casi siempre supone que el modelo será operante, es decir, una colección de programas de computadora que simularán algunas o todas las funciones que el usuario desea. Pero dado que se pretende que dichos programas sean sólo de modelo, también se supone que *al concluirse el modelado, los programas se descartarán y se reemplazarán con programas REALES.* Quienes hacen prototipos generalmente usan los siguientes tipos de herramientas de software:

- Un diccionario de datos integrado
- Un generador de pantallas
- Un generador de reportes no guiado por procedimientos
- Un lenguaje de programación de cuarta generación
- Un lenguaje de consultas no guiado por procedimientos
- Medios poderosos de administración de bases de datos

El ciclo de vida de prototipos propuesto por Boar se muestra en la figura 5.6. Comienza con una actividad de sondeo, similar a la que propone este libro. De esto sigue inmediatamente una determinación de si el proyecto es un buen candidato para un enfoque de prototipos. Los buenos candidatos son aquellos que tienen las siguientes características:

- El usuario no puede o no está dispuesto a examinar modelos abstractos en papel, tales como diagramas de flujo de datos.
- El usuario no puede o no está dispuesto a articular (o "pre-especificar") sus requerimientos de ninguna forma y sólo se pueden determinar sus requerimientos mediante un proceso de tanteo, o ensayo y error. O, como

lo dice mi colega Bob Spurgeon, es la situación en la que el usuario dice: "No sé qué es lo que quiero, pero lo reconoceré cuando lo vea".

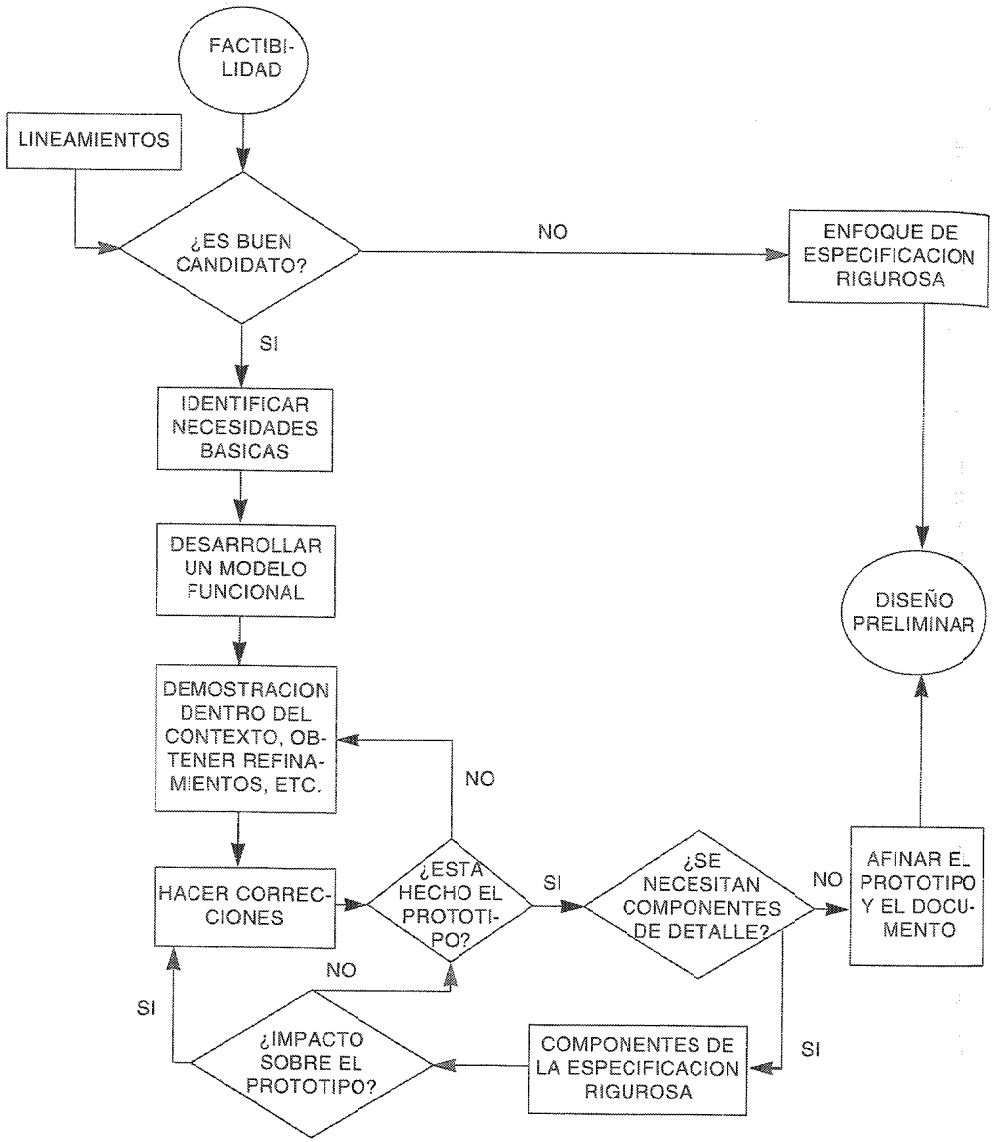

Figura 5.6: **El ciclo de vida por prototipos**

- Se tiene la intención de que el sistema sea en línea y con operación total por la pantalla, en contraposición con los sistemas de edición, actualización y reportes operados por lotes. (Casi todas las herramientas de software de prototipos apuntan al enfoque orientado a terminales en línea y manejadas por bases de datos; existen pocas herramientas de software en el mercado para ayudar a la creación de prototipos de sistemas de procesamiento por lotes.)

- El sistema *no* requiere la especificación de grandes cantidades de detalles algorítmicos, ni de muchas especificaciones de procesos para describir los algoritmos con los cuales se obtienen los resultados. Por ello, los sistemas de apoyo a decisiones, de recuperación ad hoc (a propósito) y de administración de registros son buenos candidatos para el prototipo. Los buenos candidatos suelen ser sistemas en los cuales el usuario se preocupa más por el formato y distribución de los datos de entrada y salida en la pantalla, y por los mensajes de error, que por los cómputos que realiza el sistema para lograrlo.

Es importante notar que el ciclo de vida de prototipos que se muestra en la figura 5.6 concluye con una fase de diseño de un ciclo de vida estructurado "tradicional" como los que describe este libro. Específicamente, esto significa que *no se tiene la intención* de que el prototipo haga las veces de un sistema operacional; la intención es tan sólo que modele los requerimientos del usuario.

El enfoque de prototipos ciertamente tiene su mérito en muchas situaciones. En algunos casos, el administrador del proyecto tal vez quiera utilizar este enfoque como alternativa al de análisis estructurado que se presenta en este libro; en otros casos, pudiera desear utilizarlo *en conjunto* con la creación de modelos en papel, como los diagramas de flujo de datos. Tenga en mente lo siguiente:

- El enfoque descendente descrito en la sección anterior es otra manera de hacer un prototipo, pero en vez de usar herramientas que se pueden obtener en el mercado, como generadores de pantallas y lenguajes de cuarta generación, el equipo que realiza el proyecto utiliza el sistema mismo como su propio prototipo. Es decir, las diversas versiones de un esqueleto del sistema proveen un modelo operativo con el cual el usuario puede interactuar y darse así una idea más realista de las funciones del sistema que la que se pudiera formar a partir de un modelo en papel.

- El ciclo de vida de prototipos, como se describió anteriormente, involucra el desarrollo de un modelo funcional, que luego se descarta y se reemplaza con un sistema de producción. Existe un peligro considerable de que el usuario o el equipo que desarrolla el sistema traten de convertir al prototipo mismo en un sistema de producción. Esto suele resultar un desastre, pues el prototipo no puede trabajar eficientemente con grandes volúmenes de transacciones, y porque carece de detalles operacionales

tales como recuperación de errores, auditorías, características de respaldo/reinicio, documentación para el usuario y procedimientos de conversión.

- Si de hecho se descarta el prototipo y se reemplaza con el sistema de producción, existe el peligro real de que pudiera concluirse el proyecto sin dejar un registro permanente de los requerimientos del usuario. Esto probablemente dificulte cada vez más el mantenimiento con el paso del tiempo (por ejemplo, diez años después de la construcción del sistema, será difícil que los programadores de mantenimiento incorporen algún cambio, pues nadie, incluyendo a los usuarios de "segunda generación" que están trabajando actualmente con el sistema, recordará lo que se suponía en primer lugar que debía hacer). El ciclo de vida que se presenta en este libro se basa en la idea de que los modelos en papel desarrollados durante la actividad de análisis no sólo serán una entrada para la actividad de diseño, sino que también se conservarán (y se modificarán según vaya siendo necesario) durante el mantenimiento. De hecho, los modelos pudieran sobrevivir más allá del sistema en el cual se implantaron, y pudieran servir como especificación para el sistema de reemplazo.

5.7 RESUMEN

El principal propósito de este capítulo fue proporcionar una visión global de los ciclos de vida de los proyectos en general. Si examina el ciclo de vida formal de proyectos en cualquier organización de desarrollo de sistemas, debería poder distinguir si se trata de uno clásico, semiestructurado, estructurado, o de prototipos.

Si su proyecto sólo permite una actividad a la vez, la discusión sobre implantación descendente radical y conservadora de la Sección 5.6 puede haberlo perturbado. Este fue mi propósito, y el principal objetivo de esa sección fuéehacerle pensar acerca de la *posibilidad* de traslapar algunas de las principales actividades en el proyecto de desarrollo de un sistema. Obviamente, es más difícil administrar un proyecto en el cual diversas actividades se llevan a cabo en paralelo, pero, hasta cierto punto, eso sucede en todo proyecto. Aún si el administrador decide que su gente se concentrará en una actividad a la vez, de todos modos habrá varias subactividades que se llevarán a cabo en paralelo. Múltiples analistas de sistemas estarán entrevistando simultáneamente a múltiples usuarios; diversas piezas del producto final del análisis se encontrarán en diversas etapas de progreso a lo largo de toda la fase de análisis. Una labor del administrador es tener el suficiente control sobre dichas subactividades como para asegurar que se coordinen propiamente. Y en casi cualquier proyecto de proceso electrónico de datos, este tipo de actividad paralela se da también a alto nivel; es decir, a pesar de lo que pueda haber recomendado el ciclo de vida formal del proyecto de una organización dada, la realidad es que muchas de las principales actividades del proyecto sí se traslapan hasta cierto punto. No obstante, si el administrador decide insistir en una progresión de actividades estrictamente secuencial, aún funcionará el ciclo de vida presentado por este libro.

Para obtener mayores detalles acerca de ciclos de vida de proyectos, consulte [Dickinson, 1981] y [Yourdon, 1988]. También cubren este concepto una variedad de libros de ingeniería de software y de libros de administración de proyectos.

REFERENCIAS

1. Edward Yourdon y Larry L. Constantine, *Structured Design: Fundamentals and Applications in Software Engineering,* 2a. edición, Englewood Cliffs, N.J.: YOURDON Press, 1988.
2. Meilir Page-Jones, *The Practical Guide to Structured Systems Design,* 2ª edición Englewood Cliffs, N.J.:YOURDON Press, 1988.
3. Bernard Boar, *Application Prototyping.* Reading, Mass.: Addison-Wesley, 1984.
4. James Grier Miller, *Living Systems.* Nueva York: McGraw- Hill, 1978.
5. Brian Dickingson, *Developing Structured Systems.* Nueva York: YOURDON Press, 1981.
6. Edward Yourdon, *Managing the Systems Life Cycle,* 2ª edición, Englewood Cliffs, N.J.: Prentice-Hall, 1988.
7. James Grier Miller, *Living Systems.* Nueva York: McGraw- Hill, 1978.
8. Michael Jackson, *Principles of Program Design.* Nueva York: Academic Press, 1975.
9. Winston W. Royce, "Managing the Development of Large Software Systems", *Proceedings, IEEE Wescon,* agosto 1970, pp. 1-9.
10. Barry Boehm, *Software Engineering Economics.* Englewood Cliffs, N.J.: Prentice-Hall, 1981.
11. Bill Inmon, *Information Engineering for the Practitioner: Putting Theory into Practice.* Englewood Cliffs, N.J.: Prentice-Hall, 1988.
12. Marvin Gore y John Stubbe, *Elements of Systems Analysis,* 3ª edición, Dubuque, Iowa: William Brown, 1983.

PREGUNTAS Y EJERCICIOS

1. Mencione dos sinónimos de metodología.
2. ¿Cuál es la diferencia entre una herramienta, como se utiliza en este libro, y una metodología?
3. ¿Cuáles son los tres principales propósitos del ciclo de vida de un proyecto?

4. Proyecto de investigación: Encuentre el precio de tres productos para metodología comerciales ofrecidos por proveedores de software o empresas consultoras.

5. ¿Por qué normalmente las organizaciones pequeñas de proceso de datos no usan metodologías formales?

6. ¿Por qué es útil para los administradores nuevos una metodología?

7. ¿Por qué es importante tener una metodología en una organización en la que se estén llevando a cabo muchos proyectos diferentes?

8. ¿Cómo es que una metodología es útil para controlar proyectos?

9. ¿Cuáles son las principales características que distinguen el ciclo de vida clásico?

10. ¿Qué significa la expresión implantación ascendente?

11. ¿Cuáles son las cuatro principales dificultades con la estrategia de implantación ascendente?

12. ¿Qué tipo de ambiente es el adecuado para un enfoque de implantación ascendente?

13. ¿Por qué es importante que "nada está hecho hasta que todo esté hecho", que además es lo que caracteriza al enfoque ascendente?

14. ¿Por qué debieran encontrarse los errores triviales primeramente en la fase de prueba de un proyecto?

15. ¿Qué diferencia hay entre prueba y eliminación de errores?

16. ¿Por qué es difícil la eliminación de errores en una implantación ascendente?

17. ¿Qué se entiende por la frase "progresión secuencial" cuando se describe el ciclo de vida de un proyecto?

18. ¿Cuáles son los dos principales problemas de la progresión secuencial?

19. ¿Cuáles son las principales diferencias entre el ciclo de vida semiestructurado y el clásico?

20. ¿Cuáles son las dos principales consecuencias del enfoque de la implantación descendente?

21. ¿Por qué, en el ciclo de vida semiestructurado, el diseño a menudo involucra trabajo redundante?

22. ¿Cuáles son las principales diferencias entre los ciclos de vida semiestructurado y estructurado?

23. Nombre las nueve actividades del ciclo de vida estructurado del proyecto.

24. ¿Quiénes son los tres tipos de personas que proveen de entradas primarias al ciclo de vida del proyecto?

25. ¿Cuáles son los cinco principales objetivos de la actividad de la encuesta?

26. ¿Qué es un diagrama de contexto?

27. ¿Cuál es el principal propósito de la actividad de análisis?

28. ¿Cuáles son los tipos de modelos producidos por la actividad de análisis?

29. ¿Cuál es el propósito de la actividad de diseño?

30. ¿Cuáles son los dos asuntos principales que normalmente le preocupan al usuario en la actividad de diseño?

31. ¿Cuándo puede comenzar la generación de pruebas de aceptación? (actividad 5)

32. ¿Cuál es el propósito de la actividad de descripción del procedimiento?

33. ¿Por qué se utilizó un diagrama de flujo de datos en la Figura 5.4 para mostrar el ciclo de vida del proyecto?

34. ¿Cuál sería un posible sinónimo para la palabra actividad?

35. ¿Por qué es importante la retroalimentación en el ciclo de vida estructurado del proyecto?

36. ¿Qué diferencia hay entre los enfoques radical y conservador para el ciclo de vida estructurado del proyecto?

37. ¿Cuáles son los cuatro principales criterios para elegir el enfoque radical vs. el enfoque conservador?

38. ¿Se le ocurre algún criterio adicional para elegir un enfoque radical vs. un enfoque conservador?

39. ¿Qué tipo de enfoque (radical vs. conservador) debe escoger el administrador de un proyecto si es probable que el usuario cambie de opinión respecto a los requerimientos del sistema?

40. ¿Qué tipo de enfoque (radical vs. conservador) debe escoger el administrador del proyecto si tiene una gran presión de tiempo?

41. ¿Qué tipo de enfoque (radical vs. conservador) debe escoger el administrador del proyecto si se encuentra con riesgos técnicos importantes?

42. ¿Qué diferencia existe entre el ciclo de vida de prototipos y el radical?

43. ¿Qué características tiene el proyecto de prototipos ideal?

44. ¿Qué clase de herramientas se requieren típicamente para un proyecto de prototipos?

45. ¿Por qué no son generalmente buenos candidatos para proyectos de prototipo los sistemas por lotes?

46. ¿Cuáles son los peligros del enfoque de prototipos?

47. ¿Cómo pueden utilizarse en combinación en un proyecto los ciclos de vida estructurado y de prototipos?

6 ASPECTOS IMPORTANTES EN EL DESARROLLO DE SISTEMAS

Los dogmas del tranquilo pasado son inadecuados para el borrascoso presente. La ocasión está atiborrada de dificultades y debemos estar a la altura. Como nuestro caso es nuevo, debemos pensar y actuar en forma novedosa. Debemos desenredarnos, y entonces salvaremos a nuestra nación.

Abraham Lincoln,
Segundo Mensaje Anual al Congreso

En este capítulo se aprenderá:

1. Por qué la productividad es un asunto importante
2. Las soluciones comunes al problema de la productividad
3. El número de errores en un sistema típico
4. La relación entre la edad de un sistema y el número de errores encontrados

Como analista de sistemas, formará parte de un equipo de personas cuyo propósito es desarrollar un sistema de información útil y de alta calidad, que cubrirá las necesidades del usuario final. Al llevar a cabo su trabajo, usted y sus compañeros miembros del equipo sin duda se verán influenciados por las siguientes importantes cuestiones: